KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Vilius Lengvensas

Steganografijos metodas, naudojant HTTP/2 protokolą

Baigiamasis magistro darbas

Vadovas
Prof. Algimantas Venčkauskas

KAUNAS, 2019

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Steganografijos metodas, naudojant HTTP/2 protokolą

Baigiamasis magistro darbas
Informacijos ir informacinių technologijų sauga (kodas 6211BX008)

Vadovas
(parašas) Prof. Algimantas Venčkauskas
(data)

Recenzentas
(parašas) Doc. dr. Rasa Brūzgienė
(data)

Projektą atliko
(parašas) Vilius Lengvensas
(data)

KAUNAS, 2019

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos

(Fakultetas)

Vilius Lengvensas

(Studento vardas, pavardė)

Informacijos ir informacinių technologijų sauga, 6211BX008

(Studijų programos pavadinimas, kodas)

„Steganografijos metodas, naudojant HTTP/2 protokolą“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 19 m. gegužės d.

Kaunas

Patvirtinu, kad mano, **Viliaus Lengvenso**, baigiamasis projektas tema „Steganografijos metodas, naudojant HTTP/2 protokolą“ yra parašytas visiškai savarankiškai, o visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

Lengvensas, V. „Steganografijos metodas, naudojant HTTP/2 protokolą“. Magistro baigiamasis projektas / vadovas prof. Algimantas Venčkauskas; Kauno technologijos universitetas, informatikos fakultetas, kompiuterių katedra.

Kaunas, 2019. 70 p.

SANTRAUKA

Steganografiniai metodai, suteikia galimybes paslėpti duomenis įvairiuose nešliuose, vienas iš jų – tinklo protokolai. Skirtingo lygio TCP/IP modelio protokoluose galima sudaryti slaptus kanalus duomenims pernešti, panaudojus protokolo laukus, laikines protokolų komunikavimo ypatybes ar abu. Šiame darbe, panaudojamas taikymo lygio protokolas – HTTP/2, sudaryti slaptą kanalą panaudojus jo kadrų laukus. Dažnai, slaptų kanalų aptikimui naudojamas entropijos skaičiavimas, šiame darbe patikrinamas tokios steganalizės efektyvumas tiriant HTTP/2 srautą. Darbo pabaigoje, taip pat, pateikiamas HTTP/2 steganografinio metodo programinis kodas demonstraciniams tikslams.

Lengvensas, Vilius. *Steganographic Method Based on HTTP/2 Protocol*: *Master‘s* thesis in Information and Information Technology Security / supervisor assoc. prof. Algimantas Venčkauskas. The Faculty of Informatics, Kaunas University of Technology.

Research area and field: *Network steganography and the design of steganographic methods based on network protocols*

Key words: *steganography, TCP/IP, HTTP/2, protocols, covert channel, storage method, entropy*

Kaunas, 2019. 70 p.

SUMMARY

Steganographic methods are means to hiding data inside various carriers. Network protocols are such carriers. TCP/IP model encompasses all network protocols, many of them can be used to create covert channels which transmit the secret data. The covert channels are created by modifying protocol specific fields, timing features or both. In this paper, HTTP/2, an application layer protocol, is used to create a covert channel by modifying its frame fields. One of the most prominent steganalysis methods for detecting messages hidden using steganography is calculation of Shannon’s entropy, in this paper, its effectiveness is measured against HTTP/2 stream. The source code of HTTP/2 steganographic method is also provided at the end of the paper to act as a demonstration.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

OSI modelis (angl. *Open Systems Interconnection model*) – modelis abstrakčiai aprašantis ryšio ir kompiuterinių tinklų komunikacines funkcijas.
HTTP (angl. *Hypertext Transfer Protocol*) – pagrindinis protokolas duomenų perdavimui pasauliniame tinkle.
HTTPS (angl. *Hypertext Transfer Protocol Secure*) – HTTP praplėtimas saugiai komunikacijai.
DNS (angl. *Domain Name System*) – hierarchinė, decentralizuota pavadinimų suteikimo sistema kompiuteriams, servisams ir kitiems resursams sujungtiems internete arba vietiniame tinkle.
FTP (angl. *File Transfer Protocol*) – protokolas skirtas persiųsti failus tarp kliento ir serverio kompiuteriniame tinkle.
IMAP (angl. *Internet Message Access Protocol*) – protokolas, kurį elektroninio pašto klientai naudoja gauti elektroninio pašto žinutes.
POP (angl. *Post Office Protocol*) – protokolas, kurį elektroninio pašto klientai naudoja gauti elektroninio pašto žinutes.
SMTP (angl. *Simple Mail Transfer Protocol*) – komunikacinis protokolas skirtas elektroninio pašto žinučių perdavimui.
NFS (angl. *Network File System*) – failų sistemos protokolas leidžiantis pasiekti failus per tinklą.
RPC (angl. *Remote Procedure Call*) – protokolas leidžiantis tinklu duoti užklausas į servisus esančius kituose kompiuteriuose.
TCP (angl. *Transmission Control Protocol*) – protokolas patikimai persiunčiantis duomenis tinklu.
UDP (angl. *User Datagram Protocol*) – protokolas persiųsti duomenis tinklu greitai, bet nebūtinai patikimai.
IP (angl. *Internet Protocol*) – esminis interneto protokolas, kitų protokolų duomenų perdavimui ir kompiuterinių tinklų susiejimui.
IPv4 (angl. *Internet Protocol version 4*) – ketvirta IP versija.
IPv6 (angl. *Internet Protocol version 6*) – šešta IP versija.
ICMP (angl. *Internet Control Message Protocol*) – pagalbinis protokolas, tinklo įrenginių naudojamas siųsti klaidų pranešimus ir kitas pasiekiamumą nurodančias žinutes.
ICMPv6 (angl. *Internet Control Message Protocol version 6*) – šešta ICMP versija.
ARP (angl. *Address Resolution Protocol*) – komunikacinis protokolas skirtas aptikti žemo lygio adresus.
ISN (angl. *Initial Sequence Number*) – pradinis eilės numeris, kurį klientas sugeneruoja pradėdamas TCP jungtį.
URG (angl. *Urgent*) – vėliava TCP protokole, skirta gavėjams pranešti, kad kai kurie duomenys segmente turėtų būti prioretizuojami.
ACK (angl. *Acknowledgement*) – vėliava TCP protokole, skirta patvirtinti gautus duomenis.
PSH (angl. *Push*) – vėliava TCP protokole, skirta perduoti duomenis jų nekaupiant į buferį.
RST (angl. *Reset*) – vėliava TCP protokole, skirta atšaukti sujungimą įvykus klaidai.
SYN (angl. *Synchronize*) – vėliava TCP protokole, skirta pradėti sujungimą.
FIN (angl. *Finish*) – vėliava TCP protokole, skirta nutraukti sujungimą.
DF (angl. *Don't Fragment*) – vėliava IP protokole, skirta nustatyti ar duomenis galima fragmentuoti.
MF (angl. *More Fragments*) – vėliava IP protokole, skirta nustatyti ar tai paskutinis duomenų fragmentas.
MTU (angl. *Maximum Transmission Unit*) – nustato didžiausią perduodamų duomenų dydį vienu tinklo perdavimu.
IDS (angl. *Intrusion Detection System*) – prietaisas arba programinis įrankis, kuris stebi tinklą ir jame aptinka piktavališkus veiksmus arba nustatytos politikos pažeidimus.
PIN (angl. *Personal Identification Number*) – trumpas, iš skaičių sudarytas, slaptažodis autentifikuoti vartotoją.
MAC (angl. *Media Access Control*) – unikalus identifikatorius priskirtas prietaiso tinklo plokštei.
CRC (angl. *Cyclic Redundancy Check*) – klaidų aptikimo kodas naudojamas tinkluose ir talpos prietaisuose aptikti netyčinius pakitimus duomenyse.

IV (angl. *Initialization Vector*) – nustatyto dydžio pradinė įvestis kriptografinėje schemoje, kuri dažniausiai privalo būti atsitiktinė.
DES (angl. *Data Encryption Standard*) – simetrinio rakto algoritmas duomenų užšifravimui.
MD5 (angl. *MD5 message-digest algorithm*) – santraukos funkcija sudaranti 128 bitų santrauką.
WEP (angl. *Wired Equivalent Privacy*) – apsaugos algoritmas bevieliuose tinkluose.
RFC (angl. *Request for Comments*) – informacinių ir komunikacinių technologijų publikacija.
CLI (angl. *Command-line interface*) – komandinės eilutės sąsaja.
GUI (angl. *Graphical user interface*) – grafinė vartotojo sąsaja.

ĮVADAS

„Steganografijos metodas, naudojant HTTP/2 protokolą“ – informacijos ir informacinių technologijų saugos programos, kodas – 6211BX008, magistro baigiamasis darbas.

Darbo problematika ir aktualumas

Tinklo steganografija, tai jauna, dar nepilnai ištyrinėta steganografijos šaka. Nors ši steganografijos šaka dar, tik vystosi, šia sritimi susidomėję ekspertai jau rado įvairius būdus paslėpti duomenis daugelyje TCP/IP modelio protokolų, iš kurių labiausiai išbandyti antrojo ir trečiojo lygio protokolai. Atkreipiant dėmesį, kad skiriamas mažesnis dėmesys bandymams sudaryti slaptus kanalus aukščiausio lygio protokoluose, šiuo darbu, siekiama panagrinėti steganografijos galimybes taikomajame tinklo protokolų lygyje, specifiškai, dar gan naujame protokole – HTTP/2.

Darbo tikslas ir uždaviniai

Išsiaiškinus, kad darbo rašymo metu (2017m.), dar nebuvo praktiškai įgyvendintas steganografijos metodas panaudojus HTTP/2 protokolą, sudarytas darbo tikslas – realizuoti steganografinį metodą HTTP/2 protokolu. Sėkmingam, šio tikslo pasiekimui, iškelti uždaviniai:

1. Atlikti literatūros analizę steganografijos tema;
2. Išanalizuoti steganografijos metodų veikimą skirtinguose TCP/IP lygiuose;
3. Išsiaiškinti HTTP/2 protokolo struktūrą ir skirtingų kadrų ypatybes;
4. Sumodeliuoti HTTP/2 steganografinį metodą modeliavimo kalba;
5. Realizuoti HTTP/2 steganografinio metodo prototipą programiškai;
6. Ištirti realizuoto steganografinio metodo talpą, neaptinkamumą ir tvirtumą.

Darbo rezultatai ir jų svarba

Programiškai realizavus ir ištyrus HTTP/2 steganografinį metodą, pirmiausiai buvo įrodyta, kad protokolu siųsti duomenis, ne tam skirtose protokolo laukuose yra įmanoma. Antra, bandymai parodė, kad steganalizės būdas, kaip, kad entropijos skaičiavimas gali nebūtinai įrodyti slapto kanalo būvimą. Entropijos skaičiavimas aptikti slaptus kanalus efektyvus yra, tik tuo atveju, jei žinoma, kad įprastame HTTP/2 sraute nenaudojamos kai kurios HTTP/2 kadrų funkcijos.

Nors HTTP/2 protokolas, dalinai, buvo projektuojamas su idėja ištaisyti, savo pirmtako saugos spragas, tačiau, galimybių kaip protokolą išnaudoti slaptų kanalų kūrimui vis tiek daug.

Darbo struktūra

Darbas sudarytas iš keturių pagrindinių skyrių:

1. „Steganografija“, skyrius skirtas steganografijos apibrėžimui ir steganografijos metodų, įvairiuose TCP/IP modelio lygiuose, analizei;
2. „Steganografinis metodas HTTP/2 protokolu“, skyrius skirtas HTTP/2 steganografinio metodo aprašui ir modeliavimui;
3. „HTTP/2 steganografinio metodo tyrimas“, skyrius skirtas ištirti realizuotą steganografinį metodą, atsižvelgiant į metodo talpą, neaptinkamumą ir tvirtumą;
4. „Išvados“, skyrius skirtas apibendrinti, viso darbo rezultatą.

1. STEGANOGRAFIJA

1.1. Analizės tikslas

Steganografijos temos analize siekiama:

- Sužinoti kuo tinklo steganografija, skiriasi nuo kitų steganografijos rūšių;
- Susipažinti su steganografiniais metodais visuose TCP/IP lygiuose, taip įgaunant supratimą kaip steganografija įgyvendinama skirtinguose tinklo protokoluose;

1.2. Steganografija

Žodis steganografija, kilęs iš graikų kalbos, reiškia uždengti, užslėpti ar apsaugoti rašmenys. Šnekant apie steganografijos pritaikymą plačiąją prasme, tai informacijos rašymas ir apsikeitimas apie kurią žino tiktais siuntėjas ir gavėjas.

Steganografija, skaitmeninėje aplinkoje, slaptai informacijai persiųsti gali naudoti įvairius skirtingus nešlius į kuriuos ir yra įterpiama slapta informacija. Staganografiniai nešliai yra skirstomi į penkias pagrindines rūšis [1]:

1. Paveikslėlių steganografija – šioje steganografijos rūšyje informacijos užslėpimo objektu yra panaudojamas paveikslėlis. Metodai užslėpti informaciją yra skirstomi į dvi grupes: paveikslėlio srities ir transformacijos srities. Paveikslėlio srities būdų slapta informacija yra tiesiogiai įterpiama į pikselius, o transformacijos srities būdų, paveikslėlis iš pradžių yra transformuojamas ir tik tada slapta informacija įterpiama į paveikslėlį. Ši steganografijos rūšys yra labai populiari, nes paveikslėlių duomenų tipas yra lengvai persiunčiamas, o patys paveikslėliai turi daug perteklinių bitų, kurie gali būti panaudoti slaptai informacijai užslėpti.
2. Tinklo steganografija – šioje steganografijos rūšyje, slaptai informacijai užslėpti ir pernešti, yra naudojami įvairūs OSI modelio tinklo protokolai. Vienas iš būdų įvykdyti tinklo steganografija yra į, kurio nors, protokolo nereikšmingus ar nenaudojamus laukus įrašyti slaptą informaciją ir ją nepastebimai perduoti gavėjui nesutrikdžius tinklo darbo. Informacijai užslėpti gali būti panaudojami vienas arba keli protokolai, pagal naudojamų protokolų kiekį tinklo steganografija skirstoma į dvi grupes: „Intra-protocol“ tinklo steganografija – naudojamas vienas protokolas steganografijai įvykdyti ir „Inter-protocol“ tinklo steganografija – steganografijai naudojami keli protokolai kartu. Šio tipo steganografijos šaka yra dar nepilnai išvystyta ir ištyrinėta, tačiau yra sparčiai besivystanti.
3. Teksto steganografija – šioje steganografijos rūšyje, informacijos slėpimui, yra naudojamas tekstas. Steganografijos metodai skirti tekstui, informaciją slepia kurioje nors kiekvieno žodžio raidėje. Tekstą naudoti steganografijai yra sudėtinga, nes raidės turi mažą talpą, tai yra, informacijos kiekis kurią galima užslėpti yra mažas.
4. Garso steganografija – šioje steganografijos rūšyje, slaptai informacijai užkoduoti ir atkoduoti, yra naudojamas garsas. Steganografijai gali būti naudojami įvairūs plačiai paplitę garso formatai kaip, kad MP3, MPEG, MIDI, WAVE ir t.t.
5. Vaizdo įrašų steganografija - šioje steganografijos rūšyje, steganografijai įvykdyti, yra naudojami vaizdo įrašai. Ši rūšys gera tuo, kad gali užslėpti ypatingai daug informacijos, nes vaizdo įrašai, tai paveikslėlių einančių vienas po kito junginys, o kaip jau minėta, paveikslėliai turi daug pertekliškumo. Steganografijai galima panaudoti įvairius vaizdo įrašų formatus kaip, kad H.264, MP4 ir kitus.

1.3. TCP/IP ir OSI modeliai

Informacijai apsikeisti tinklu, juo keliauja informaciją sudarantys paketai. Tačiau, kad šis paketų siuntimas ir gavimas tinklu veiktų, turėjo būti sudarytos tam tikros taisyklės ir standartai – protokolai. Tinklų komunikacijoms buvo sukurta daugybė protokolų, o jų rinkinys skirtinguose lygmenyse vadinamas modeliu. Plačiai naudojami yra du modeliai – „Transmission Control Protocol/Internet Procotol“, toliau TCP/IP ir „Open System Interconnection“, toliau OSI.

Abu TCP/IP ir OSI modeliai yra sluoksniuoti, kiekvienas sluoksnis duomenų siuntimo metu atlieka jiems paskirtas funkcijas, tam, kad paketai būtų sėkmingai išsiųsti ir gauti. Modeliai kartu pavaizduoti 1.1 paveiksle.

	TCP/IP	OSI	
4	Taikymo (FTP, HTTP...)	Taikymo	7
		Atvaizdavimo	6
		Sesijos	5
3	Transporto (TCP, UDP)	Transporto	4
2	Tinklo (IP)	Tinklo	3
1	Tinklo sąsajos	Ryšio	2
		Fizinis	1

1.1 pav. TCP/IP ir OSI modeliai

Persiuntimo metu kiekvienas sluoksnis sukuria savo antraštę, kuri atpažįsta paketą ir jį pasiunčia tolimesniu sluoksniu žemyn, kurie taip pat atlieka šiuos veiksmus, tai vadinama enkapsuliacija. Kai informacija gaunama vyksta atvirkštinė operacija, paketai keliauja iš žemiausio sluoksnio aukštyn panaudojant antraštėse esančią informaciją, tai vadinama dekapsuliacija.

Sluoksnių atliekamos funkcijos [2]:

- 7 sluoksnis – taikymo, šis sluoksnis suteikia taikomosioms programoms komunikavimo tinklu galimybes, šiame sluoksnyje veikia šie protokolai – DNS, FTP, HTTP, IMAP, POP, SMTP.
- 6 sluoksnis – atvaizdavimo, šis sluoksnis atsakingas už spausdintuvų, failų sistemų prijungimo prie tinko, užtikrinimo, kad resursai yra suderinami, taip pat suteikia duomenų suspaudimo ir šifravimo funkcijas.
- 5 sluoksnis – sesijos, šis sluoksnis sujungia komunikuojančius įrenginius, prižiūri sujungimą ir kai reikia atjungia, turi autentifikavimo ir autorizavimo funkcionalumą. Šiam sluoksniui priklauso NFS, RPC protokolai.
- 4 sluoksnis – transporto, šis sluoksnis užtikriną visišką duomenų perdavimą teisingiems procesams, taip pat turi vientisumo užtikrinimo, klaidų taisymo galimybes. Protokolų pavyzdžiai – TCP, UDP.
- 3 sluoksnis – tinklo, atsakingas už paketų maršrutizavimą loginiais adresais bei klaidų kontrolę. Šiam sluoksniui priklauso IP protokolas.
- 2 sluoksnis – ryšio, supakuoja ir išpakuoja siunčiamus duomenis į kadrus, nustato fizinius adresus, pagrindinė šio sluoksnio funkcija yra išlaikyti duomenis vientisus fiziniame lygmenyje, dažnai veikia kaip buferis duomenims tarp tinklo ir fizinio sluoksnio perduoti. Šiam sluoksniui priklauso eterneto protokolas.
- 1 sluoksnis – fizinis, šiame sluoksnyje vyksta duomenų perdavimas per tinklo komunikacinius tinklus, kaip galima spręsti iš pavadinimo, šis sluoksnis apima visą reikiamą aparatūrą tinklo komunikacijoms atlikti. Nuo šio sluoksnio priklauso

duomenų perdavimo greitis, duomenų konversijos funkcijos, kurios sudaro paketų srautus iš vieno įrenginio į kitą.

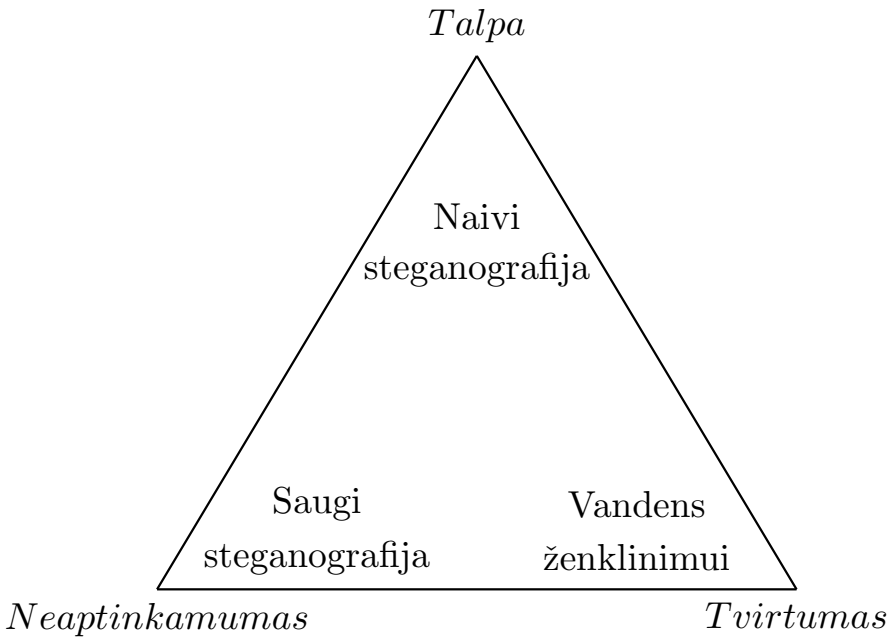
OSI modelis, yra idealizuojama tinklo koncepcija, nors ir yra neprivaloma, protokolai ir sistemos dažnai laikosi šio modelio apibrėžtų specifikacijų. Modelis yra naudingas, nes suskirstant tinklą į sluoksnius, lengviau suprasti kaip tinklas ir jame esančios funkcijos veikia [2].

TCP/IP protokolas turi keturis sluoksnius, iš kurių vienas sluoksnis gali atitikti kelis OSI modelio sluoksnius, kaip, kad matoma 1.1 paveiksle.

1.4. TCP/IP steganografija

1.4.1. Tinklo steganografijos metodų savybės

Dažniausiai minimos trys pagrindinės steganografijos metodus apibūdinančios savybės – talpa (tinklo steganografijoje – pralaidumas), neaptinkamumas ir tvirtumas (kai kurie įvardija ir ketvirtą savybę – kainą [3]). Šios savybės gali būti nubrėžtos trikampiu, mat visas tris pasiekti yra gana sunku, nes yra viena nuo kitos priklausančios, trikampis matomas 1.2 paveiksle [4].



1.2 pav. Steganografijos savybių trikampis

Pralaidumas – tai teoriškai numanomas ar praktiškai pasiekiamas slaptų žinučių ilgis, matuojamas įvairiai – bitais per laiko tarpą, bitais per paketą, bitais per kadrą.

Neaptinkamumas – tai steganografinio metodo atsparumas nuo aptikimo – steganalizės (plačiau 1.4.3 skyriuje).

Tvirtumas – tai slaptos žinutės vientisumo išsilaikymas visos slaptos komunikacijos metu.

Tinklo steganografijos pagrindinis tikslas, kaip minėta 1.1 skyriuje – informacijos perdavimas slaptai, nepastebint kitoms šalims, taigi kuriant steganografinį metodą pirmenybė turėtų būti skiriama nepastebimumui, tada tvirtumui ir galiausiai pralaidumui. Svarbu žinoti, kad kuo didesnė talpa, tuo slapti kanalai lengviau pastebimi, kaip ir matoma 1.2 paveiksle – didelės talpos steganografija yra naivu.

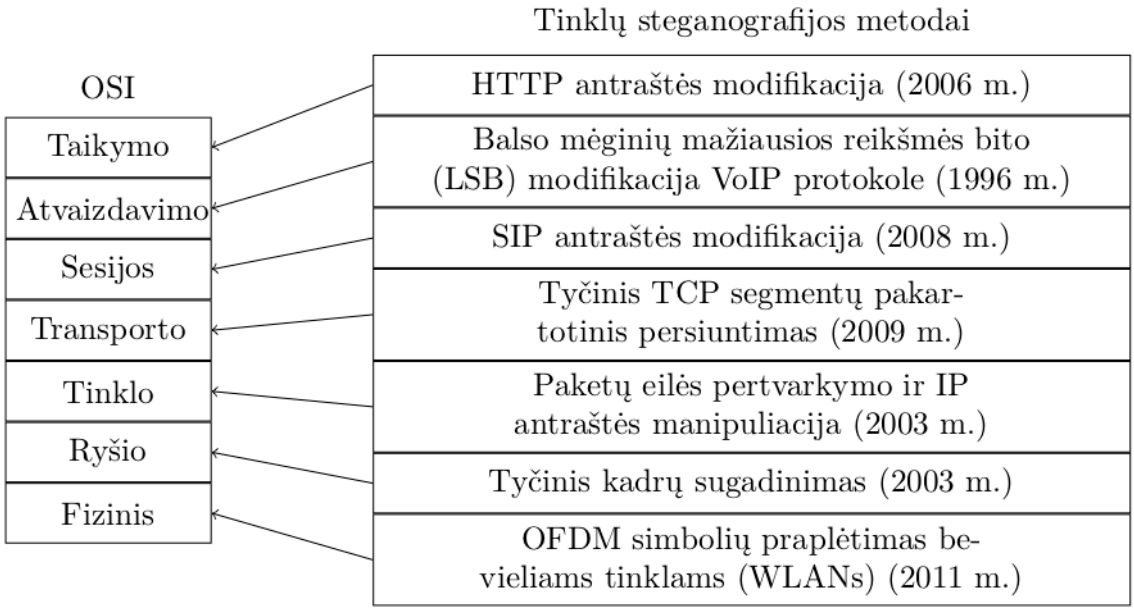
Pagal tai, kaip visos trys savybės atliktos, galima vertinti, steganografinio metodo efektyvumą [5].

Kriptografijos taikymas kartu su steganografija padidina slaptų kanalų egzistavimo aptikimo tikimybę [5], tačiau taikymas rekomenduotinas – slaptų duomenų aptikimo atveju, jie bus užšifruoti.

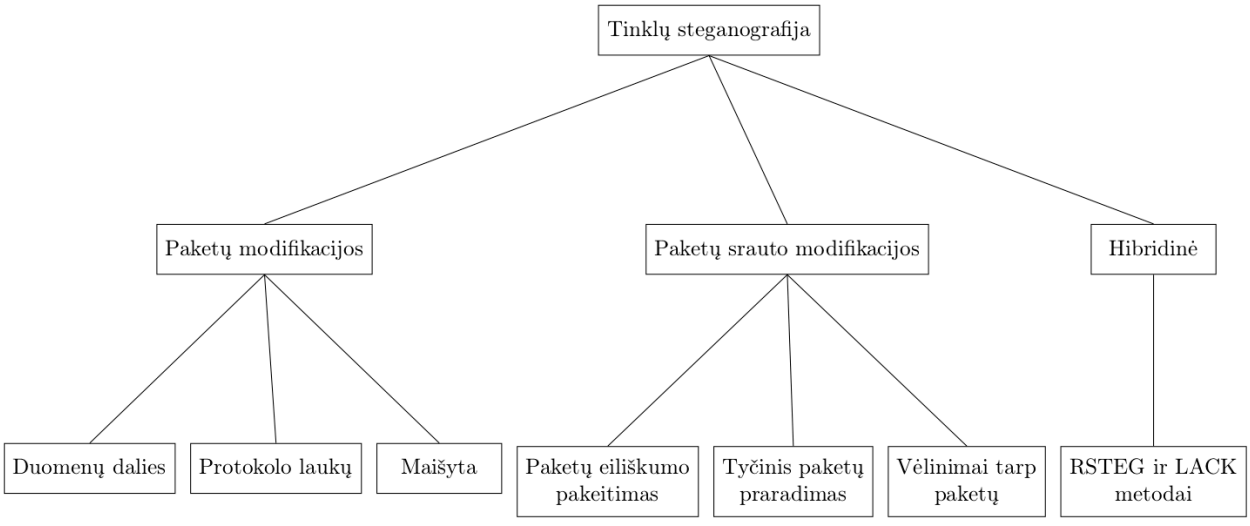
1.4.2. Tinklo steganografijos metodų skirstymas

Tinklo steganografijos metodai gali būti skirstomi dvejais būdais [5]:

1. pagal protokolo atliekamą funkciją OSI modelio sluoksnyje (1.3 paveikslas)
2. pagal atliekamos modifikacijos protokolui tipą, yra trys (1.4 paveikslas):
    a. paketų modifikacijas:
        i. tinklo protokolų – IP, TCP, UDP antraščių modifikavimas;
        ii. persiunčiamo paketo duomenų srities modifikavimas;
        iii. maišytų technikų, vykdomos ir paketų antraščių, ir duomenų dalių modifikacijos;
    b. paketų srauto struktūros modifikacijas:
        i. paketų eiliškumą sutrikdantys metodai;
        ii. paketus tyčia uždelsiantys metodai;
        iii. paketų eiliškumo numerių praleidimą sukeliantys metodai;
    c. Hibridinis:
        i. metodai pritaiko ir paketo, ir paketų srauto modifikavimą kartu.



1.3 pav. Steganografijos metodų klasifikacija pagal protokolų atliekamas funkcijas OSI modelyje



1.4 pav. Steganografijos skirstymas pagal metodo atliekamas modifikacijas

Skirtingi modifikacijos būdai turi tam tikrų pliusų ir minusų [5]:

- Modifikuojant protokolo antraštes, galimos persiųsti slaptos informacijos kiekis yra gan didelis, įgyvendinimas nesudėtingas, tačiau yra galimybė sutrikdyti protokolų veikimą, kartu ir tinklo.
- Modifikuojant paketo duomenų sritį, persiunčiamų slaptų duomenų kiekis mažesnis, steganografija sunkiau įgyvendinama ir aptinkama, gali sutrikdyti duomenų persiuntimo tinkle kokybę.
- Paketų srauto modifikavimo metodais, slaptų duomenų persiuntimo kiekis nėra didelis, bei gali sukelti uždelsimus, kurie sumažintų perdavimo kokybę, tačiau šio tipo steganografija gan lengvai įgyvendinama ir sunkiau aptinkama nei modifikuojant protokolo antraštes.
- Patys naujausi ir geriausi metodai yra maišyti, nes iš esmės paveldi geriausias savybes – didesnė perduodamos informacijos talpa, sunkiai aptinkama, gan lengvai įgyvendinama, deja, bet kaip ir daugelis kitų metodų gali suprastinti ryšio kokybę.

1.4.3. Steganalizė tinkle – gynimosi mechanizmai

Tinklo steganografija suteikia galimybes tinklu persiųsti slaptus duomenis, to galimai, net nenutuokiant tinklo vartotojams ir administratoriams, žinoma, kad toks informacijos perdavimas tinklu matomas kaip saugumo grėsmė, siekiant užkirsti tokį slaptą duomenų persiuntimą tinkle, yra kuriami metodai aptikti slaptus kanalus. Metodai siekiantys aptikti steganografiją, vadinami – steganalize, ji paprastai yra atliekama dviem būdais – pasyviais stebėtojais arba aktyviais stebėtojais [6]:

- Pasyvūs stebėtojai, tik stebi tinkle vykstančius paketų apsikeitimus ir gali aptikti anomalijas, sukeltas steganografijos metodų. Persiųstos slaptos informacijos jie nepanaikina.
- Aktyvūs stebėtojai gali modifikuoti tinkle persiunčiamą srautą, jei aptinkami neįprastumai kuriuos steganografija paprastai sukelia. Pavyzdžiui, jei pastebi retai naudojamų antraščių laukų naudojimą – šiuos laukus užrašo nuliais, taip slapta informacija galutinio taško lieka nepasiekusi. Aktyvus stebėtojas turi pastoviai įrašinėti tinklo būseną, todėl gali reikėti papildomo įrenginio dideliems informacijos kiekiams įrašinėti, taip pat, šio tipo stebėtojai, gali sukelti tinklo vėlinimus.

1.4.4. Tinklo steganografijos slaptumo gerinimo metodai

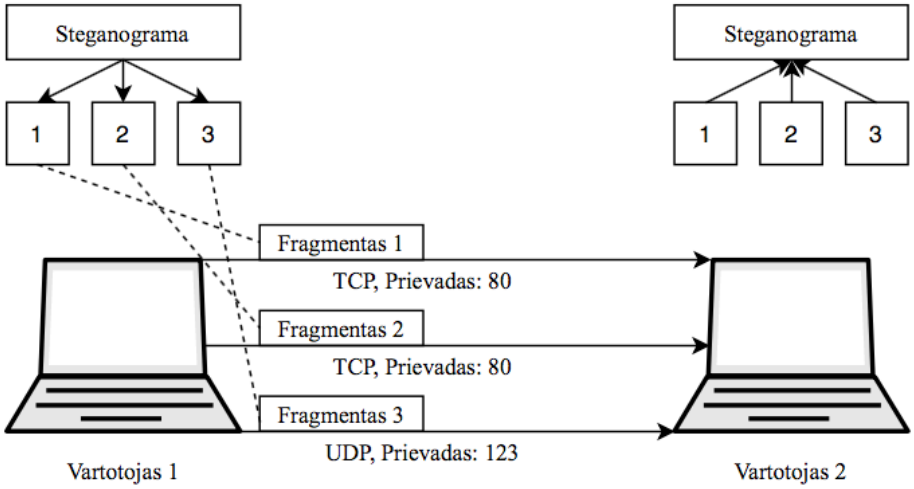
Kaip aptarta 1.4.3 skyriuje, steganalizės metodai gali efektyviai aptikti ar užkirsti kelią slaptų kanalų sudarymui, tačiau yra pasiūlytos kelios „Deep Hiding“ technikos, kurios gali labai pagerinti slaptų kanalų neaptinkamumą, tiesa, dažnai pralaidumo kaina [7].

Siūlomos penkios technikos [7]:

- steganaogramų išbarstymas – steganograma padalijama į dalis ir siunčiama laiko tarpais;
- steganogramų šokinėjimas – naudojami vis kiti steganogramų metodai;
- nešlio modifikacijų slaptinimas – bandoma paslėpti faktą, kad vyksta protokolų modifikacijos;
- „Inter-protocol“ steganografija – naudojami keli TCP/IP protokolai kartu;
- kelių sluoksnių steganografija – aukšto lygio steganografinis metodas panaudojamas sudaryti žemo lygio, kuris visiškai priklausomas nuo aukštesniojo.

Išbarstyti steganogramas yra trys būdai: srautu, įrenginiais ir mišriai.

Srauto atveju galima sudaryti kelis srautus, net skirtingais protokolais, pavyzdžiui, TCP ir UDP, ar skirtingais prievadais, ir kiekvienu jų išsiųsti po gabalėlį išskaidytos slaptos žinutės, pavyzdys matomas 1.5 paveiksle. Kad gavėjas galėtų atgal surinkti ir perskaityti žinutę reikia arba sužymėti žinutės indeksus, arba, geresnis būdas, padaryti slaptos žinutės išskaidytas dalis priklausomas nuo laiko, kada buvo išsiųsta, kur kiekviena dalis siunčiama vis vėliau.



1.5 pav. Steganogramos išbarstymas srautu

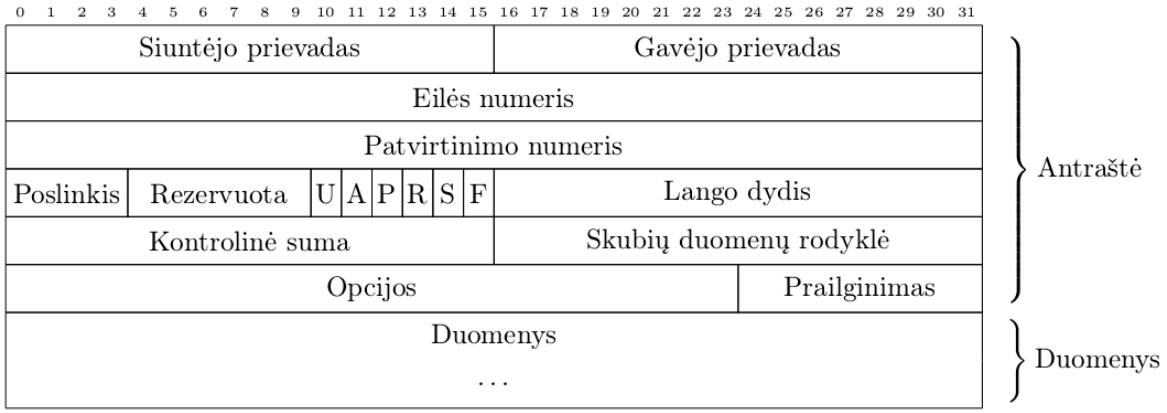
Nešlio modifikacijų slaptinimui yra įvairiausių būdų, kad ir paprasčiausias siunčiamų slaptų žinučių kiekio mažinimas, neblogas būdas yra stebėti įprastą tinklo darbą, kuriame bus atliekama slapta komunikacija ir prie šio srauto derintis.

„Inter-protocol“ atveju slaptos informacijos nešliai gali būti keli suderinami protokolai, norint aptikti, ar vyksta slapta komunikacija, reikia žinoti, kuriuos protokolus stebėti, tai padaro šią techniką naudojančius metodus sunkiau aptinkamus.

Norint pasunkinti steganogramos atkūrimą tinklo stebėtojams, reikėtų naudoti srauto barstymą, steganogramų šokinėjimą arba kelių sluoksnių steganografiją.

1.4.5. Steganografija transporto sluoksnyje

TCP ir UDP – protokolai, kurie veikia transporto lygmenyje. Kol TCP sujungimai užtikrina patikimą sujungimą su vientisų duomenų gavimo patvirtinimu ir duomenų korekcija, UDP siūlo sujungimą be jokių patvirtinimų, dėl to neaišku, ar duomenys buvo išsiųsti ir nesugadinti, bet kartu duoda didesnę sujungimo spartą.

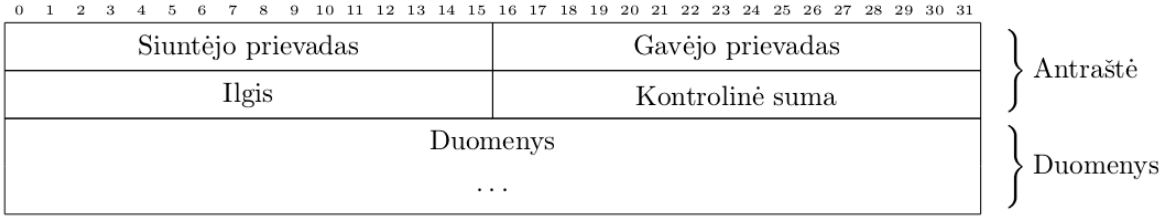


1.6 pav. TCP protokolo antraštė

Šaltinyje [8] aprašomas metodas sukuria slaptą kanalą panaudodamas 32 bitų „Eilės numeris“ ir „Patvirtinimo numeris“ TCP antraštės (1.6 paveikslas) laukus. „NUSHU“ [9] ir „Lathra“ [10] programos yra kiti steganografijos metodai, kurie naudoja šiuos laukus, pirmasis naudoja tuos pačius TCP laukus, taigi veikia panašiai, antrasis naudoja, tik ISN lauką – generuoja šio lauko numerį „Linux“ ir „OpenBSD“ sistemoms, kuris yra beveik neatskiriamas nuo TCP steko, esančio šiose sistemose, sugeneruojamų numerių, nebent, žinoma paslaptis [11].

Iš 64 URG, ACK, PSH, RST, SYN, FIN kombinacijų yra 29, kurios leidžia sukurti slaptą kanalą. Pavyzdžiui, jei URG nėra nustatytas, galima sukurti slaptą kanalą, kuriuo galėtų išsiųsti 16 bitų per paketą, taip pat įmanoma panaudoti „Rezervuota“ lauką [11].

Naudojant TCP, taip pat yra sukurtas vienas iš hibridinių steganografijos metodų – RSTEG. Šis metodas naudoja visus TCP pakartotinio persiuntimo mechanizmus. Metodas priverčia teisingai gautą TCP segmentą būti nepriimtu kaip teisingu ir taip priverstinai sukelti pakartotinį persiuntimą, o šis persiuntimas atsiunčia slaptą informaciją vietoj tikrojo iš pradžių siųsto paketo turinio. Šio metodo trūkumas yra tai, kad pakartotinių persiuntimų kiekį reikia stengtis išlaikyti prie įprasto TCP protokolui, taip apsunkinant aptikimą [11]. Slaptos žinutės gali būti žymimos santrumpų funkcija. Šis metodas plačiau nagrinėjamas 1.5 skyriuje.

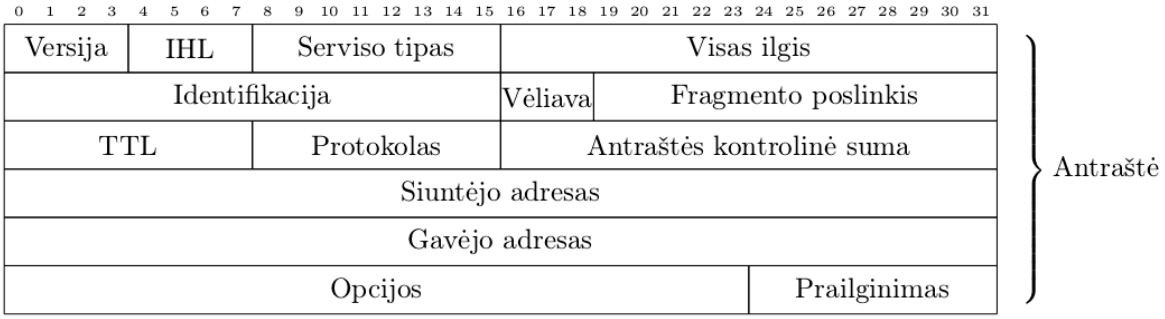


1.7 pav. UDP protokolo antraštė

UDP steganografijoje nėra labai naudojamas, viena priežastis – protokolo nepatikimumas, dėl to nebūtų aišku ar steganograma pasiekė tikslą, ar ne, kita – lengvas slaptų žinučių siuntimo aptinkamumas. Pagrindinis panaudojamas laukas steganografijai yra „Kontrolinė suma“, esantis UDP antraštėje (1.7 paveikslas), kadangi šis laukas yra neprivalomosios rūšies, tai yra, jo nebūtina išnaudoti. Kartu su „Kontrolinė suma“, taip pat įmanoma panaudoti „Siuntėjo prievadas“ ir „Ilgis“ laukus, norint pasiekti iki 6 bitų per paketą slaptai informacijai perduoti. Kitas šaltinis teigia, kad „Ilgis“ laukas steganografijai jokiais būdais neturėtų būti naudojamas, tačiau neįvardija kodėl [11] [6].

1.4.6. Steganografija tinklo sluoksnyje

Tinklo sluoksnyje yra vienas svarbiausių TCP/IP steko protokolų – IP. Šis protokolas turi dvi versijas IPv4 ir IPv6. Iš transporto sluoksnio gautą informaciją IP protokolas enkapsuliuoja, nurodydamas siuntėjo ir gavėjo adresus. Šis protokolas yra tinkamas steganografijai ir turi didelį kiekį tam sukurtų metodų.



1.8 pav. IP protokolo antraštė

Vienas iš būdų sukurti slaptus kanalus panaudojus IP protokolą yra antraštės laukai, kurie yra dažnai nepanaudojami, šie laukai – „Identifikacija“, „Vėliava“, „Fragmento poslinkis“ ir „Opcijos“ (1.8 paveikslas). Steganografijos taikymas šiame protokole turi trukumą – slapti kanalai gali būti lengvai panaikinti, pavyzdžiui, perrašius mažai naudojamus laukus nuliais [11].

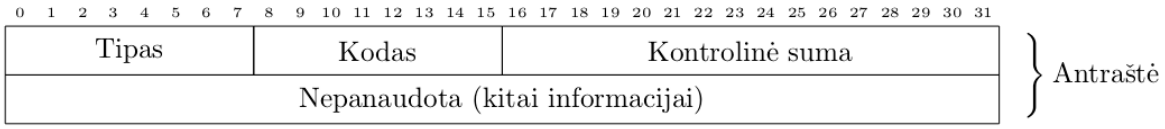
„Vėliava“ laukas yra sudarytas iš trijų bitų, kurių pirmas yra rezervuotas ir privalo būti nulis, antras laukas – DF, skirtas fragmentavimo nustatymui, trečias laukas – MF, kuris nustatomas jei siunčiamas paskutinis fragmentas [12]. Jei žinomas MTU galima panaudoti DF lauko bitą, slaptam 1 bito per paketą kanalui sudaryti.

„Identifikacija“ 16 bitų lauko reikšmė yra priskiriama siuntėjo ir tam tikrą laiko tarpą turi būti unikali, kad fragmentai būtų sudėlioti teisinga tvarka. [13] sukurtas metodas panaudoja 8 aukščiausius „Identifikacija“ lauko bitus sudaryti 8 bitų per paketą slaptą kanalą, kol žemiausi 8 bitai yra generuojami atsitiktinių skaičių generatoriumi.

IPv4 32 bitų „Laiko žyma“ lauke, kuris naudojamas, tik kai nustatytas IP protokolo „Opcijos“ lauke, yra galimybė įdėti slaptą žinutę – 1 baitą skiriant identifikatoriui kitus 3 slaptai žinutei. IP paketai turintys „Laiko žyma“ lauką maksimaliai gali nukeliauti 20 šuolių, todėl, realus praktinis šio metodo panaudojimas yra tik vietiniame tinkle. IP paketai labai retai naudoja šį lauką, todėl turėtų būti lengvai aptinkami ar ištrinami ugniasienių ar IDS sistemų, tačiau metodo autoriaus teigimu [14], metodas nebuvo aptiktas, ir slaptos žinutės sėkmingai keliavo tinklu.

Kiti [11] IP nagrinėti metodai lengvai aptinkami pasyvaus stebėtojo arba ištaisomi tinklo srauto normalizatorių, taigi nors IP protokolas turi daug galimybių sukurti slaptus kanalus, jie yra lengvai aptinkami.

ICMP yra dar vienas tinklo sluoksnyje esantis protokolas, jis atsakingas už informacijos ir įvykusių klaidų siuntimą. ICMP žinutės yra enkapsuliuojamos į IP protokolą. Yra 14 skirtingų ICMP žinučių tipų, kurios naudoja 4 baitus iš 8 galimų ICMP antraštėje (1.9 paveikslas) [11].



1.9 pav. ICMP protokolo antraštė

„Loki“ projektas [15] parodė, kad galima panaudoti „ICMP Echo Request“ ir „ICMP Echo Reply“ paketų duomenų dalis norint sukurti slaptą kanalą ICMP protokolu. Dėl paprasto aptikimo, šis slaptas kanalas veiks, tik jei tinklo įrenginiai nefiltruoja „ICMP Echo“ srauto.

Yra sukurtų įvairių programų ICMP steganografijai, pavyzdžiui, „ICMP-Chat“ – paprasta, komandinės eilutės tipo programa, kuri taip pat, siūlo slaptų žinučių šifravimo AES algoritmu funkcionalumą [16].

„V00d00n3t“ įrankis [17] gali sukurti slaptą kanalą IPv6 ir ICMPv6 versijoje – siuntėjas ir gavėjas susitaria 4 skaičių PIN ir tada gali keistis slaptomis žinutėmis.

ARP yra dar vienas tinklo sluoksnio protokolas, kuris naudojamas atrasti IP adresus priklausančius fiziniams įrenginiams. Yra pasiūlymas naudoti paskutinius aštuonis „Target protocol address“ lauko bitus slaptų žinučių siuntimui, tačiau nepaaiškinta kaip [18].

1.4.7. Steganografija taikymo sluoksnyje

Taikymo sluoksnis apima daugybę protokolų, iš kurių, labiausiai naudojami yra HTTP ir DNS, šie ir kiti taikymo sluoksnio protokolai gali būti naudojami slaptų kanalų sudarymui.

Taikymo sluoksnio protokolai susilaukia vis daugiau susidomėjimo steganografijos srityje, dėl to, kad dažnai numatyti prievadų nustatymai praleidžia kai kurių protokolų srautą (DNS – 53 prievadas, HTTP – 80 prievadas, HTTPS – 443 prievadas), tai sudaro galimybę šiais protokolais sudaryti slaptus kanalus, kurie galėti keliauti srautu neliečiami ugniasienės [19] [20].

DNS – srities vardų išrišimo sistema neatsiejama tinklų ir interneto dalis, šis servisas tinkle būtinai turi veikti, jei norima, kad iš domenų būtų išrišami IP adresai. Steganografijai šis protokolas yra naudojamas sukurti slaptus kanalus, kuriuose patalpinami kiti protokolai. Paprastai į DNS talpinami IPv4 [21] [22], TCP ir UDP protokolai. „NS“, „CNAME“ ir „TXT record“ – laukai į kuriuos galima patalpinti iki 255 baitų informacijos [11].

1.4.8. Steganografija fiziniame sluoksnyje

Steganografija įmanoma ir žemiausiame TCP/IP sluoksnyje, tačiau norint realizuoti tokius steganografinius metodus reikia realaus priėjimo prie įrangos, internetu – nutolusiai neįvykdoma, todėl aukštesnio lygio tinklo protokolai labiau domina kaip potencialūs slaptų duomenų nešliai. Vienas fizinio sluoksnio steganografinis metodas plačiau analizuojamas 1.5.1 skyriuje.

1.5. TCP/IP steganografinių metodų realizavimas

1.5.1. HICCUPS – steganografijos realizavimas fiziniame sluoksnyje

HICCUPS – „HIdden Communication system for CorrUPted networkS“, tai žemiausiame TCP/IP sluoksnyje įvykdomas steganografijos metodas, kuris pasinaudoja tinklo įrangos sukeliamu triukšmu. Šiuo metu, metodas labiau tinkamas bevieliams tinklams nei, kad laidais sujungtiems, nes bevieliai tinklai kur kas labiau paveikiami duomenų iškraipymų.

HICCUPS išnaudoja galimybę stebėti visus keliaujančius kadrus vietiniame tinkle ar esant bevielės stotelės zonoje ir siunčia sugadintus kadrus su neteisingomis CRC reikšmėmis.

Įgyvendinti šį steganografijos metodą siūlomos trys sąlygos (pirma būtina) [20]:

1. Jau minėta sąlyga – yra galimybė stebėti perimetre keliaujančius kadrus;
2. Viešai žinomas metodas kaip sudaromas inicializavimo vektorius (IV) šifravimui;
3. Tinkle naudojamas CRC.

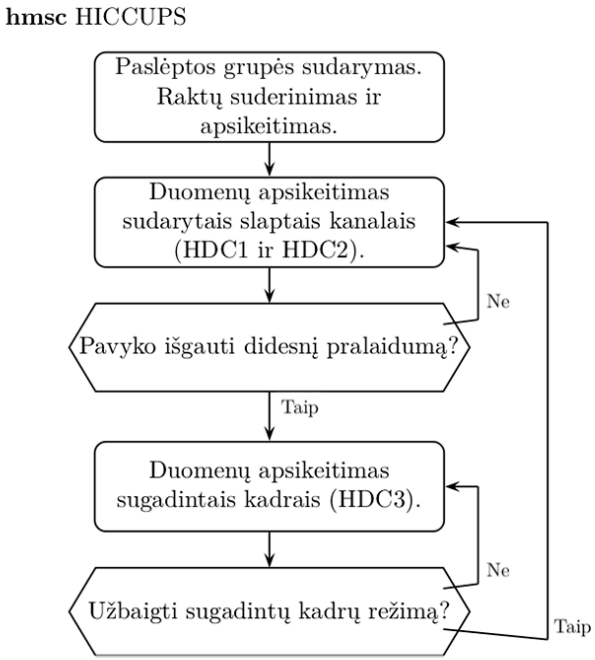
Galimi sudaryti trys slapti kanalai, priklausomai nuo minėtų sąlygų išpildymo [20]:

1. Inicializavimo vektoriuje;
2. MAC „source“ ir „destination“ antraštės laukuose;
3. CRC mechanizme.

Prieš kuriant slaptus kanalus yra paruošiamos slaptos stotys, kurios yra priskiriamos slaptai grupei, bei jų tinklo kortos sukonfigūruojamos į stebėjimo režimą, tam, kad stebėti visus kadrus (sugadintuose kadruose bus slaptos žinutės).

1.10 paveiksle matyti HICCUPS veikimo schema, veikimo eiga ir būdai:

1. Tarp slaptos grupės stočių apsikeičiama slaptais raktais;
2. Pradedamas vykdyti paprastasis režimas, kuris naudoja pirmąjį ir antrąjį slaptuosius kanalus (HDC1 ir HDC2), pralaidumas labai mažas – mažiau, nei procentas viso kadro;
3. Įvykdžius trečiąją sąlygą (naudojamas CRC), galima sukelti kadrų sugadinimą, kol įprastos stotys šiuos kadrus atmes, esančios slaptoje grupėje priims siunčiamus slaptus duomenis (HDC3).



1.10 pav. HICCUPS veikimo schema

Bevielės prieigos atveju metodas veikia panašiai, CRC paremtas slaptas kanalas gali būti išnaudotas esant WEP apsaugai.

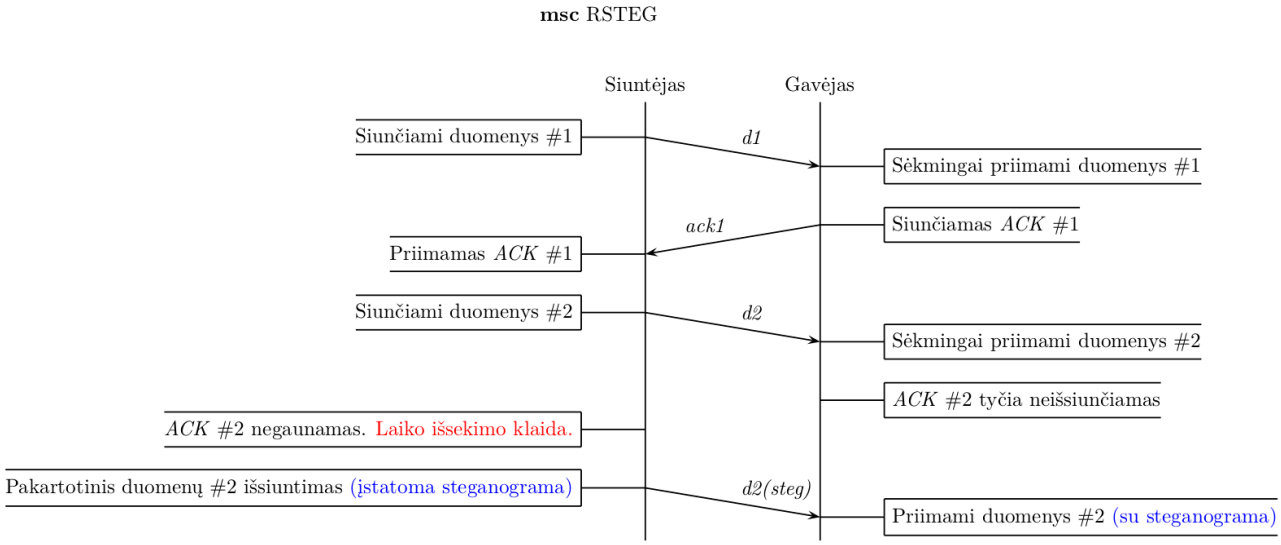
Šio steganografinio metodo įgyvendinimą apsunkina ne tik ankščiau minėtos priežastys, bet ir tai, kad bent jau, metodo autoriams nepavyko rasti tinklo kortos, kuri leistų įvesti savo suklastotus CRC [20], taigi, vienintelė galimybė išnaudoti trečiąjį, esminį slaptą kanalą – tinklo plokštę su CRC įvedimo funkcija pasigaminti patiems.

Teorinis, įgyvendinto slapto kanalo pralaidumas didelis – 802.11g stotelės tinkle 216 kbit/s.

1.5.2. RSTEG – hibridinio steganografinio metodo realizavimas

RSTEG – „Retransmission steganography and its detection“, tinklo steganografinis metodas galimas pritaikyti, net keliems TCP/IP šeimos protokolams skirtinguose sluoksniuose, sąlygą – protokolas turi palaikyti pakartotinio persiuntimo galimybę, metodo autoriai šio metodo įgyvendinimui pasirinko TCP protokolą. [23]

Metodas paremtas tuo, kad pakartotinį persiuntimą vykdantis protokolas negavęs ACK pranešimo, po nustatyto laiko paketą persiunčia iš naujo. Ši savybė buvo pamodifikuota taip, kad iš naujo persiunčiamuose paketuose įterpiamos slaptos žinutės, modifikacija matoma 1.11 paveiksle.



1.11 pav. RSTEG veikimo principas

Abu, siuntėjas ir gavėjas yra paruošiami slaptų duomenų persiuntimui:

- siuntėjas – paruošiamas būti valdytoju, kuris nustato kada turi būti inicijuojamas slaptų duomenų persiuntimas;
- gavėjas – sukonfigūruojamas taip, kad atpažintų kada reikia neatsakyti į siunčiamą žinutę.

Ši konfigūracija pasiekiama tokia santraukos funkcija:

$$IS = H(SK||Sequence\ Number||TCP\ Checksum||CB)$$

IS – identifikavimo eilė,
H – santraukos funkcija,
SK – iš anksto sutartas steganografijos raktas,
„Sequence Number“ – eilės numeris,
„TCP Checksum“ – TCP kontrolinė suma,
CB – kontrolės bitas.

Kontrolės bitas (CB) nustato ar bus siunčiamas atsakymas, ar ne. Kai gaunamas CB su nustatyta bito reikšme, kuri reiškia nesiųsti ACK, siuntėjas negavęs iš gavėjo pranešimo, pakartotinai persiunčia duomenis, tačiau vietoje pirminių buvusių duomenų, įterpia slaptą žinutę.

Kadangi, šis metodas naudoja realų tinklo srautą, tai yra, nemodifikuoja protokolo antraštės laukų įterpti slaptus duomenis, o naudoja lauką būtent ir skirtą duomenims persiųsti, būtina stebėti tinklo elgseną ir metodą sureguliuoti taip, kad tinklo srautas atrodytų įprastas tai aplinkai. Svarbiausia stebėti įprastą pakartotinių persiuntimų kiekį.

TCP turi tris pakartotinio persiuntimo mechanizmus: RTO, FR/R ir SACK.

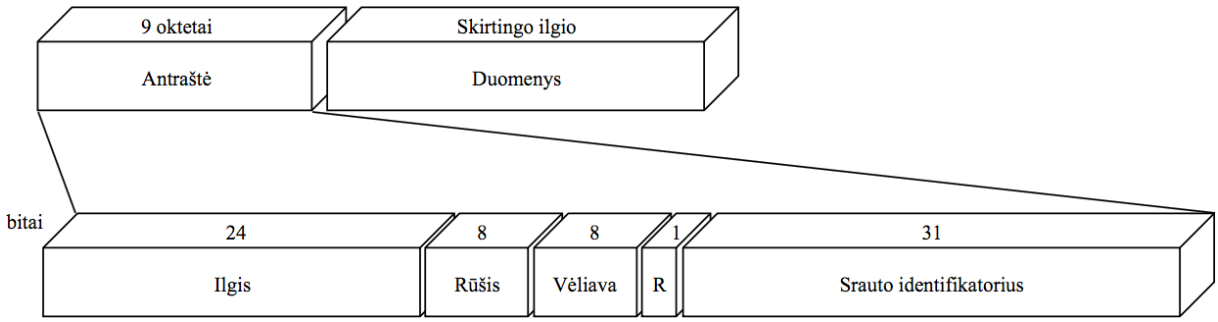
Priklausomai, nuo to, kam teikiama pirmenybė RSTEG kuriamais slaptais kanalais, reikėtų pasirinkti naudoti tinkamą mechanizmą – RTO jei siekiama didžiausio neaptinkamumo, SACK jei siekiama geriausio duomenų pralaidumo [23].

Slaptumo požiūriu, šis steganografinis metodas yra labai geras, stebint pakartotinių persiuntimų lygį tinkle ir atitinkamai metodui persikonfigūruojant yra labai sunku aptikti slaptų kanalų būvimą, nes pakartotiniai persiuntimai tinklo darbui yra įprasti.

1.6. HTTP/2 protokolas

2015 metais ilgą laiką nekeistas HTTP/1.1 pagaliau gavo atnaujinimą, su daug atnaujinimų, pavyzdžiui, galimybe išsiųsti kelias HTTP užklausas tame pačiame TCP sujungime keliais srautais – nebereikia kelių TCP sujungimų tarp serverio ir kliento [24].

Kiekvienas HTTP/2 kadras sudaromas iš antraštės ir duomenų, pati antraštė sudaryta iš 9 oktetų ir turi laukus – 24 bitus duomenų ilgiui, 8 bitus kadro tipui, 8 bitus vėliavėlėms, 1 bitą rezervuotą ir 31 bitą srauto identifikatoriuj. Grafiškai HTTP/2 kadras matomas 1.12 paveiksle.



1.12 pav. HTTP/2 kadras

HTTP/2 protokolas turi 10 kadrų, jie yra:

- *DATA*;
- *HEADERS*;
- *PRIORITY*;
- *RST STREAM*;
- *SETTINGS*;
- *PUSH PROMISE*;
- *PING*;
- *GOAWAY*;
- *WINDOWS UPDATE*;
- *CONTINUATION*.

Kiekvienas kadras prasideda 9 oktetų antrašte, toliau, duomenų dalyje, įstatomi kadrui unikalūs laukai.

HTTP/2 kūrimo eigoje buvo daug koncentruotasi saugos aspektu, taigi, steganografiniu požiūriu, šioje versijoje, tų pačių slaptų kanalų užmegzti, kuriuos galėjo HTTP/1.1 versijoje turėtų nepavykti, kadangi yra integruotos apsaugos, kurios panaikina anomalijas [19]. Tačiau, šio protokolo steganografinės galimybės dar nėra ištyrinėtos.

1.7. Analizės išvados

Išanalizavus steganografijos taikymą tinkluose ir jos pritaikymą tinklo protokolams paaiškėjo:

1. Steganografiją galima vykdyti visuose TCP/IP modelio protokoluose, su sąlyga, jei tas protokolas turi – neišnaudotus laukus, nebūtinus laukus, jei funkcija nebūtina, nėra griežtų nurodymų kokios vertės turi būti įrašomos;

2. Steganografijos tikslas yra duomenis užslėpti, taigi kuriamų steganografinių metodų prioritetas turėtų būti neaptinkamumas;
3. Pagal metodų tipus, sunkiausiai aptinkami yra mišrūs steganografiniai metodai, antraštes modifikuojantys metodai dažniausiai aptinkami lengviausiai;
4. Steganografinių metodų kuriamus slaptus kanalus aptikti arba sunaikinti yra naudojami steganalizės metodai;
5. Norint labiau pagerinti steganografijos metodų neaptinkamumą galima naudoti slaptinimo technikas, pavyzdžiui, paketų išbarstymo;
6. Tinklo sluoksnio steganografijos protokolai – IP ir ICMP yra labiausiai ištyrinėti, todėl lengvai aptinkami;
7. Taikymo sluoksnio steganografija labai populiarėja, verta dėmesio, nes turi protokolų, kurie dažnai praleidžiami pro ugniasienes;
8. Fizinio lygmens steganografija įvykdyti sunku, būtinas priėjimas prie tinklo įrangos;
9. Šiuo metu, RSTEG vienas geriausių steganografijos metodų, kol kas taikytas, tik TCP;
10. HTTP/2 naujas taikymo lygio protokolas, kurio steganografinės galimybės dar neišnagrinėtos, praktiškai realizuotų steganografinių metodų nėra.

1.8. Darbo tikslas, uždaviniai, planas ir siekiami privalumai

Įvertinus išvadas nuspręsta, kadangi, HTTP/2 dar neturi praktiškai realizuotų steganografinių metodų ir taikymo sluoksnio naudojimas steganografijai yra populiarėjantis, keliamas darbo tikslas – praktiškai realizuoti ir ištirti steganografinį metodą paremtą HTTP/2 protokolu.

1.9. Siekiamo sprendimo apibrėžimas

Numatomi uždaviniai metodo kūrimo eigoje:

- Panaudojami vienas ar keli HTTP/2 kadrai, slapto kanalo kūrimui;
- Sėkmingai siunčiami ir gaunami duomenys sudarytu slaptu kanalu;
- Metodui pritaikomos papildomas funkcijos – duomenų suspaudimas ir užšifravimas;
- Tyrimas pagal steganografinius metodus apsakančias savybes.

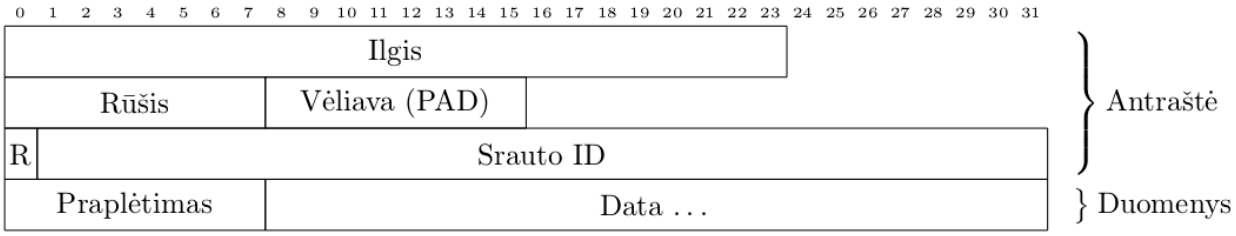
2. STEGANOGRAFINIS METODAS HTTP/2 PROTOKOLU

2.1. HTTP/2 kadrų panaudojimas slaptiems kanalams sudaryti

Atlikus analizę, paaiškėjo, kad HTTP/2 protokolas duomenims siųsti naudoja dešimt skirtingų kadrų. Kadangi, HTTP/2 sujungime loginė žinutė sudaryta iš vieno ar daugiau kadrų, manoma, kad pasinaudojus kuriuo nors vienu ar kelias kadrais ar jų antraščių laukais, galima sudaryti slaptą kanalą slaptiems duomenims persiųsti.

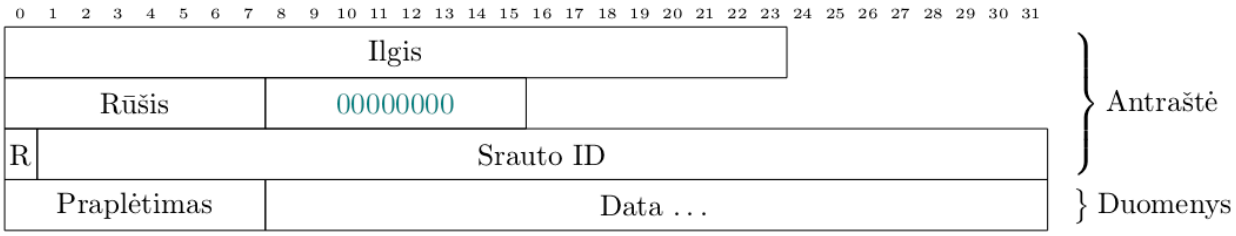
2.1.1. *DATA*, *HEADERS* ir *PUSH PROMISE* kadrai

HTTP/2 *DATA*, *HEADERS* ir *PUSH PROMISE* kadrai antraštės dalyje turi vėliavą – *PAD* (2.1 paveikslas), ši vėliava nustato ar bus prailginti duomenys. Jei „Vėliava“ laukas yra nustatomas, kadre yra naudojamas „Praplėtimas“ laukas, šis laukas yra vieno baito ilgio, į jį tinkamos įrašyti reikšmės yra nuo 0 iki 255. Jei šį lauką pavyktų išnaudoti, kuriuo nors iš trijų kadrų būtų galima sudaryti 1 baito per kadrą slaptą kanalą.

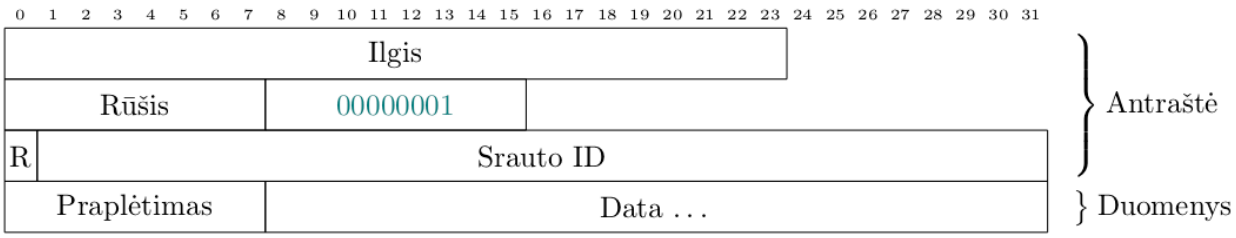


2.1 pav. *DATA* kadro antraštės ir duomenų dalis

Kitas variantas, galėtų būti – vėliavos būvimo interpretavimas kaip loginis nulis arba vienetas. Taip būtų sudarytas vieno bito per kadrą slaptas kanalas, jei *PAD* vėliava nustatyta serveris duomenis interpretuotu kaip dvejetainį vienetą, jei ne – kaip dvejetainį nulį. *DATA* kadras, kurio *PAD* vėliava nėra nustatyta, matomas 2.2 paveiksle, atvirkštinis kadro variantas matomas 2.3 paveiksle, įstatyti duomenys paryškinti spalva.



2.2 pav. *DATA* kadras su nenustatyta *PAD* vėliava



2.3 pav. *DATA* kadras su nustatyta *PAD* vėliava

RFC 7540 nustato, kad jei *DATA* kadras turi duomenų praplėtimą, „Praplėtimas“ lauke esantys duomenys turėtų išlikti, *HEADERS* ir *PUSH PROMISE* kadrų atveju lauko duomenis galima pašalinti. Taigi, slapto kanalo sudarymui panaudojus duomenų praplėtimą geriausiai tinka *DATA* kadras, tačiau jei praplėtimo lauko duomenys nėra trinami, taip pat, gali būti panaudoti ir vėlesni du kadrai.

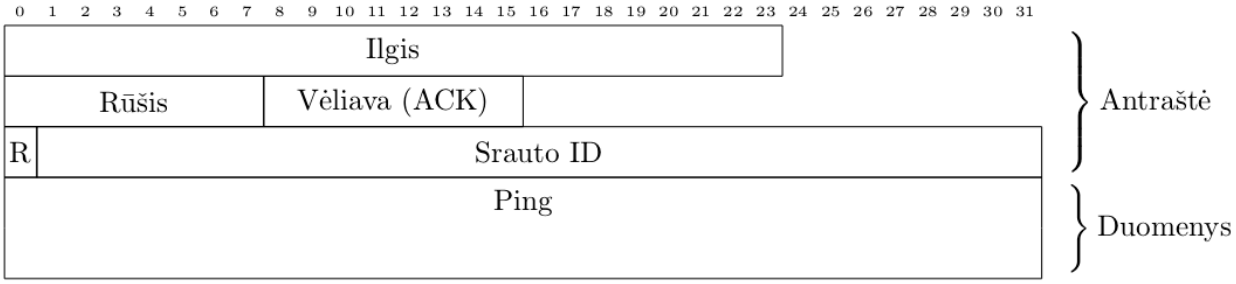
Pagal RFC 7540 [24], *PAD* atlieka saugumo funkciją – duomenų praplėtimu juos daro mažiau nuspėjamus, todėl šios vėliavos, net dažnas naudojimas HTTP/2 sraute neturėtų būti įtartinas.

2.1.2. *PING* kadras

PING kadras (2.4 paveikslas) naudojamas patikrinti:

- ar sujungimas tarp kliento ir serverio vis dar nenutrukęs;
- patikrinti vėlinimą tarp kliento ir serverio.

HTTP/2 protokole *PING* kadras asocijuojamas su srauto identifikatoriumi, kuris visada lygus nuliui. Slaptą kanalą, *PING* kadro atveju, galima sudaryti ne, tik į serverio pusę, bet ir į kliento, vienas *PING* kadras, pagal specifikaciją [24], privalo turėti įstatytus visus 64 bitus duomenų, kuriuos vartotojas gali išnaudoti kaip nori, įprastai, laukas užpildomas nuliais. Panaudojus „Ping“ lauką, teorinė steganografinė talpa yra iki 8 baitų per kadrą.



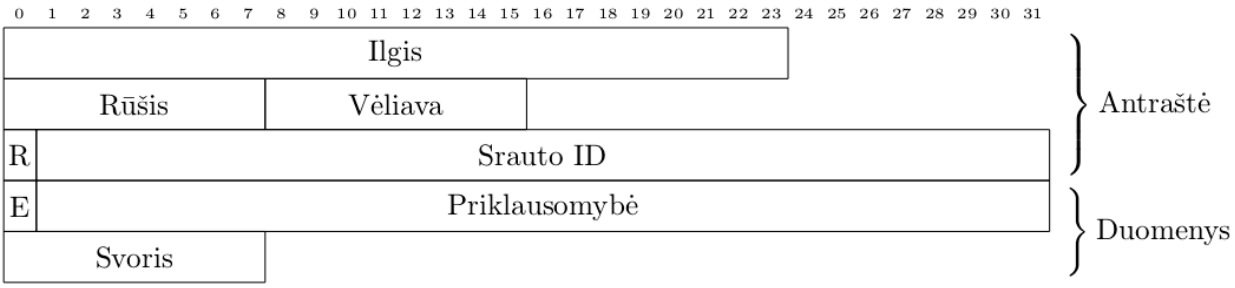
2.4 pav. *PING* kadro antraštės ir duomenų dalis

2.1.3. *PRIORITY* kadras

HTTP/2 protokole yra galimybė srautams suteikti eiliškumo prioritetus, tam naudojamas *PRIORITY* kadras. „Svoris“ lauke, matomas 2.5 paveiksle, galima nustatyti srauto svorį, kuris turi būti nuo 1 iki 256, didesnė svorio reikšmė reiškia didesnį srauto prioritetą, numatytasis srauto svoris – 15.

Pirmas *PRIORITY* kadras yra pradedamas *HEADERS* kadre nustačius prioriteto vėliavą.

Jei būtų panaudotas „Svoris“ laukas turėtų būti išgautas 1 baito per kadrą slaptas kanalas, tačiau, dėl to, kad laukui tinkama reikšmė prasideda ne nuo nulio, gavėjo pusėje gaunamos steganogramos reikšmės turėtų būti mažinamos per vienetą.



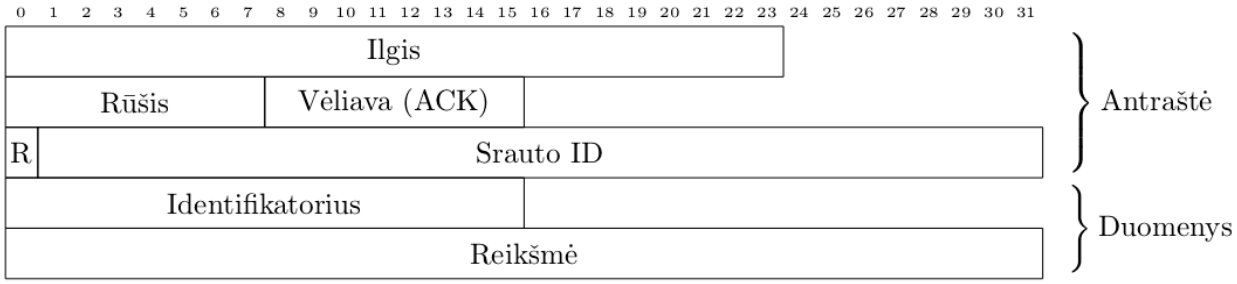
2.5 pav. *PRIORITY* kadro antraštės ir duomenų dalis

Slapto kanalo sudarymui, srauto priklausomybės lauko negalima panaudoti, nes jis būtinai turi nurodyti kito kadro nuo kurio yra priklausomas identifikatorių, jei jame įrašyta kas nors kita siunčiamas klaidos pranešimas.

Kadangi, *PRIORITY* kadras, yra nebūtino naudojimo, dėl to rečiau naudojamas, tai galėtų lemti didesnę aptikimo tikimybę, jei stebimo tinklo sraute HTTP/2 prioriteto kadrai neįprasti.

2.1.4. *SETTINGS* kadras

HTTP/2 sujungimo konfigūracijai nustatyti ir keisti naudojamas *SETTINGS* kadras. Šis kadras panaudojamas ne tik pirmo sujungimo metu, bet gali būti naudojamas, bet kurio, jau egzistuojančio sujungimo metu. *SETTINGS* kadro duomenų dalis sudaryta iš dviejų laukų – 16 bitų „Identifikatorius“ lauko ir 32 bitų „Reikšmė“ lauko, vizualiai kadro laukai matomi 2.6 paveiksle.



2.6 pav. *SETTINGS* kadro antraštės ir duomenų dalis

Į „Reikšmė“ lauką galimi įrašyti šeši parametrai, kurie yra nuskaitomi iš eilės.

Parametrai yra:

- „SETTINGS_HEADER_TABLE_SIZE (0x1)“;
- „SETTINGS_ENABLE_PUSH (0x2)“;
- „SETTINGS_MAX_CONCURRENT_STREAMS (0x3)“;
- „SETTINGS_INITIAL_WINDOW_SIZE (0x4)“;
- „SETTINGS_MAX_FRAME_SIZE (0x5)“;
- „SETTINGS_MAX_HEADER_LIST_SIZE (0x6)“.

„SETTINGS_HEADER_TABLE_SIZE“, gali įgauti bet kokią, iki 4 baitų reikšmę, tokia pati teoriška steganografinė talpa. Pirmas nustatomas dydis dešimtainėje sistemoje – 4096.

„SETTINGS_ENABLE_PUSH“, gali įgauti reikšmes – 0 arba 1, toks *SETTINGS* kadras galėtų sudaryti 1 bito per kadrą slaptą kanalą.

„SETTINGS_MAX_CONCURRENT_STREAMS“, gali būti bet kokia reikšmė iki 4 baitų, tačiau dėl to, kad reikšmės nustatymas į 0, nebeleidžia kurti naujų srautų, tai galėtų sutrikdyti slaptų duomenų siuntimą.

„SETTINGS_INITIAL_WINDOW_SIZE“, maksimalus galimas dydis yra 2^{31-1}, todėl slaptam kanalui sudaryti galėtų būti panaudojami 3 baitai per kadrą.

„SETTINGS_MAX_FRAME_SIZE“, minimali galima reikšmė – 2^{14}, maksimali – 2^{24-1}, todėl iš trijų galimų baitų galėtų pilnai išnaudoti, tik vieną, aukščiausią baitą, 1 baito per kadrą kanalui.

Visus kadrus, į kuriuos įtelpa bent baitas duomenų, sudėjus bendram kanalui, teoriškai turėtų būti sudarytas 12 baitų per paketą kanalas. Tačiau, kyla abejonių ar toks chaotiškas nustatymų keitimas sudarytų patikimą kanalą, tuo labiau pastovus nustatymų keitimas sraute gali atrodyti įtartinai.

2.2. HTTP/2 steganografinio metodo koncepcinis modelis

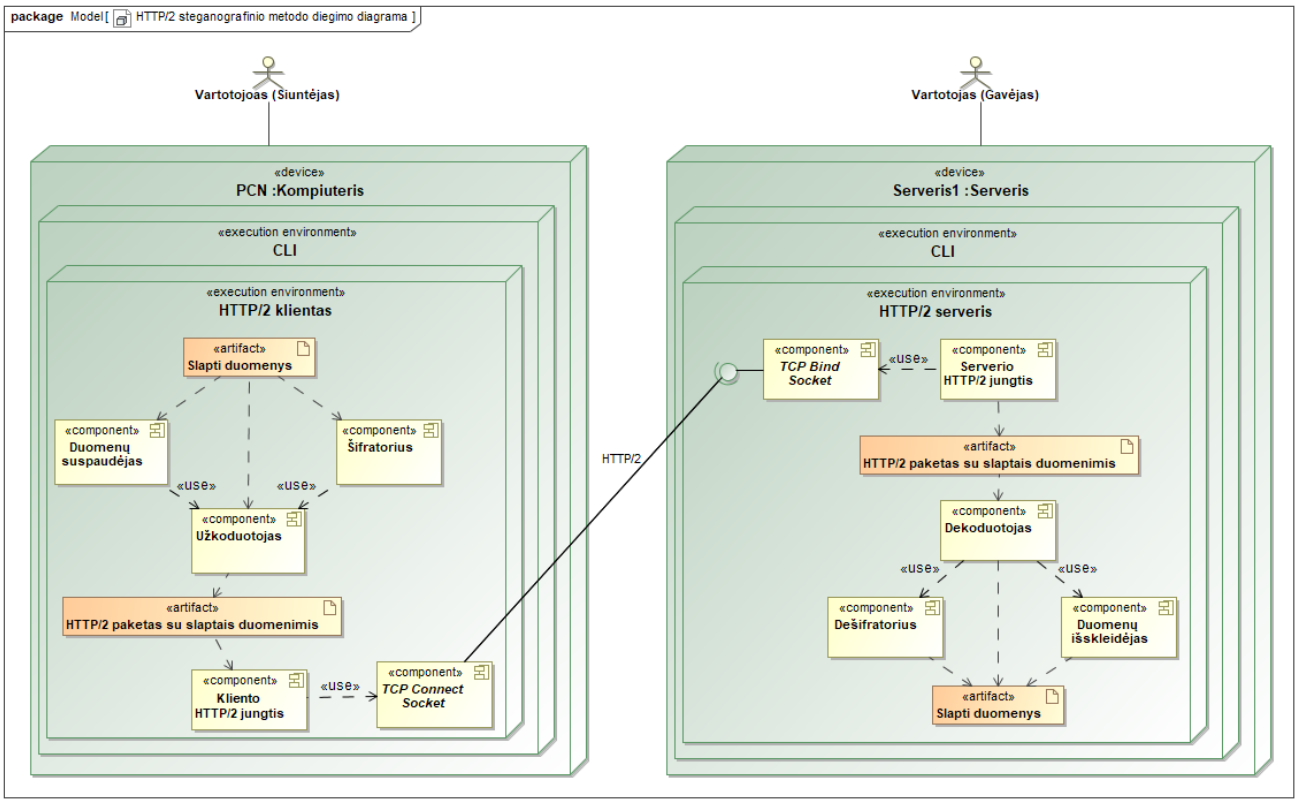
HTTP/2 protokolas yra transportuojamas panaudojus TCP protokolą, todėl pirmiausiai turi būti sukuriama TCP jungtis, šią jungtį sudaryti, kuriamas metodas pasiremia klientas–serveris modeliu. Tai reiškia, kad viena, steganografinį metodą naudojanti, prototipo pusė veiks kliento režimu (jungsis per TCP), kita – veiks serverio režimu (klausysis ateinančių TCP jungčių).

Sudarius TCP ir HTTP/2 jungčių metodus, reikia sukurti metodus:

- Kliento pusei:
  - Įvestos slaptos žinutės duomenų išskaidymas į baitus ar bitus (priklausomai nuo to kaip bus nuspręsta panaudoti HTTP/2 kadruose kuriamus slaptus kanalus) ir jų užkodavimas;
  - Papildomas funkcijos:
    - Slaptos žinutės duomenų suspaudimas;
    - Slaptos žinutės duomenų užšifravimas;
  - Užkoduotų slaptų duomenų (atviro teksto, suspausti, užšifruoti, suspausti ir tada užšifruoti) įdėjimas į 2.1 skyriuje apibrėžtus HTTP/2 kadrų laukus.
- Serverio pusei:
  - Slaptų duomenų ištraukimas iš kadrų laukų apibrėžtų 2.1 skyriuje;
  - Duomenų sudėjimas ir atkodavimas;

- Papildomų funkcijų naudojimo atveju:
  - Slaptų duomenų išskleidimas;
  - Slaptų duomenų atšifravimas.

2.7 paveiksle pateikiama HTTP/2 steganografinio metodo prototipo koncepcinė diegimo diagrama, čia komponentų dalyse regimi aprašyti metodai. *TCP Connect Socket*, tai TCP jungimosi metodas, *TCP Bind Connect,* tai TCP jungimų besiklausantis metodas. Ir serverio ir kliento režimu prototipas, paprastumo dėlei, bus leidžiamas per CLI (komandinę eilutę). Paleistas serveris turi būti vienas, klientų, norinčių siųsti slaptus duomenis, gali būti daug.



2.7 pav. HTTP/2 steganografinio metodo koncepcinė diegimo diagrama

2.3. Duomenų persiuntimo sudarant slaptą kanalą HTTP/2 protokolu veiklos proceso modelis

Šiame skyriuje, pagal 2.2 skyriuje pateiktą diegimo diagramą ir aprašytus metodus, detaliau modeliuojami visi prototipe atliekami procesai panaudojus BPMN notaciją.

2.3.1. Procesų apžvalga

Visą procesų veiklą sudaro trys baseinai:

1. Vartotojas – Rankiniu būdu įveda:
   a. konfigūracinius duomenis serveriui ir jį paleidžia;
   b. konifgūracinius ir norimus išsiųsti slaptus duomenis klientui ir jį paleidžia;
2. Klientas atsakingas už:
   a. tinklo jungčių sudarymą:
      i. TCP jungimosi;
      ii. HTTP/2 kliento.
   b. duomenų:
      i. kodavimą;
      ii. suspaudimą;
      iii. užšifravimą.
   c. slaptų duomenų:
      i. įdėjimą į HTTP/2 kadrus;
      ii. išsiuntimą HTTP/2 kadrais.

3. Serveris atsakingas už:
    a. tinklo jungčių sudarymą:
        i. TCP klausymosi;
        ii. HTTP/2 serverio.
    b. duomenų:
        i. atkodavimą;
        ii. išskleidimą;
        iii. iššifravimą.
    c. slaptų duomenų:
        i. išėmimą iš HTTP/2 kadrų;
        ii. atvaizdavimą.

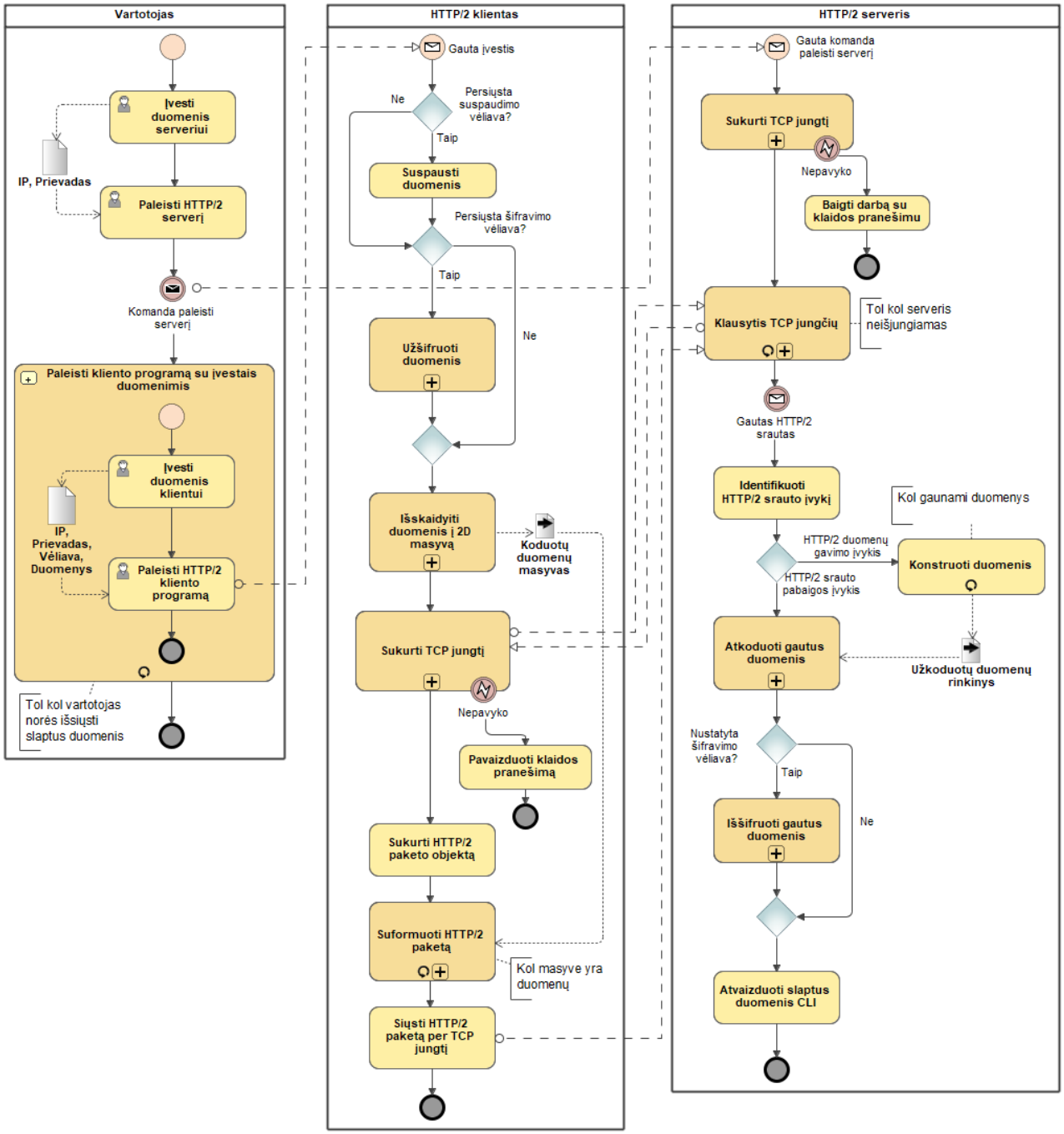
Procesų veikla su visais išvardintais baseinais matoma 2.8 paveiksle.

2.3.2. Vartotojo baseino veiklos procesai

Šiame skyriuje aprašomi vartotojo veiksmai norint sudaryti slaptą HTTP/2 kanalą persiųsti slaptus duomenis. Vartotojo atliekami procesai (2.8 paveikslas, "Vartotojas" baseinas):

1. Vartotojas į CLI suveda HTTP/2 steganografijai paruošto prototipo pavadinimą, konfigūracines vėliavas, norimus išsiųsti slaptus duomenis. Vėliavų pavyzdžiai:
    a. vėliava nustatyti prototipo veikimo režimą – klientas ar serveris (būtina);
    b. kokiu IP adresu siųsti duomenis, kurių adresų klausytis (numatytasis atvejis - 0.0.0.0, klausytis visų);
    c. kokiu prievadu siųsti duomenis ar kurio prievado klausytis (numatytas prievadas – 80);
    d. vėliava papildomoms funkcijoms (suspaudimui, šifravimui, kito tipo kodavimui).

    Kad išeitų siųsti slaptus duomenis, pirmiausiai prototipas turi būti paleistas serverio režimu.
2. Suvedus duomenis (su serverio vėliava) vartotojas paspaudžia klavišą paleisti prototipą, jei nebuvo klaidų paleidžiamas HTTP/2 steganografijos serveris;
3. Kai paleistas serveris, vartotojas gali pradėti siųsti slaptus duomenis pasinaudojęs tuo pačiu HTTP/2 steganografijai paruoštu prototipu, tik, šį kartą, su įvestais duomenimis, taip pat, įvedama kliento režimui paleisti reikalinga vėliava. Be įvestų slaptų duomenų, vartotojas kliento režimui, taip pat, gali įvesti tokias vėliavas:
    a. suspaudimas – slapti duomenys bus suspausti;
    b. kodavimas – dvejetainis ar šešioliktainis, be kodavimo, nuo to taip pat priklauso kiek paketų prireiks persiųsti slaptą pranešimą. Tai gali įtakoti slaptų duomenų persiuntimo slaptumą;
    c. šifravimas – prieš persiunčiant duomenis jie yra užšifruojami simetriniu raktu, tam pateikiama slapta frazė, tokia pačia slapta fraze turi būti paleistas ir serveris;
    d. vėlinimas – nustatomas laikas kas kiek laiko siųsti paketus, taip siekiant sumažinti įtarimą, kad vykstą slaptų duomenų siuntimas. Priklausomai kokio dydžio duomenys, tai gali labai prailginti jų persiuntimą.
4. Vartotojas pernaudoja prototipo kliento režimą slaptų duomenų siuntimui, kol pats nori ką nors išsiųsti.



2.8 pav. Duomenų persiuntimo sudarant slaptą kanalą HTTP/2 protokolu veiklos proceso modelis

2.3.3. Kliento baseino veiklos procesai

Kliento procese yra apdorojami vartotojo įvesti duomenys, kurie yra sudedami į HTTP/2 paketo kadrus ir išsiunčiami per sudarytą TCP jungtį.

Kliento atliekami procesai plačiau (2.8 paveikslas, “HTTP/2 klientas” baseinas):

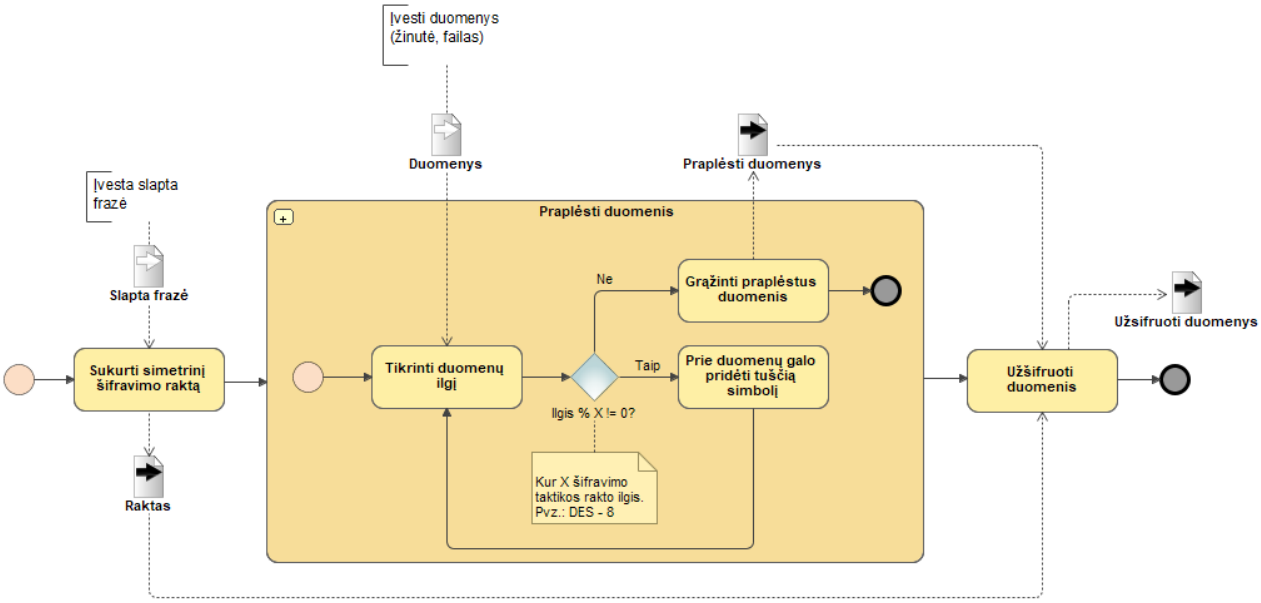
1. Klientas gavęs pranešimą iš vartotojo, pirmiausiai patikrina ar buvo įvesta suspaudimo vėliava, jei:
    a. taip – slapti duomenys yra suspaudžiami;
    b. ne – slapti duomenys nėra suspaudžiami.
2. Toliau, klientas tikrina ar buvo įvesta šifravimo vėliava su slapta fraze, jei:
    a. taip - keliaujama į duomenų šifravimo procesą, kur duomenys yra užšifruojami (2.9 paveikslas);
    b. ne – praleidžiamas šifravimas ir keliaujamą į duomenų skaidymo procesą.

3. Po papildomų funkcijų, seka duomenų skaidymo procesas, kur slapti duomenys išskaidomi į 2D masyvą (2.10 paveikslas);
4. Sudarius 2D duomenų masyvą, klientas bando sudaryti TCP jungtį su serveriu, šis procesas matomas 2.11 paveiksle;
5. Jei TCP jungtis sudaroma sėkmingai, klientas sukuria tuščią HTTP/2 paketą ir iš sudaryto slaptų duomenų 2D masyvo į jį sudeda duomenis, tai pavaizduota 2.12 paveiksle matomame procese;
6. Kiekvieną paruoštą HTTP/2 paketą (su slaptais duomenimis nustatytame kadre) klientas įdeda į TCP objektą ir juo persiunčia duomenis į serverį.

2.3.3.1. Slaptų duomenų šifravimas

Įvestus duomenis vartotojas gali pasirinkti šifruoti įvedus vėliavą ir slaptą frazę. Duomenų šifravimas atliekamas, taip, kaip matoma 2.9 paveiksle.

1. Iš slaptos frazės sudaromas slaptas raktas;
2. Tada, atliekamas tikrinimas ar duomenys atitinka šifravimo schemos palaikomą duomenų ilgį, pavyzdžiui DES rakto ilgis aštuoni baitai, todėl turi dalintis iš aštuonių be liekanos, jei lieka liekana duomenys yra ilginami, tol kol liekana nulis.
3. Turinti simetrinį raktą ir praplėstus duomenis, jie yra praleidžiami pro šifravimo schemą, ir gaunami šifruoti duomenys.

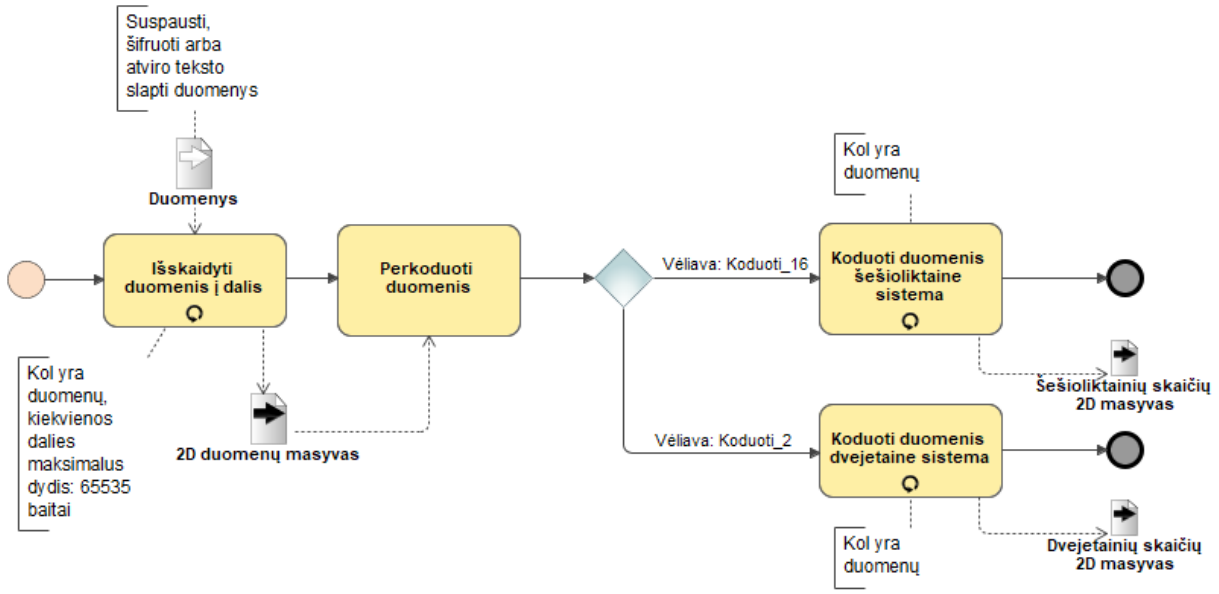


2.9 pav. Slaptų duomenų šifravimo procesas

2.3.3.2. Slaptų duomenų skaidymas ir kodavimas

Norint išsiųsti daugybės baitų ilgio duomenis, gali nepakakti numatyto 65535 TCP lango dydžio, todėl reikėtų duomenis išskaidyti būtent iki šio skaičiaus (nebent nustatomas didesnis lango dydis). Skaidymas atliekamas kaip matoma 2.10 paveiksle.

1. Duomenys išskaidomi į masyvus po maksimaliai 65535 baitus;
2. Tada išskaidytų duomenų masyvas yra perkoduojamas, jei vėliava:
    a. koduoti šešioliktainiais skaičiais – duomenys koduojamai šešioliktainiais skaičiais ir sudedami į naują duomenų masyvą;
    b. koduoti dvejetainiais skaičiais – duomenys koduojamai dvejetainiais skaičiais ir sudedami į naują duomenų masyvą.
3. Galutinis rezultatas – koduotų duomenų 2D masyvas.

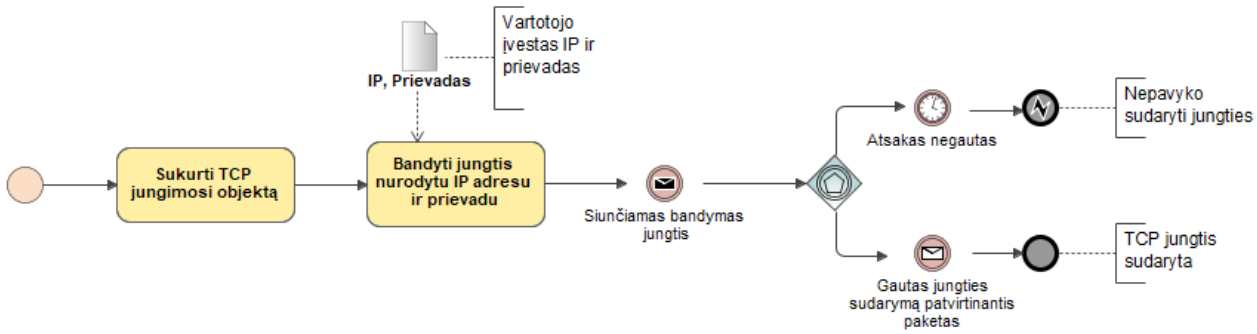


2.10 pav. Slaptų duomenų skaidymo procesas

2.3.3.3. TCP besijungiančios jungties sudarymas

Persiųsti HTTP/2 paketus turima sudaryti, žemesnio lygio tinklo protokolo – TCP jungtį. TCP jungties sudarymo procesas matomas 2.11 paveiksle.

1. Sukuriamas TCP jungimosi objektas;
2. Pagal vartotojo įvestus IP ir prievadą bandoma jungtis į serverį;
3. Pagal įvykusį įvykį gali atsitikti vienas iš dviejų dalykų:
    a. negavus atsako, pavaizduojama žinutė, kad sudaryti TCP jungties nepavyko;
    b. gavus atsakymą, sudaroma TCP jungtis su serveriu.



2.11 pav. TCP besijungiančios jungties sudarymo procesas

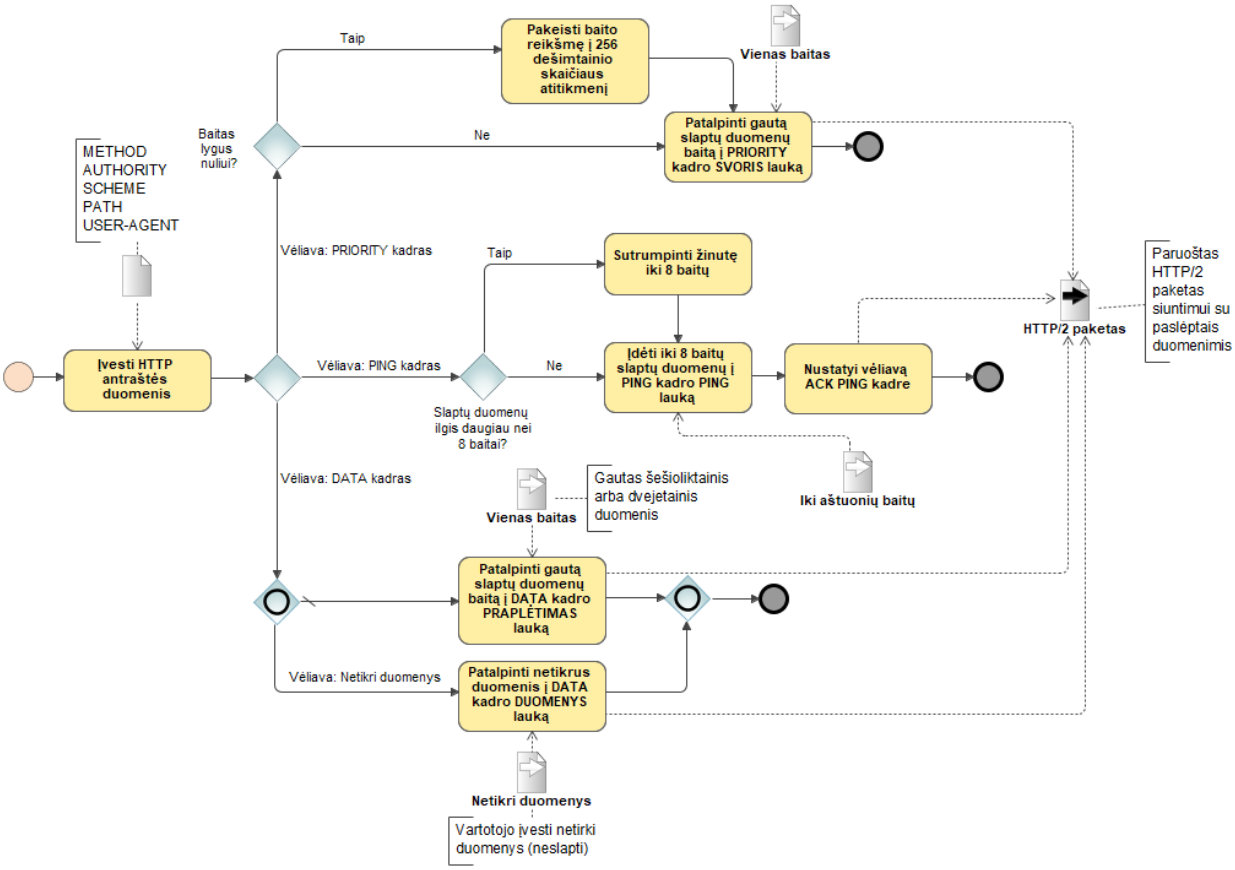
2.3.3.4. HTTP/2 kadrų su įstatytais slaptais duomenimis sudarymas

Sudaryti slaptus kanalus HTTP/2 protokolu naudojami *DATA*, *PRIORITY* ir *PING* kadrų laukai aprašyti 2.1 skyriuje. Slaptų duomenų įstatymo į HTTP/2 kadrus veiklos procesas matomas 2.12 paveiksle. Slaptų duomenų įdėjimas į kadrus vyksta taip:

1. Pirmiausiai nustatoma HTTP antraštė;
2. Toliau, tikrinama kokį kadrą vartotojas pasirinko slaptiems duomenims įstatyti, jei:
    a. DATA – į DATA kadro „Praplėtimas“ lauką įdedamas baitas koduotų duomenų, taip pat, jei vartotojas nusprendė, į „Duomenys“ lauką galima įstatyti netikrus duomenis, kad išsiųstas kadras atrodytų labiau įtykinamai;
    b. PRIORITY – patikrinama ar gautas baitas lygus nuliui, jei taip, reikšmė pakeičiama į 256 („Svoris“ laukas galimos reikšmės nuo 1 iki 256) ir įstatoma į „Svoris“ lauką, jei reikšmė nelygi nuliui, tokia ir įstatoma;

c. PING – Patikrinama ar gautų duomenų ilgis iki 8 baitų, jei ne, duomenys sutrumpinami iki 8 baitų ir tada įstatomi į „Ping“ lauką, jei duomenys buvo iki 8 baitų, tokie ir įstatomi. Po to, kad serveris priimtų „Ping“ kadrą kaip atsaką, kadre nustatoma ACK vėliava.

3. HTTP/2 paketas su įstatyta slaptų duomenų dalimi paruoštas išsiuntimui.



2.12 pav. HTTP/2 kadrų formavimas ir slaptų duomenų įdėjimas

2.3.4. Serverio baseino veiklos procesai

Serverio procesas atsakingas už ateinančių TCP jungčių klausymąsi ir slaptų duomenų išėmimą iš HTTP/2 kadrų, bei jų sudėjimą į žmogui skaitomą formą.

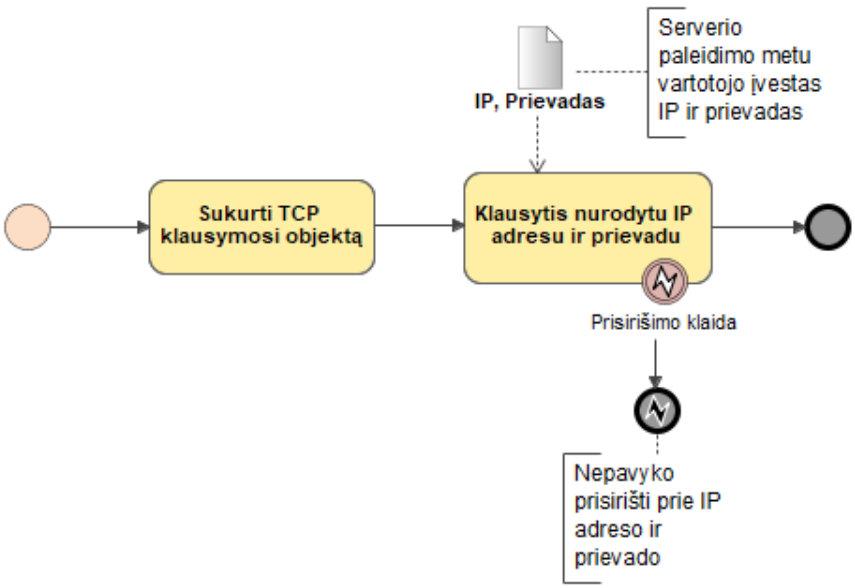
Plačiau apie serverio atliekamus procesus (2.8 paveikslas, “HTTP/2 serveris” baseinas):

1. Iš vartotojo gavęs komandą pradėti, serveris bando sukurti besiklausančią TCP jungtį, kaip matoma 2.13 paveikslo procese;
2. Sukūrus besiklausančią TCP jungtį, serveris laukia jungčių, kaip parodyta 2.14 paveikslo procese;
3. Gavus HTTP/2 srautą pirmiausiai identifikuojamas HTTP/2 sraute perneštas įvykis, pavyzdžiui: „Gauti duomenys“, „Srauto pabaiga“;
4. Jei gautas įvykis – „Gauti duomenys“ vienas prie kito yra pridedami duomenys, kol negaunamas „Srauto pabaiga“ įvykis;
5. Kai gaunamas „Srauto pabaiga“ įvykis atliekamas duomenų atkodavimo procesas matomas paveiksle 2.15;
6. Atkodavus duomenis, jei serveriui buvo paduota šifravimo vėliavėlė su slapta fraze, duomenys yra iššifruojami, kaip matoma 2.16 paveikslo vaizduojamame procese;
7. Galiausiai, slapti duomenys yra atvaizduojami į CLI.

2.3.4.1. TCP besiklausančios jungties sudarymas

Iš kliento norint gauti ateinančius prisijungimus pirmiausiai reikia, kad serveris klausytųsi TCP jungčių, besiklausančios jungties sudarymas matomas 2.13 paveiksle vaizduojamame procese.

1. Sukuriamas TCP klausymosi objektas;
2. TCP objektu bandoma pririšti prie vartotojo nurodyto IP ir prievado:
    a. jei prisirišti pavyksta, sudaryta besiklausanti TCP jungtis;
    b. jei prisirišti nepavyksta, pavaizduojamas klaidos pranešimas CLI.

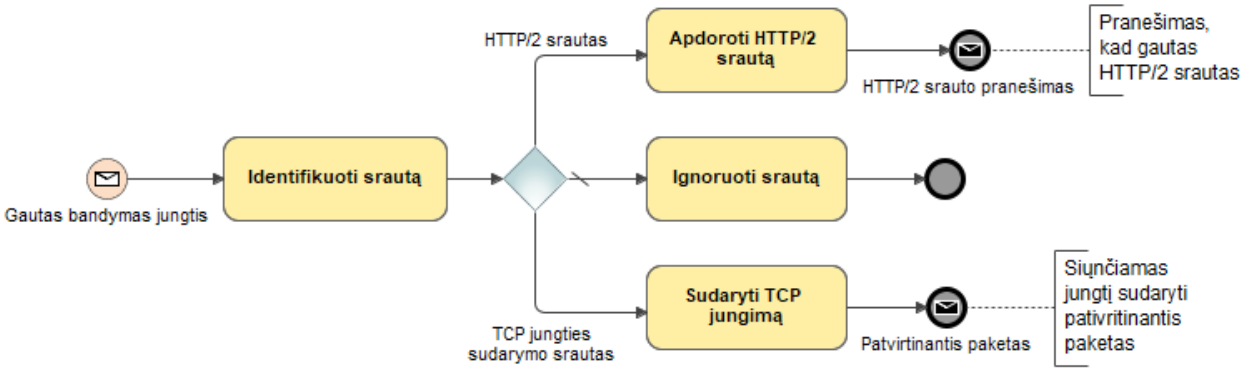


2.13 pav. TCP besiklausančios jungties sudarymo procesas

2.3.4.2. TCP jungčių klausymasis

Į skirtingą ateinantį srautą reikia reaguoti atitinkamai, kas atliekama su srautu skirtingais atvejais matoma 2.14 paveiksle vaizduojamame procese.

1. Pirmiausiai identifikuojamas srautas, jei tai:
    a. HTTP/2 srautas – pradėti jo apdorojimą;
    b. bandymas sudaryti TCP jungtį – siųsti patvirtinančius paketus klientui;
    c. kitas srautas – ignoruoti visą kitą srautą.

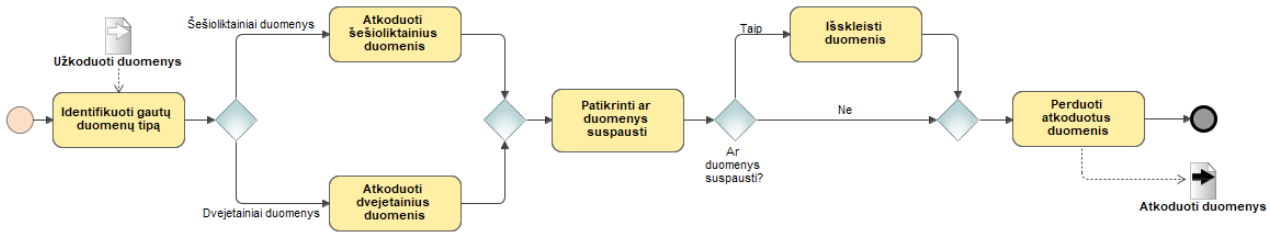


2.14 pav. TCP jungčių klasuymosi procesas

2.3.4.3. Slaptų duomenų atkodavimas

Sudėjus duomenis į vieną visumą galima pradėti jų atkodavimą, kaip matoma 2.15 paveiksle vaizduojamame procese.

1. Identifikuojamas duomenų kodavimas;
2. Pagal identifikuotą kodavimą atliekamas atkodavimas;
3. Tada patikrinama ar duomenys turi suspaudimo antraštę, jei taip, jie yra iškleidžiami;
4. Atkoduoti duomenys perduodami tolimesniems veiksmams.

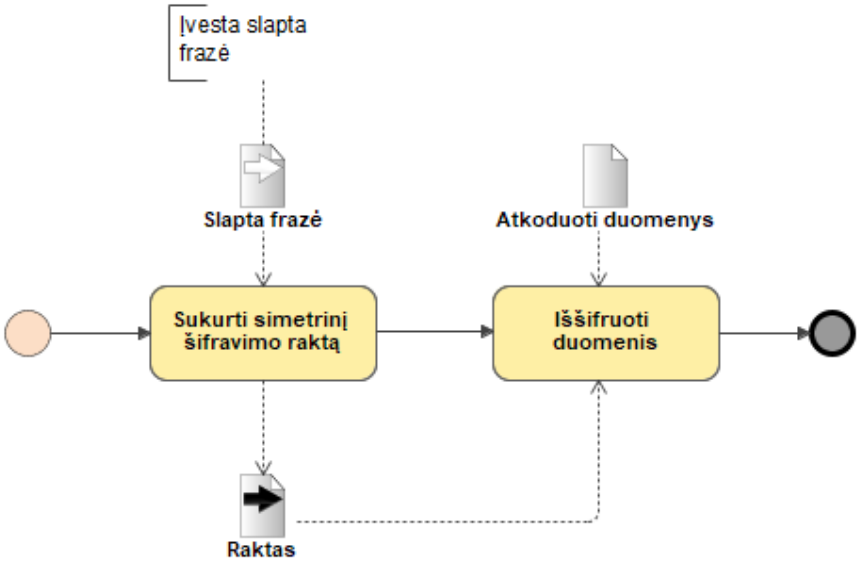


2.15 pav. Slaptų duomenų atkodavimo ir išskleidimo procesas

2.3.4.4. Slaptų duomenų iššifravimas

Gavus užšifruotus duomenis, norint juos perversti į žmogui skaitomą formatą, reikia, kad vartotojui įjungus serverį būtų buvusi įvesta šifravimo vėliava ir slapta frazė. Iššifravimas atliekamas kaip matoma 2.16 paveiksle vaizduojamame procese.

1. Su įvesta slapta fraze sukuriamas simetrinis iššifravimo raktas;
2. Į iššifravimo schemą paduodamas iššifravimo raktas ir šifruoti duomenys;
3. Grąžinami iššifruoti duomenys.



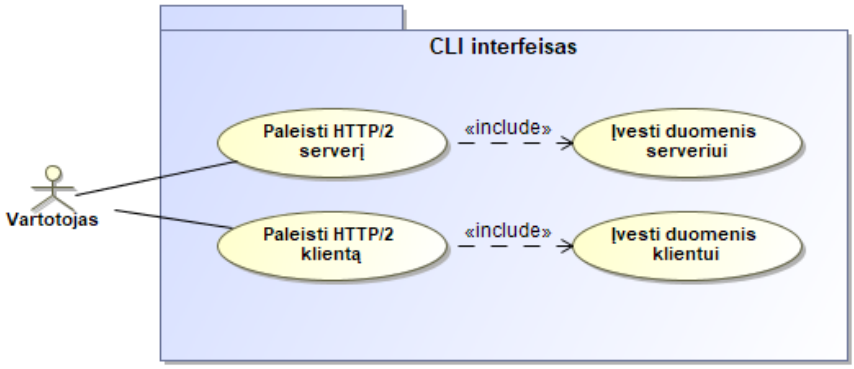
2.16 pav. Slaptų duomenų iššifravimo procesas

2.4. HTTP/2 steganografinio metodo panaudos atvejai

HTTP/2 steganografinio metodo prototipo panaudos atvejai išskirti į penkias pakuotes pagal vartotoją:

1. Vartotojas (Žmogus);
2. Klientas, tinklo jungčių ir duomenų apdorojimo (Sistema veikianti kliento režimu);
3. Serveris, tinklo jungčių ir duomenų apdorojimo (Sistema veikianti serverio režimu).

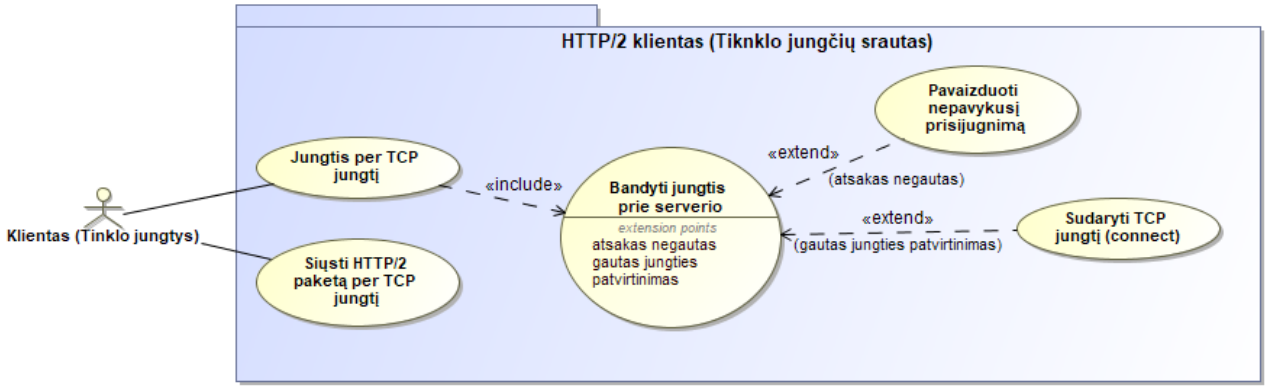
2.17 paveiksle matomi vartotojo panaudos atvejai. Vartotojas gali arba paleisti serverį, arba paleisti klientą, abejais atvejais vartotojas turi įvesti kitus reikalingus duomenis.



2.17 pav. Vartotojo panaudos atvejai

2.18 paveiksle matomi kliento panaudos atvejai susiję su tinklo jungtimis. Klientas gali:

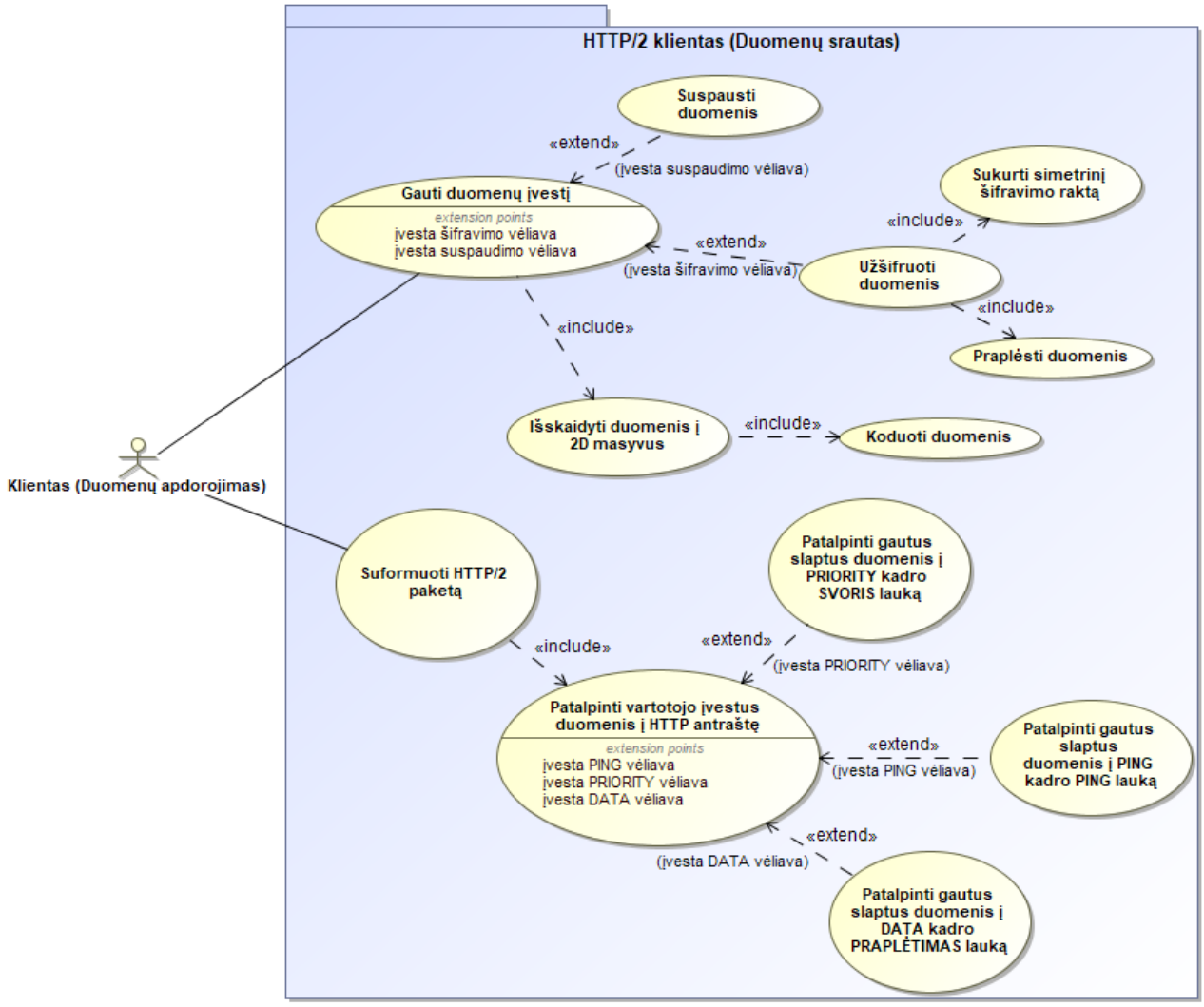
- Jungtis per TCP jungtį, iš to seka, kad arba TCP jungtis arba sudaroma, arba nepavykimo atveju vaizduojamas klaidos pranešimas;
- Siųsti HTTP/2 paketą (su paslėptais duomenimis) per TCP jungtį.



2.18 pav. Kliento tinklo jungčių panaudos atvejai

2.19 paveiksle matomi kliento panaudos atvejai susiję su duomenų apdorojimu. Klientas gali:

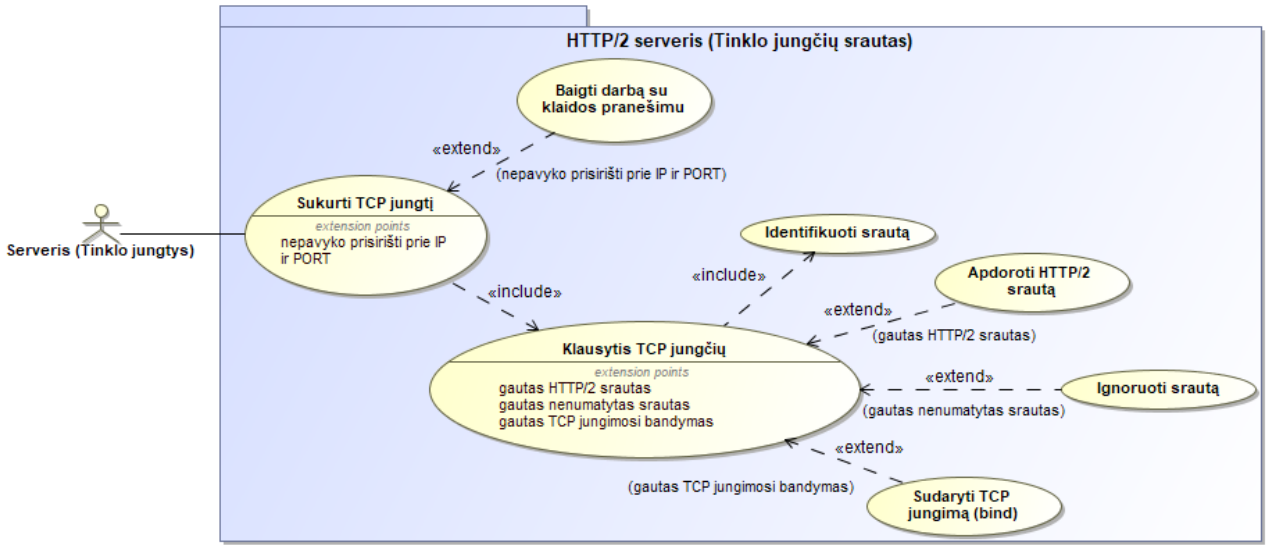
- Gauti įvestus duomenis, kurie bus išskaidyti į 2D masyvus ir užkoduoti;
- Suformuoti HTTP/2 paketą ir pagal vartotojo pasirinktį paslėpti duomenis:
  - *DATA* kadre;
  - *PRIORITY* kadre;
  - *PING* kadre.



2.19 pav. Kliento duomenų apdorojimo panaudos atvejai

2.20 paveiksle matomi serverio panaudos atvejai susiję su tinklo jungtimis. Serveris gali:

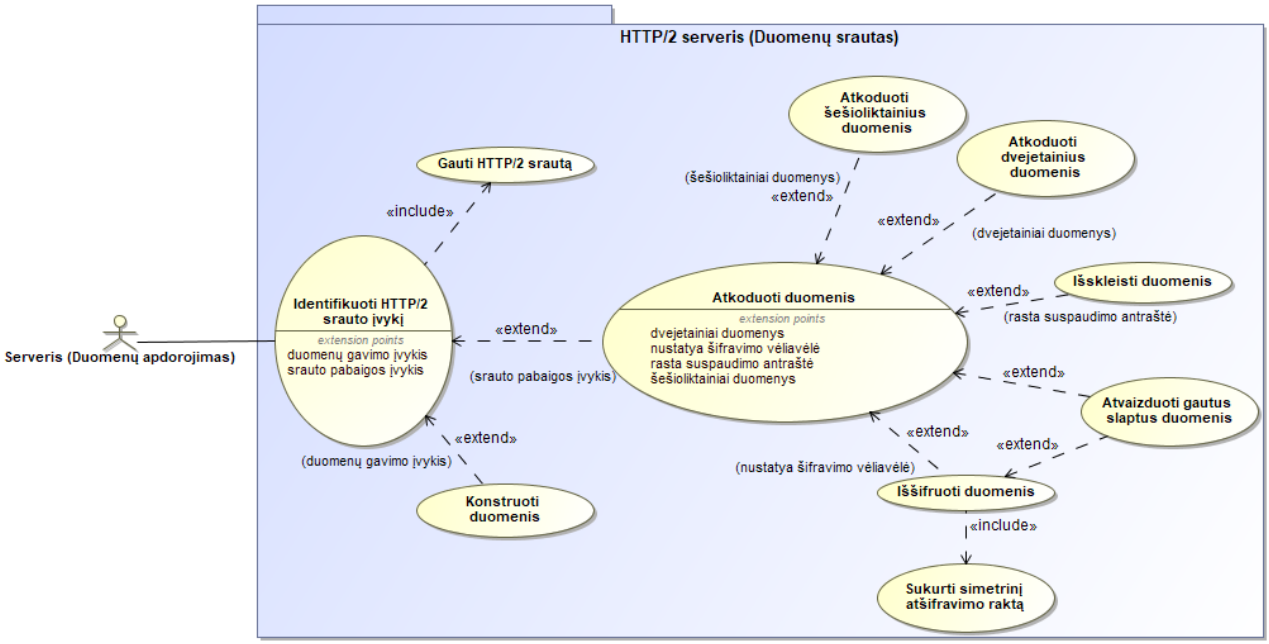
- Sukurti TCP jungtį (ją užbaigti nesėkmės atveju);
- Sukūrus TCP jungtį jos klausytis ir reaguoti į įvairias sąlygas pagal srautą.



2.20 pav. Serverio tinklo jungčių panaudos atvejai

2.21 paveiksle matomi serverio panaudos atvejai susiję su duomenų apdorojimu. Serveris gali:

- Gauti HTTP/2 srautą;
- Identifikuoti HTTP/2 įvykius, pagal juos konstruoti duomenis;
- Atkoduoti duomenis iš šešioliktainio ir dvejetainio kodavimo;
- Išskleisti duomenis;
- Sudaryti simetrinį iššifravimo raktą ir iššifruoti duomenis;
- Atvaizduoti gautus slaptus duomenis CLI.



2.21 pav. Serverio duomenų apdorojimo panaudos atvejai

2.5. HTTP/2 steganografinio metodo koncepcinis duomenų modelis

Šiame skyriuje pateikiama kuriamo HTTP/2 steganografinio metodo prototipo klasių diagrama, ji matoma 2.22 paveiksle. Klasių diagrama paremta pagal 2.3 skyriuje modeliuotus „Vartotojas“, „HTTP/2 serveris“ ir „HTTP/2 klientas“ procesų baseinus.

Į „Vartotojas“ klasę brėžiamas generalizacijos ryšys, kuris nurodo, kad „Vartotojas“ gali būti arba „Siuntėjas“ arba „Gavėjas“.

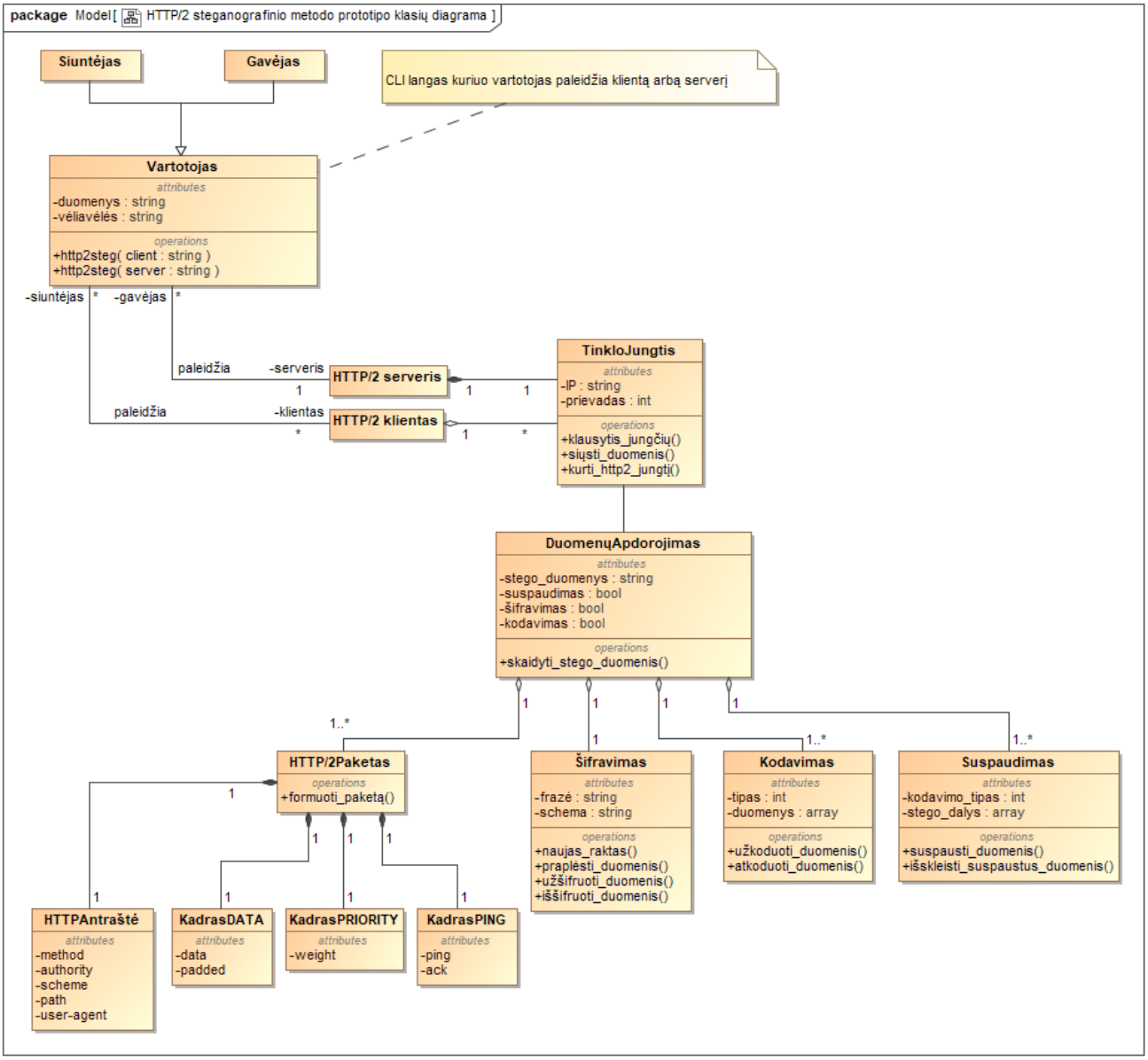
Pradedant prototipo programą, „Varotojas“ – „Gavėjas“ paleidžia „HTTP/2 serveris“ objektą, o „Vartotojas“ – „Siuntėjas“ paleidžia „HTTP/2 klientas“ objektą.

„HTTP/2 serveris“ su „TinkloJungtis“ susietas kompozicija, nes mirus „TinkoJungtis“ objektui turi mirti ir „HTTP/2 serveris“.

„HTTP/2 klientas“ su „TinkloJungtis“ susietas agregacijos ryšiu, nes mirus „TinkoJungtis“ objektui, „HTTP/2 klientas“ nebūtinai miršta.

„TinkloJungtis“ ir „DuomenųApdorojimas“ sujungtos asociacijos ryšiu, nes yra susijusios ir bendradarbiauja duomenų persiuntime.

Likusios klasės yra „DuomenųApdorojimas“ vaikai, į šią klasę sujungtos agregaciją, pagal galimas objektų instancijas.



2.22 pav. HTTP/2 steganografinio metodo prototipo klasių diagrama

2.6. HTTP/2 steganografinio metodo grafinės vartotojo sąsajos prototipas

HTTP/2 steganografinio metodo prototipas leidžiamas per CLI, todėl tikro GUI neturi. Kaip prototipas atrodo CLI sąsajoje vartotojui paleidus prototipą su „-h“ (pagalba) vėliava matoma 2.23 paveiksle, šiuo atveju, CLI sąsajoje vartotojui pavaizduojamos kitos galimos įvesti vėliavos ir kiti duomenys, kurie turi būti įvesti.

```
orion1vi /net/192.168.1.4/mnt/Yukino/Repos/StegHTTP2 $ python2 http2steg.py -h
usage: http2steg.py [-h] [-c | -s] [-a ADDRESS] [-m MESSAGE] [-p PORT] [-z]
                    [-M {GET,POST}] [-A AUTH] [-S {http,https}] [-P PATH]
                    [-U USER-AGENT] [-d] [-r] [-o] [-x] [-k KEY] [-f DATA]
                    [-t TIME]

HTTP/2 Steganography Client/Server

optional arguments:
  -h, --help            show this help message and exit
  -c, --client          run as steganography client
  -s, --server          run as steganography server
  -a ADDRESS, --address ADDRESS
                        source or destination depending on whether client or
                        server mode is used
  -m MESSAGE, --message MESSAGE
                        secret message
  -p PORT, --port PORT  server port, default = 80
  -z, --compress        compress parts of secret message
  -M {GET,POST}, --header-method {GET,POST}
                        set method, e.g. GET, POST
  -A AUTH, --header-authority AUTH
                        set authority
  -S {http,https}, --header-scheme {http,https}
                        set connection scheme, e.g. http, https
  -P PATH, --header-path PATH
                        set path that method will connect to, e.g. /
  -U USER-AGENT, --header-user-agent USER-AGENT
                        spoof user agent
  -d, --data-frame      use padding of data frame as a covert channel
  -r, --priority-frame  use priority frame as a covert channel
  -o, --ping-frame      use ping frame as a covert channel
  -x, --hex             send hex arrays instead of bit arrays (faster
                        throughput)
  -k KEY, --enc-key KEY
                        set DES symmetric encryption key
  -f DATA, --fake-data DATA
                        set fake data that will be sent through non covert
                        channels, default = '1'
  -t TIME, --delay TIME
                        set delay time between sent stego bits in seconds,
                        float is supported
```

2.23 pav. HTTP/2 steganografinio metodo prototipo CLI sąsaja

Komandų pavyzdžiai:

- Paleisti serverį, kuris sudės šešioliktainės koduotės duomenis, bandys duomenis iššifruoti ir išskleisti:
  - *python2 http2steg.py -s -x -z -k 12345678.*
- Siųsti slaptą žinutę serveriui, esančiam vietiniame tinkle, panaudojant *DATA* kadrą, koduojant duomenis šešioliktaine koduote, juos suspaudžiant ir užšifruojant:
  - *python2 http2steg.py -c -a localhost -m slaptažinutė -d -x -z -k 12345678.*

2.7. HTTP/2 steganografinio metodo projektavimo išvados

Skyriuje apsibrėžti HTTP/2 kadrai ir jų protokolo laukų dalys, kuriomis manoma, kad įmanoma sudaryti kanalus persiųsti slaptoms žinutėms. Apsibrėžti kadrai – *DATA*, *HEADERS*, *PUSH PROMISE*, *PRIORITY*, *PING* ir *SETTINGS*, iš kurių, slaptiems duomenų mainams pavyko panaudoti *DATA, PRIORITY* ir *PING* kadrus. Tai nebūtinai reiškia, kad kitais kadrais neįmanoma sudaryti slaptų kanalų, kitų kadrų, prototipo projektavimo metu, steganografinių galimybių tiesiog nespėta išbandyti. Taip pat, buvo apsibrėžtos, tik akivaizdžiausiai dominančios vietos slaptų kanalų sudarymui, ilgiau pasigilinus į RFC apibrėžiantį HTTP/2 protokolą, turėtų pavykti rasti ir kitų steganografijai išnaudojamų kadrų ir jų laukų.

3. HTTP/2 STEGANOGRAFINIO METODO TYRIMAS

3.1. Prototipas ir tyrimo metodika

Norint ištirti siūlomą HTTP/2 steganografinį metodą, pirmiausiai prototipas realizuojamas programiškai. Realizavus prototipą programiškai (priedas 6.1), tyrimui, pagal vertinamas savybes, sukuriami scenarijai surinkti HTTP/2 paketus, juos apdoroti ir gauti išvestį grafikais (priedas 6.2). Šio tyrimo metu ir prototipas, ir duomenims surinkti skirti scenarijai buvo realizuoti panaudojus *Python 2.7* programavimo kalbą.

3.2. Tyrimo metu analizuojamos savybės

Steganografinį metodą nuspręsta tirti pagal tris pagrindines visus steganografinius metodus apsakančias savybes:

- talpą;
- neaptinkamumą;
- tvirtumą.

Steganografinio metodo talpa nusako kiek duomenų galima išsiųsti sudarytais slaptais kanalais, paprastai bitų ar baitų per paketą. Ištirti HTTP/2 steganografinio metodo talpą, naudojamas tinklo srauto stebėjimo įrankis ir RFC 7540 dokumentas [24] patvirtinti kadrų struktūras ir įstatomų duomenų ilgį.

Steganografinio metodo neaptinkamumas, tai jo atsparumas nuo aptikimo – steganalizės. Analizuoti steganografinių metodų neaptinkamumą dažnai yra panaudojamas entropijos skaičiavimas [25] [26] [27] [28] [29] [30] [31] [32] [33], šiame tyrime, taip pat, nuspręsta panaudoti šį tyrimo metodą.

Kai entropija minima informacinių technologijų pasaulyje, paprastai, iš tiesų, kalbama apie *Shannon* entropiją, kuria galima apskaičiuoti kiek informacijos reikia, kad būtų išreikštas simbolis. Pavyzdžiui, jei tekstas suspaustas ar užšifruotas, turėtų būti, tikimasi gauti aukštesnį entropijos lygį, nei, kad atviro teksto atveju.

Shannon entropija išreiškiama, tokia (1) formule [34]:

$$H_2(\mathrm{X}) = -\sum_{i=0}^{n} \frac{kiekis_i}{N} \log_2\left(\frac{kiekis_i}{N}\right); \quad (1)$$

X – tekstas (paketo turinys).

N – teksto ilgis (paketo turinio ilgis).

$kiekis_i$ – simbolio kiekis tekste (paketo turinyje).

$\log_2$ – logaritmo pagrindas nurodo kiek skirtingų simbolių reikia norint išreikšti tekstą, jei pagrindas 2, tai matuojama dvejetainiais skaičiais, todėl gauto atsakymo entropija vienetais yra bitai per simbolį arba bitai per baitą.

HTTP/2 paketuose esančių duomenų entropija, tyrimo metu, skaičiuojama atsižvelgiant į:

1. HTTP/2 naudojamą kadro tipą duomenims slėpti:
    a. *DATA*;
    b. *PRIORITY*;
    c. *PING*.
2. Naudojamas papildomas funkcijas:
    a. suspausti duomenys;
    b. užšifruoti duomenys.
3. Naudojamą duomenų kodavimo būdą:
    a. šešioliktainiais skaičiais;
    b. dvejetainiais skaičiais.

Steganografinio metodo tvirtumas, tai slaptos žinutės vientisumo išsilaikymas komunikacijos metu. Vienas labiausiai siūlomų metodų, kaip apsaugoti tinklą nuo slaptų kanalų sudarymų įvairiais protokolais yra srauto normalizatoriai, kurių tikslas protokolo laukuose pašalinti dviprasmybes, pavyzdžiui, beveik nenaudojamas protokolų dalis ištrinti ar užrašyti nuliais, taip užkertant kelią slaptų kanalų kūrimui [35]. Steganografinio metodo tvirtumo tyrimu patikrinama ar protokolo laukai kuriais kuriami slapti kanalai yra normalizuojami ar kitaip paveikiami:

1. Vietiniame tinkle
2. Išoriniame tinkle

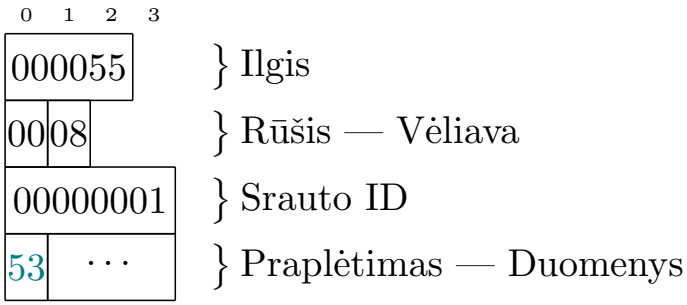
Kadangi, neabejotinai svarbiausia steganografinių metodų savybė yra neaptinkamumas, šiai savybei ištirti skiriama daugiausiai dėmesio.

3.3. HTTP/2 steganografinio metodo talpos tyrimas

Steganografinio metodo talpos tyrimu siekiama patikrinti ar paketu išsiunčiamos informacijos kiekis atitinka teorinę talpą, tuo pačiu įsitikinti, kad prototipu išsiunčiami kadrai atitinka RFC 7540 nurodytas kadrų struktūras. Tiriama *DATA*, *PRIORITY* ir *PING* kadrų talpa. Terminale paleidžiamas *tcpdump* įrankis stebėti tinklo srautą, tada realizuotu prototipu nustatomas kadro tipas, kuriuo išsiunčiama slapta žinutė, kurios tekstas – „SLAPTA ŽINUTĖ“ (be kabučių). Galiausiai, srauto stebėjimo įrankio surinktuose duomenyse, pasirėmus RFC 7540, nusistatomi kadro rėmai ir patikrinama kiek viename kadre buvo įdėta informacijos.

3.3.1. *DATA* kadro struktūra ir talpa

Į *DATA* kadrą įdėjus slaptus duomenis, tikimasi, kad pirmasis kadras atitiks 3.1 paveiksle pavaizduotą kadrą. Tikimasi, kad pirmi trys baitai laikys ilgio reikšmę, *DATA* kadro ilgio reikšmė lygi praplėtimo lauko ilgio, duomenų ilgio (prototipu siunčiamas vienas baitas, jei nenustatoma kitaip) ir pačio praplėtimo ilgio sumai, ši suma, slaptai siunčiamos „S“ raidės atveju būtų – 000055. Ketvirto baito reikšmė turi būti 00, nes šia reikšme apibūdinamas *DATA* kadro tipas. Penktame baite reikšmė turi būti 08, nes tai vėliava *PAD*, kuri nusako, kad šis kadras yra su praplėtimu. Nuo šešto iki devinto baito įrašoma srauto identifikacijos reikšmė, jei tai naujas srautas, *DATA* kadre reikšmė visada prasidės nuo 00000001. Dešimtame baite turėtų būti regima įdėtų slaptų duomenų reikšmė – 53, tai šešioliktainio kodavimo atitikmuo raidei „S“.



3.1 pav. HTTP/2 *DATA* kadro struktūra, kai įdėta slapta informacija – „S“ raidė

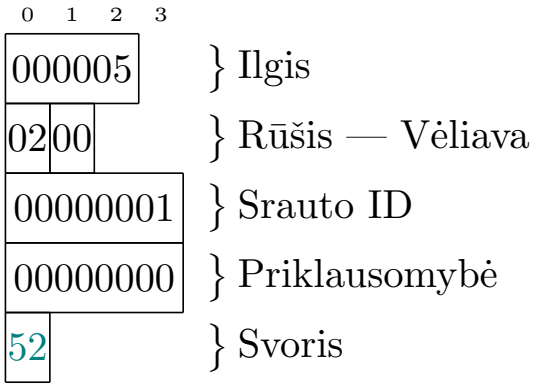
Pirmas išsiųstas *DATA* kadras prototipu matomas 3.2 paveiksle, spalva pažymėta vieta *DATA* kadro antraštė ir duomenys. Kadro duomenys atitiko aprašytus numanomus, daroma išvada, kad prototipu siunčiamas *DATA* kadras atitinka RFC 7540 aprašomą kadro struktūrą ir panaudojus praplėtimo lauką galima išsiųsti 1 baitą slaptų duomenų per kadrą.

```
02:49:36.924613 IP vilius-mac.lan.54496 > k55vd-solus.lan.http: Flags [P.], seq 0:183, ack 1, win 2058, options [nop,nop
,TS val 1088637550 ecr 2962235006], length 183: HTTP
        0x0000:  4502 00eb 0000 4000 4006 0000 c0a8 0103  E.....@.@.......
        0x0010:  c0a8 0106 d4e0 0050 dafa afc8 42ef b258  .......P....B..X
        0x0020:  8018 080a 8437 0000 0101 080a 40e3 4a6e  .....7......@.Jn
        0x0030:  b090 1e7e 5052 4920 2a20 4854 5450 2f32  ...~PRI.*.HTTP/2
        0x0040:  2e30 0d0a 0d0a 534d 0d0a 0d0a 0000 2404  .0....SM......$.
        0x0050:  0000 0000 0000 0100 0010 0000 0200 0000  ................
        0x0060:  0100 0300 0000 6400 0400 00ff ff00 0500  ......d.........
        0x0070:  0040 0000 0600 0100 0000 000b 0104 0000  .@..............
        0x0080:  0001 8241 86a0 e41d 139d 0987 8400 0055  ...A...........U
        0x0090:  0008 0000 0001 5331 0000 0000 0000 0000  ......S1........
        0x00a0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x00b0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x00c0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x00d0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x00e0:  0000 0000 0000 0000 0000 00              ..........
```

3.2 pav. HTTP/2 *DATA* kadro struktūros vaizdas *tcpdump* įrankyje

3.3.2. *PRIORITY* kadro struktūra ir talpa

PRIORITY kadre sutalpinus slaptus duomenis, pirmasis kadras turėtų atitikti 3.3 paveiksle pavaizduotą kadrą. Pirmi trys baitai privalo būti 000005, nes *PRIORITY* kadro ilgis nusakomas penkiais oktetais. Ketvirtas baitas – 02, tai *PRIORITY* kadro tipo reikšmė. Penktas baitas 00, *PRIORITY* kadras nepalaiko jokių vėliavų. Nuo šešto iki devinto baito, srauto, kuriam nustatomas prioritetas, reikšmė, jei prioritetas nustatomas pirmam srautui reikšmė yra 00000001. Nuo dešimto iki keturiolikto baito, priklausomybės reikšmė, jei srautas nepriklauso nuo jokio kito, reikšmė bus 0000000. Paskutinis baitas, tai svorio reikšmė, kurioje turėtų būti įstatomi slapti duomenys – 53 („S“), tačiau tinkama svorio reikšmė yra nuo 1 iki 256, todėl įstatomų duomenų reikšmė sumažinama vienetu – 52.



3.3 pav. HTTP/2 *PRIORITY* kadro struktūra, kai įdėta slapta informacija – „S“ raidė

Pirmas išsiųstas *PRIORITY* kadras prototipu matomas 3.4 paveiksle, spalva pažymėta vieta *PRIORITY* kadro antraštė ir duomenys. Kadro duomenys atitiko aprašytus numanomus, daroma išvada, kad prototipu siunčiamas *PRIORITY* kadras atitinka RFC 7540 aprašomą kadro struktūrą ir panaudojus svorio lauką galima išsiųsti 1 baitą slaptų duomenų per kadrą.

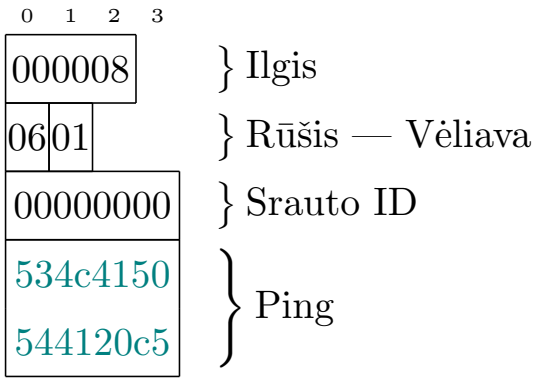
```
03:12:52.060186 IP vilius-mac.lan.54649 > k55vd-solus.lan.http: Flags [P.], seq 0:103, ack 1, win 2058, options [nop,nop
,TS val 1090026675 ecr 2963630141], length 103: HTTP
        0x0000:  4502 009b 0000 4000 4006 0000 c0a8 0103  E.....@.@.......
        0x0010:  c0a8 0106 d579 0050 71c6 7ac4 9103 ff4f  .....y.Pq.z....O
        0x0020:  8018 080a 83e7 0000 0101 080a 40f8 7cb3  ............@.|.
        0x0030:  b0a5 683d 5052 4920 2a20 4854 5450 2f32  ..h=PRI.*.HTTP/2
        0x0040:  2e30 0d0a 0d0a 534d 0d0a 0d0a 0000 2404  .0....SM......$.
        0x0050:  0000 0000 0000 0100 0010 0000 0200 0000  ................
        0x0060:  0100 0300 0000 6400 0400 00ff ff00 0500  ......d.........
        0x0070:  0040 0000 0600 0100 0000 000b 0104 0000  .@..............
        0x0080:  0001 8241 86a0 e41d 139d 0987 8400 0005  ...A............
        0x0090:  0200 0000 0001 0000 0000 52              ..........R
```

3.4 pav. HTTP/2 *PRIORITY* kadro struktūros vaizdas *tcpdump* įrankyje

3.3.3. *PING* kadro struktūra ir talpa

Kai į *PING* kadrą sutalpinami slapti duomenys, kadras turėtų būti atitikmuo kadrui pavaizduotam 3.5 paveiksle. Pirmi trys baitai – 000008, tai ilgio reikšmė, aštuoni, nes *ping* laukas sudarytas iš aštuonių oktetų. Ketvirto baito reikšmė turėtų būti 06, tokia reikšmė aprašo *PING* kadro tipą. Penktas baitas vėliava, kadangi siunčiami slapti duomenys, priverstinai turi būti nustatyta *ACK* (atsakymas į *ping*) vėliavėlė, todėl lauko reikšmė turėtų atitikti 01. Nuo šešto iki devinto baito srauto identifikacijos reikšmė, kuri *PING* kadro atveju visada turi būti lygi 00000000. Sekantys aštuoni baitai turėtų būti slaptos žinutės reikšmės:

1. 53 – „S";
2. 4c – „L";
3. 41 – „A";
4. 50 – „P";
5. 54 – „T";
6. 41 – „A";
7. 20 – tarpas;
8. c5 – pirmas „Ž" baitas, kadangi, tai dviejų baitų ilgio simbolis.



3.5 pav. HTTP/2 *PING* kadro struktūra, kai įdėta slapta informacija – „SLAPTA Ž"

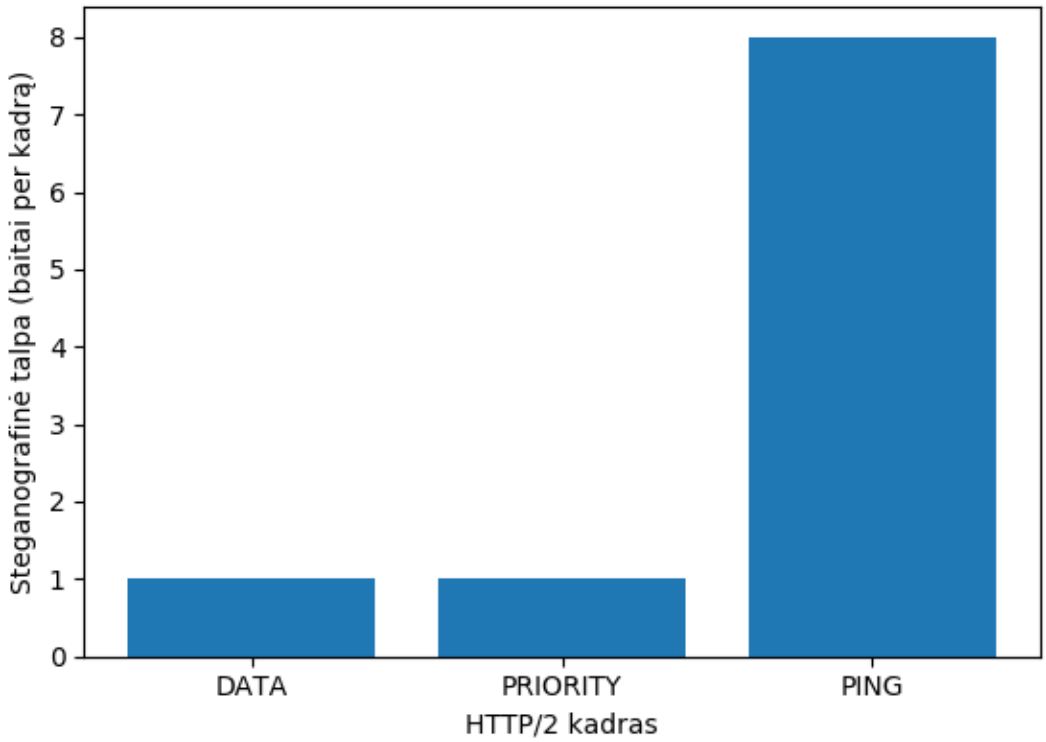
Pirmas išsiųstas *PING* kadras prototipu matomas 3.6 paveiksle, spalva pažymėta vieta *PING* kadro antraštė ir duomenys. Kadro duomenys atitiko aprašytus numanomus, daroma išvada, kad prototipu siunčiamas *PING* kadras atitinka RFC 7540 aprašomą kadro struktūrą ir panaudojus *ping* lauką galima išsiųsti iki 8 baitų slaptų duomenų.

```
03:23:46.071160 IP vilius-mac.lan.54695 > k55vd-solus.lan.http: Flags [P.], seq 0:106, ack 1, win 2058, options [nop,nop
,TS val 1090678908 ecr 2964284155], length 106: HTTP
        0x0000:  4502 009e 0000 4000 4006 0000 c0a8 0103  E.....@.@.......
        0x0010:  c0a8 0106 d5a7 0050 f898 e150 8ebd c6c5  .......P...P....
        0x0020:  8018 080a 83ea 0000 0101 080a 4102 707c  ............A.p|
        0x0030:  b0af 62fb 5052 4920 2a20 4854 5450 2f32  ..b.PRI.*.HTTP/2
        0x0040:  2e30 0d0a 0d0a 534d 0d0a 0d0a 0000 2404  .0....SM......$.
        0x0050:  0000 0000 0000 0100 0010 0000 0200 0000  ................
        0x0060:  0100 0300 0000 6400 0400 00ff ff00 0500  ......d.........
        0x0070:  0040 0000 0600 0100 0000 000b 0104 0000  .@..............
        0x0080:  0001 8241 86a0 e41d 139d 0987 8400 0008  ...A............
        0x0090:  0601 0000 0000 534c 4150 5441 20c5       ......SLAPTA..
```

3.6 pav. HTTP/2 *PING* kadro struktūros vaizdas *tcpdump* įrankyje

3.3.4. *DATA, PRIORITY, PING* talpos tyrimo rezultatai

Atlikus *DATA*, *PRIOTITY* ir *PING* kadrų talpos tyrimą patvirtinta, kad *DATA* ir *PRIORITY* kadrai gali talpinti vieną baitą per kadrą slaptai informacijai siųsti, tuo tarpu *PING* kadras gali sutalpinti iki aštuonių baitų (talpos grafikas 3.7 paveiksle). Taip pat, įsitikinta, kad prototipu siunčiami HTTP/2 kadrai atitinką RFC 7540 nurodytas specifikacijas.



3.7 pav. HTTP/2 steganografinio metodo talpa pagal naudojamą kadrą

3.4. HTTP/2 steganografinio metodo neaptinkamumo tyrimas

Bendrai, paketų entropijos skaičiavimais siekiama patikrinti ar toks steganalizės būdas gali matematiškai įrodyti slapto kanalo būvimą HTTP/2 paketuose. Tyrime, entropija skaičiuojama išsiųstiems HTTP/2 paketams, juose išfiltravus srautą pagal vieną iš naudojamų kadrų, kurie gali būti *DATA*, *PRIORITY* arba *PING*. Kiekvienam skaičiavimui išgauti paketus, realizuotu prototipu, išsiunčiamas tam tikras kiekis atsitiktinių žodžių iš žodyno pasirinkto kadro sudarytais slaptais kanalais, bei tokio pat ilgio srautas nepanaudojus jokių slaptų kanalų. Išsiunčiamų žodžių kiekis parenkamas atsižvelgiant į kadro tipą ir kodavimą. Entropijos lygis gali svyruoti nuo 0,0 iki 8,0, didesnės entropijos reikšmės gali nurodyti, kad ta duomenų dalis turi daugiau atsitiktinių ar labiau sudėtingų duomenų, nei, kad įprastas dažnai naudojamas ir nuspėjamas tekstas, tai gali būti užuomina apie slapto kanalo egzistavimą.

Surinktų paketų entropijos skaičiavimas vykdomas trejais scenarijais:

1. Skaičiuojant vidutinę visų paketų entropiją;
2. Skaičiuojant entropiją visus paketus, turinčius tam tikrą kadro tipą, sudėjus į vieną;
3. Taip pat, kaip antras scenarijus, tačiau kartu tiriant srautą, kuriame yra pritaikytas kodavimas dvejetainiais skaičiais.

Pirmu scenarijumi siekiama išsiaiškinti, kiek skiriasi entropijos lygiai limituojant skaičiavimus į HTTP/2 kadro rėmus, taip pat, šiuo bandymu tikimasi patikrinti ar grafikuose gali būti vaizdžiai matoma pasikartojanti struktūrą, pagal kurią galėtų spręsti kokio tipo, tai HTTP/2 kadras.

Antruoju scenarijumi siekiama išsiaiškinti, ne tik entropijos lygių skirtumą tarp srauto kuriame buvo sukurtas slaptas kanalas ir kuriame nebuvo, bet ir papildomų funkcijų, kaip, kad šifravimo ir suspaudimo daromą įtaką entropijos lygiams.

Trečiuoju scenarijumi siekiama išsiaiškinti kiek entropijos lygiai gali skirtis nuo pasirinkto duomenų kodavimo būdo.

Visais scenarijais, entropija skaičiuojama blokais, pavyzdžiui, jei nustatomas bloko dydis – 8, pirma iteracija apskaičiuos entropiją esančią nuo pirmo iki aštunto baito, sekanti iteracija

apskaičiuos entropiją esančią nuo antro iki devinto baito ir taip toliau, kol duomenų ilgio nebeužtenka bloko dydžio poslinkiui.

3.4.1. Steganografinio metodo neaptinkamumo tyrimo parametrai (*DATA* kadras)

Paketai turintys *DATA* kadrą surinkti siunčiant:

- 500 atsitiktinių žodžių iš žodyno, kai naudojamas kodavimo būdas – šešioliktainiai skaičiai;
- 50 atsitiktinių žodžių iš žodyno, kai naudojamas kodavimo būdas – dvejetainiai skaičiai.

Vykdant pirmąjį tyrimo scenarijų, nustatomas bloko dydis 8 baitai, taip grafikuose turėtų išryškėti *DATA* kadro antraštės ir duomenų dalių entropijos lygiai, toks bloko dydis apima beveik visą antraštės dalį (3.1 paveikslas). Slapto kanalo kūrimas *DATA* kadre pakeičia bent tris reikšmes – ilgį, vėliavą bei prailginimą.

Vykdant antrąjį tyrimo scenarijų, nustatomas bloko dydis 2048 baitai, tokį bloko dydį nuspręsta pasirinkti, kad skaičiavimai truktų trumpesnį laiko tarpą, taip pat tokio dydžio duomenų blokai turėtų pakankamai išryškinti entropijos lygių skirtumus grafikuose.

Vykdant trečiąjį tyrimo scenarijų, nustatomas bloko dydis 2048 baitai.

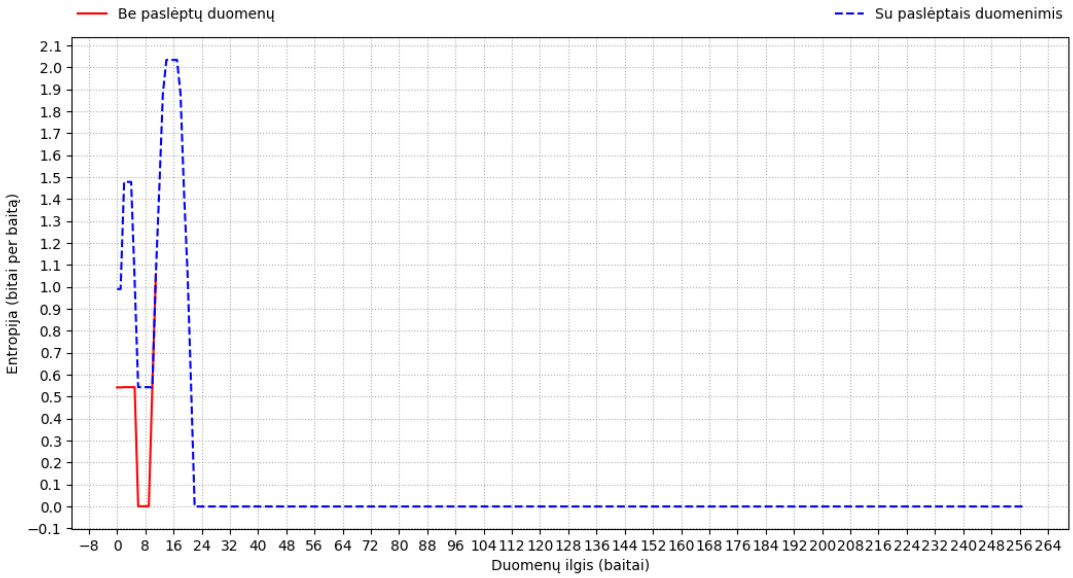
3.4.2. Steganografinio metodo neaptinkamumas (*DATA* kadras)

Entropiją skaičiuojant pirmuoju scenarijumi, iš gautų duomenų, nubrėžtame grafike (3.8 paveikslas), matosi kaip entropijos reikšmės kečiasi antraštės ir duomenų dalyse. Sudarant slaptą kanalą regimi du entropijos lygio pakilimai, pirmas – bloke apimančiame antraštės dalį, kurioje įstatomi slapti duomenys, antras – duomenų dalyje, kuri pasikečia priklausomai nuo ilgio reikšmės. Sraute kuriame paslėptų duomenų nebuvo, entropijos lygių forma labai panaši, tačiau nėra pirmo entropijos lygio pakilimo, nes prailginimo vėliava nėra nustatyta. Kitas pastebimas dalykas – nors matuotas toks pat kiekis paketų, praplėtimo naudojimas praplečia duomenų ilgį, jį užpildo nuliais, todėl regima ilga linija kur entropijos lygis yra 0,0.

Siunčiant slaptus duomenis *DATA* kadru, entropijos reikšmė gali pakilti iki 2,156 bito per baitą, o kai praplėtimas nėra išnaudojamas, iki – 1,061 bito per baitą (3.1 lentelė).

3.1 lentelė. Entropijos lygiai *DATA* kadre (pirmas scenarijus)

Srautas sudarytas iš *DATA* kadrų	Vidutinė entropija (bitas per baitą)			Maksimali entropija (bitas per baitą)			Duomenų ilgis (baitai)
	1 bloke	2 bloke	3 bloke	1 bloke	2 bloke	3 bloke	
Be paslėptų duomenų	0,542	0,001	1,059	0,544	0,544	1,061	9958
Su paslėptais duomenimis	0,99	0,544	2,035	1,061	0,544	2,156	110468



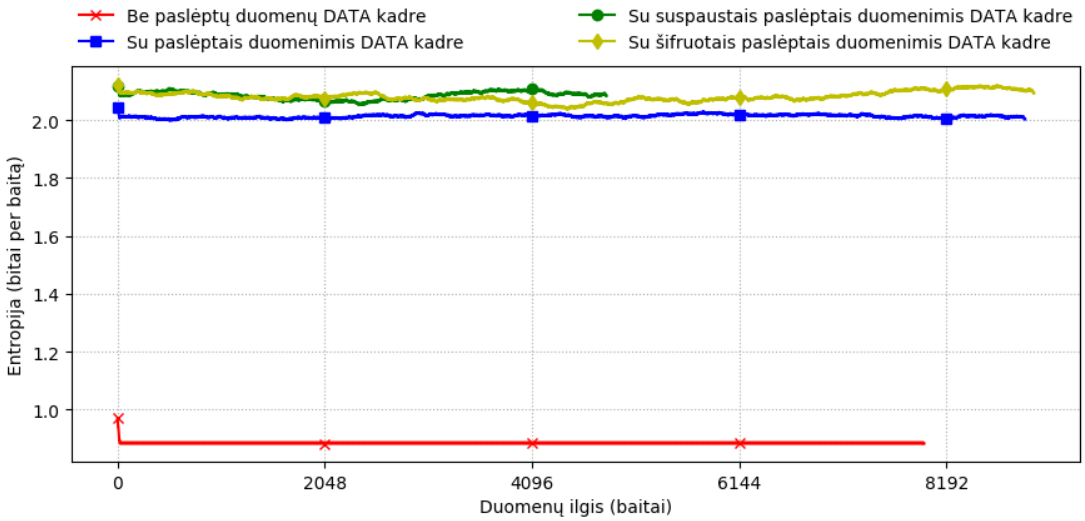
3.8 pav. Entropijos lygiai *DATA* kadre (pirmas scenarijus)

Entropiją skaičiuojant antruoju scenarijumi, iš gautų duomenų, pastebima, kad papildomų funkcijų kaip, kad duomenų šifravimo ir suspaudimo funkcijos, bent šiek tiek, pakelia entropijos lygius lyginant su atviro teksto duomenų siuntimu (3.2 lentelė). Neišnaudojus praplėtimo lauko, entropijos reikšmė apytiksliai 0,884 bito per baitą, siunčiant slaptus duomenis atviru tekstu – 2,013, užšifravus duomenis – 2,08, suspaudus duomenis – 2,084. Maksimalus entropijos lygių skirtumas, tarp srauto su paslėptais duomenimis, kai nenaudojamos papildomos funkcijos ir srauto be paslėptų duomenų, gan didelis – 1,072 bito per baitą (3.2 lentelė).

Stebint linijų formą grafike (3.9 paveikslas) galima pastebėti, kad duomenų šifravimas gali praplėsti siunčiamų baitų kiekį, tuo tarpu duomenų suspaudimas, šiuo atveju lėmė, tai, kad buvo išsiųstą 1,6 karto mažiau baitų, nei, kad siunčiant slaptus duomenis atviru tekstu.

3.2 lentelė. Entropijos lygiai *DATA* kadre (antras scenarijus)

Srautas sudarytas iš *DATA* kadrų	Vidutinė entropija (bitas per baitą)	Maksimali entropija (bitas per baitą)	Minimali entropija (bitas per baitą)	Duomenų ilgis (baitai)
Be paslėptų duomenų	0,884	0,968	0,881	10020
Su paslėptais duomenimis	2,013	2,043	1,998	11018
Su šifruotais paslėptais duomenimis	2,08	2,123	2,035	11106
Su suspaustais paslėptais duomenimis	2,084	2,115	2,052	6882



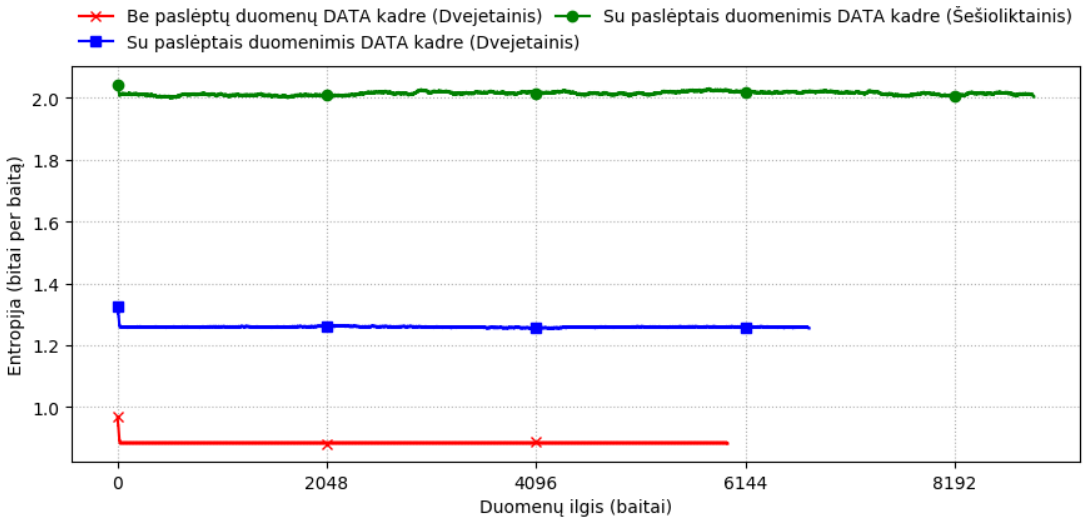
3.9 pav. Entropijos lygiai *DATA* kadre (antras scenarijus)

Entropiją skaičiuojant trečiuoju scenarijumi, iš gautų duomenų matoma, kad entropijos lygiai pakylą mažiau, kai naudojamas dvejetainis kodavimas slaptiems duomenims išsiųsti, nei kad kai naudojamas šešioliktainis (3.10 paveikslas).

Entropija, sraute kuriame slapti duomenys buvo siunčiami dvejetainiais skaičiais, apytiksliai 1,257 bito per baitą, tai apytiksliai 1,6 karto mažesnė reikšmė, nei, kad siunčiant šešioliktainiais skaičiais (3.3 lentelė).

3.3 lentelė. Entropijos lygiai *DATA* kadre (trečias scenarijus)

Srautas sudarytas iš *DATA* kadrų	Vidutinė entropija (bitas per baitą)	Maksimali entropija (bitas per baitą)	Minimali entropija (bitas per baitą)	Duomenų ilgis (baitai)
Be paslėptų duomenų	0,884	0,968	0,881	8020
Su paslėptais duomenimis (dvejetainis kodavimas)	1,258	1,323	1,253	8818
Su paslėptais duomenimis (šešioliktainis kodavimas)	2,013	2,043	1,998	11018



3.10 pav. Entropijos lygiai *DATA* kadre (trečias scenarijus)

Pirmo ir antro tyrimo scenarijų metu gauti pakankamai dideli entropijos lygių skirtumai, kurie siekia virš 1 bito per baitą, tokie skirtumai gali priversti manyti, kad sraute galėjo būti sudarytas slaptas kanalas, tačiau, tokia išvada iš karto negalėtų būti daroma jei žinoma, kad stebimame sraute, HTTP/2 duomenų prailginimo funkcija yra dažnai naudojama. Pagal RFC 7540 prailginimas yra saugumo funkcija padaryti srautą mažiau šabloniška, apsaugant jį nuo spėjimų, pavyzdžiui, kokiose internetinėse svetainėse lankosi vartotojas. Jei ši funkcija būtų naudojama, entropijos lygiai būtų labai panašūs lyginant su srautu kuriame sudarytas slaptas kanalas, todėl entropijos skaičiavimas sraute sudarytame iš *DATA* kadrų taptų neefektyvia priemone aptikti slaptus kanalus.

3.4.3. Steganografinio metodo neaptinkamumo tyrimo parametrai (*PRIORITY* kadras)

Paketai turintys *PRIORITY* kadrą surinkti siunčiant:

- 500 atsitiktinių žodžių iš žodyno, kai naudojamas kodavimo būdas – šešioliktainiai skaičiai;
- 50 atsitiktinių žodžių iš žodyno, kai naudojamas kodavimo būdas – dvejetainiai skaičiai.

Vykdant pirmąjį tyrimo scenarijų, nustatomas bloko dydis 8 baitai. Slapto kanalo kūrimas *PRIORITY* kadre pakeičia vieną reikšmę – svorį.

Vykdant antrąjį tyrimo scenarijų, nustatomas bloko dydis – 2048 baitai.

Vykdant trečiąjį tyrimo scenarijų, nustatomas bloko dydis – 2048 baitai.

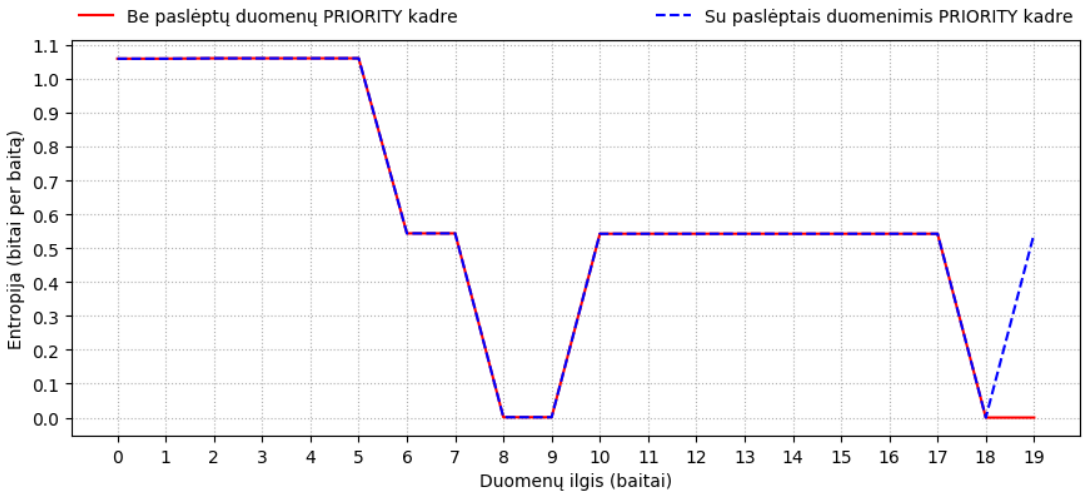
3.4.4. Steganografinio metodo neaptinkamumas (*PRIORITY* kadras)

Entropijos skaičiavimas pirmuoju scenarijumi parodė, kad ir su sudarytu slaptu kanalu, ir be jo, jei srauto identifikacija ir priklausomybė sutampa, turėtų būti gauta identiška entropija iki priešpaskutinio baito (3.11 paveikslas). Ties paskutiniu baitu, į entropijos skaičiavimo bloką patenka svoris, kuriame, *PRIORITY* kadro atveju, sudaromas slaptas kanalas, todėl entropijos reikšmė pakyla.

Svorio lauke telpa, tik vienas baitas, todėl aukštos entropijos reikšmės nematome, ji pakyla iki apytiksliai 0,542 bito per baitą. Jei svorio laukas nėra išnaudojamas, tai paskutiniame bloke entropijos reikšmė bus 0,0 (3.4 lentelė).

3.4 lentelė. Entropijos lygiai *PRIORITY* kadre (pirmas scenarijus)

Srautas sudarytas iš *PRIORITY* kadrų	Vidutinė entropija (bitas per baitą)				Maksimali entropija (bitas per baitą)				Duomenų ilgis (baitai)
	1 bloke	2 bloke	3 bloke	4 bloke	1 bloke	2 bloke	3 bloke	4 bloke	
Be paslėptų duomenų	1,059	0,001	0,542	0,0	1,061	0,544	0,544	0,0	13906
Su paslėptais duomenimis	1,059	0,001	0,542	0,542	1,061	0,544	0,544	0,544	13934



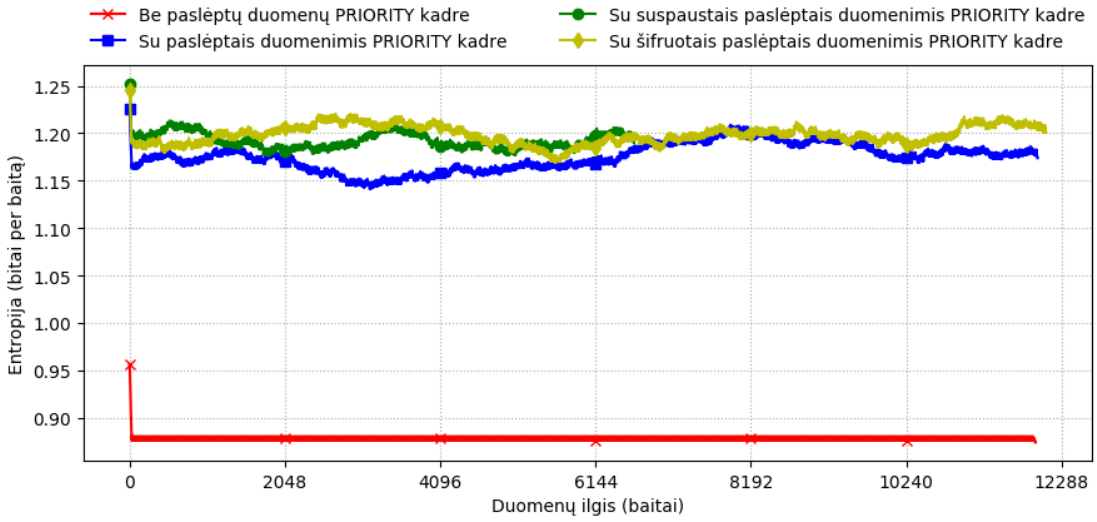
3.11 pav. Entropijos lygiai *PRIORITY* kadre (pirmas scenarijus)

Grafike (3.12 paveikslas), kuriame pavaizduoti entropijos lygiai skaičiuojant antruoju scenarijumi, matoma kaip sraute su sudarytu slaptu kanalu vis svyruoja entropijos lygiai, nes pastoviai keičiama svorio reikšmė *PRIORITY* kadre, tuo tarpu, be slapto kanalo, srauto entropija išlieka beveik viename lygyje, nes *PRIORITY* kadro svoris visada išlieka numatytas – 15.

Kaip ir tikėtasi, papildomų funkcijų naudojimas, šiek tiek, pakėlė entropijos reikšmes (3.5 lentelė). Neišnaudojus svorio lauko entropijos reikšmė apytiksliai 0,877 bito per baitą, siunčiant slaptus duomenis atviru tekstu – 1,174, užšifravus duomenis – 1,197, suspaudus duomenis – 1,191. Taip pat, pastebima, kad duomenų užšifravimo atveju regimas duomenų prailgėjimas, o duomenų suspaudimo atveju buvo išsiųsta apytiksliai 1,6 karto mažiau baitų, nei, kad atviru tekstu.

3.5 lentelė. Entropijos lygiai *PRIORITY* kadre (antras scenarijus)

Srautas sudarytas iš *PRIORITY* kadrų	Vidutinė entropija (bitas per baitą)	Maksimali entropija (bitas per baitą)	Minimali entropija (bitas per baitą)	Duomenų ilgis (baitai)
Be paslėptų duomenų	0,877	0,956	0,873	13992
Su paslėptais duomenimis	1,174	1,226	1,141	14020
Su šifruotais paslėptais duomenimis	1,197	1,245	1,169	14132
Su suspaustais paslėptais duomenimis	1,191	1,252	1,176	8700



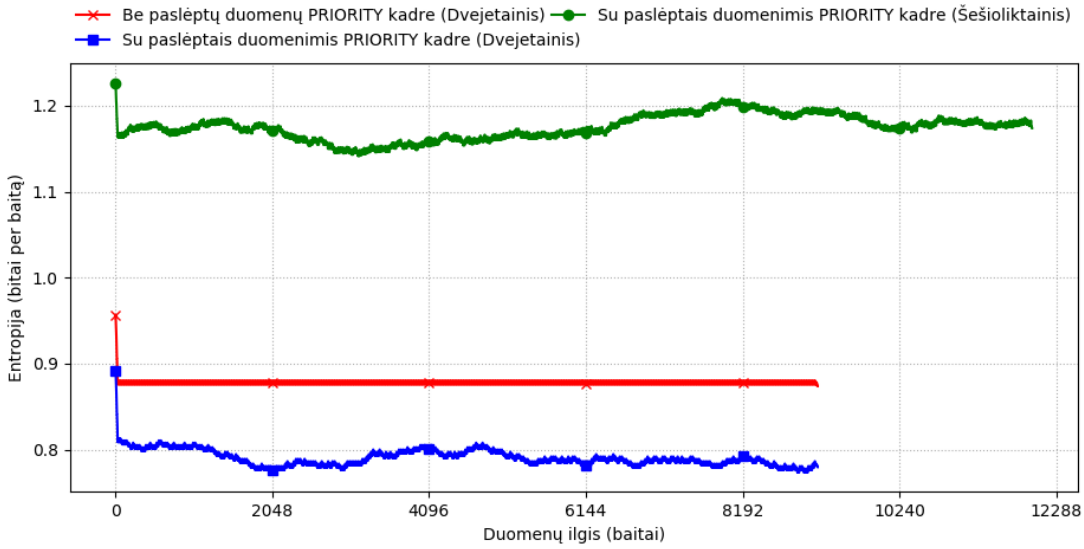
3.12 pav. Entropijos lygiai *PRIORITY* kadre (antras scenarijus)

Duomenys gauti entropiją skaičiuojant trečiuoju scenarijumi, pakeitė tendenciją – sraute, kuriame nėra siunčiamų slaptų duomenų, regimi aukštesni entropijos lygiai, nei, kad srauto kuriame jie yra siunčiami. Taip yra todėl, kad numatytoji svorio reikšmė *PRIORITY* kadre yra 15, tuo tarpu prototipu, siunčiant slaptus duomenis, kai yra loginis nulis, svorio vertė pakeičiama į nulį, o kai loginis vienetas, numatytoji reikšmė (15), laikoma vienetu. Nors ir entropijos lygis žemesnis, kai duomenys siunčiami dvejetainiais skaičiais, grafike (3.13 paveikslas) galima įžvelgti, kad srauto entropija nėra pastovi, tuo tarpu, sraute kur svorio reikšmė nekeičiama ji išsilaiko apytiksliai tame pačiame lygyje.

Vidutinė entropija, siunčiant slaptus duomenis dvejetainiais skaičiais, *PRIORITY* kadru, yra apytiksliai 0,789 bito per baitą (3.6 lentelė).

3.6 lentelė. Entropijos lygiai *PRIORITY* kadre (trečias scenarijus)

Srautas sudarytas iš *PRIORITY* kadrų	Vidutinė entropija (bitas per baitą)	Maksimali entropija (bitas per baitą)	Minimali entropija (bitas per baitą)	Duomenų ilgis (baitai)
Be paslėptų duomenų	0,877	0,956	0,873	11220
Su paslėptais duomenimis (dvejetainis kodavimas)	0,789	0,892	0,773	11220
Su paslėptais duomenimis (šešioliktainis kodavimas)	1,174	1,226	1,141	14020



3.13 pav. Entropijos lygiai *PRIORITY* kadre (trečias scenarijus)

Pirmojo bandymo rezultatai parodė, kad tiriant kadrus, tik jų rėmuose galima vaizdžiai pamatyti ties kuria kadro vieta vyksta pasikeitimai, šiuo atveju, tai pastebėta ties svorio reikšme.

Antrojo bandymo rezultatai parodė, kad entropija, kai yra sudarytas slaptas kanalas *PRIORITY* kadre, pakyla apytiksliai 3,8 karto mažiau, nei, *DATA* kadro atveju.

Žinoma, jei stebimame tinkle dažnai naudojamas *PRIORITY* kadras keisti srautų prioritetus, entropija galėtų priartėti prie srauto, kuriame sudarytas slaptas kanalas entropijos, tokiu atveju, entropijos skaičiavimas, aptikti slaptą kanalą, gali tapti nebelabai efektyviu, tada, geresnė idėja tirti ar sraute *PRIORITY* kadras seka RFC 7540 nuorodas, kaip, kad stebėti ar svorio reikšmė kyla proporcingai aukščiau už praeitą nustatytą reikšmę, ko negalima pamatyti sraute su sudarytu slaptu kanalu.

3.4.5. Steganografinio metodo neaptinkamumo tyrimo parametrai (*PING* kadras)

Paketai turintys *PING* kadrą surinkti siunčiant:

- 200, iki 8 baitų, atsitiktinių žodžių iš žodyno, kai naudojamas kodavimo būdas – šešioliktainiai skaičiai;

Vykdant pirmąjį tyrimo scenarijų, nustatomas bloko dydis 8 baitai. Slapto kanalo kūrimas *PING* kadre pakeičia vieną (iki 8 baitų) reikšmę – *ping*.

Vykdant antrąjį tyrimo scenarijų, nustatomas bloko dydis – 2048 baitai.

Prototipe nebuvo įgyvendintos suspaudimo ir duomenų siuntimo dvejetainiais skaičiais funkcijos, kai naudojamas *PING* kadras, todėl suspaustų paslėptų duomenų atvejis nėra tiriamas, o trečiasis scenarijus nėra vykdomas.

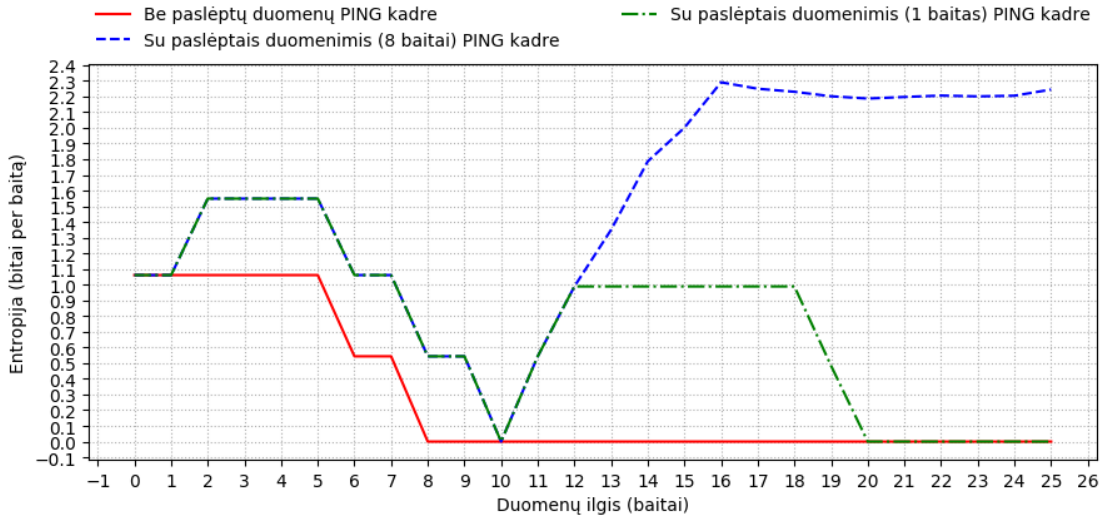
3.4.6. Steganografinio metodo neaptinkamumas (*PING* kadras)

Skaičiuojant entropiją pirmuoju scenarijumi, iš gautų duomenų, nubrėžtame grafike (3.14 paveikslas) išryškėja ir antraštės ir *ping* dalis. Nuo trečio baito matyti, kaip kuriamo slapto kanalo atveju entropija pakyla, šioje bloko dalyje yra *ACK* vėliavėlė, kurią kuriamo slapto kanalo atveju reikia priverstinai nustatyti. Toliau, ties vienuoliktu baitu, matuojama srauto identifikacijos baitų entropija, kadangi *PING* kadro srauto identifikacija visada yra 0, tai ir entropija visada bus 0,0. Nuo tolimesnių baitų, į entropijos skaičiavimo bloką įeina vis didesnė dalis *ping* lauko baitų, todėl, jei yra sukurtas slaptas kanalas, entropija kyla.

Todėl, kad *ping* laukas gali sutalpinti iki 8 baitų duomenų, matoma aukšta entropijos reikšmė, jei naudojami visi 8 baitai, entropija gali siekti bent 2,75 bitus per baitą, jei naudojamas 1 iš 8 baitų – 1.061 bitą per baitą (3.7 lentelė). Jei nėra išnaudotas *ping* laukas, paskutiniuose kadro baituose entropija visada bus lygi 0,0.

3.7 lentelė. Entropijos lygiai *PING* kadre (pirmas scenarijus)

Srautas sudarytas iš *PING* kadrų	Vidutinė entropija (bitas per baitą)				Maksimali entropija (bitas per baitą)				Duomenų ilgis (baitai)
	1 bloke	2 bloke	3 bloke	4-5 blokuose	1 bloke	2 bloke	3 bloke	4-5 blokuose	
Be paslėptų duomenų	1,061	0,0	0,0	0,0	1,061	0,0	0,0	0,0	6800
Su paslėptais duomenimis (8 baitai)	1,061	0,544	2,29	2,202	1,061	0,544	2,75	2,75	6800
Su paslėptais duomenimis (1 baitas)	1,061	0,544	0,989	0,0	1,061	0,544	1,061	0,0	6800

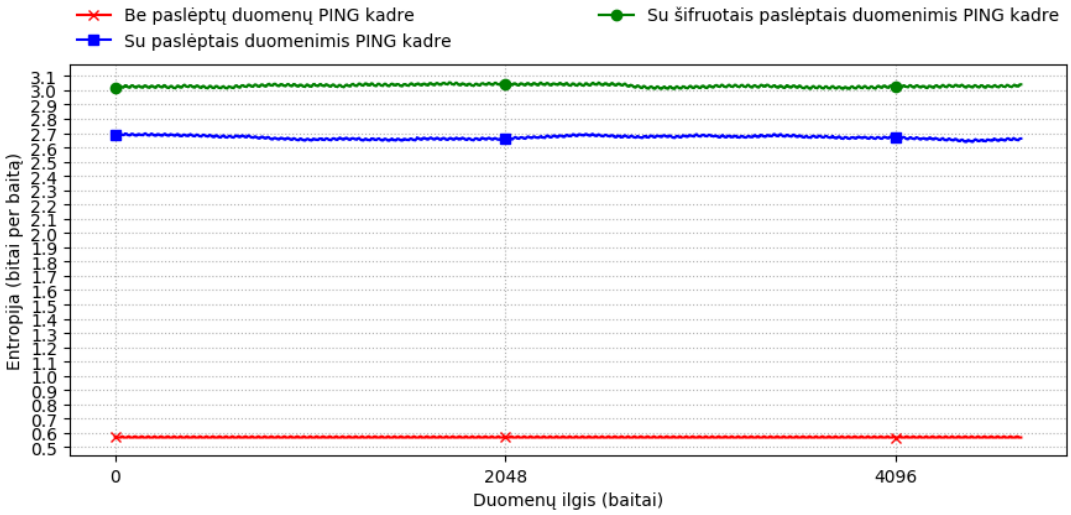


3.14 pav. Entropijos lygiai *PING* kadre (pirmas scenarijus)

Skaičiuojant entropiją antruoju scenarijumi, gauti duomenys, vėl patvirtina, kad šifravimo naudojimas padidina entropijos reikšmę (3.8 lentelė). Didelių entropijos reikšmių kitimų grafike (3.15 paveikslas) nesimato, tačiau pasimato labai aiškus entropijos skirtumas tarp srauto, kuriame siųsti slapti duomenys ir kuriame ne, kadangi neišnaudoto *ping* lauko atveju entropija labai žema – apytiksliai 0.381 bito per baitą, tuo tarpu, kai jis išnaudotas, neužšifruotų duomenų atveju, entropija – 2,669 bito per baitą, o užšifruotų – 3.03 bito per baitą (3.8 lentelė).

3.8 lentelė. Entropijos lygiai *PING* kadre (antras scenarijus)

Srautas sudarytas iš *PING* kadrų	Vidutinė entropija (bitas per baitą)	Maksimali entropija (bitas per baitą)	Minimali entropija (bitas per baitą)	Duomenų ilgis (baitai)
Be paslėptų duomenų	0,381	0,385	0,380	6800
Su paslėptais duomenimis	2,669	2,697	2,637	6800
Su šifruotais paslėptais duomenimis	3,03	3,054	3,008	6800



3.15 pav. Entropijos lygiai *PING* kadre (antras scenarijus)

Skaičiavimai ir pirmu ir antru scenarijumi parodė, kad entropijos skaičiavimas aptikti siunčiamus slaptus duomenis *PING* kadru yra efektyvi priemonė, nes paprastai *ping* lauke neįrašomi jokie duomenys, o net jei būtų nuspręstą išnaudoti *ping* lauką, tai įprastai būna statinė žinutė, kuri turėtų statinį entropijos lygį.

3.5. HTTP/2 steganografinio metodo tvirtumo tyrimas

Steganogerafinio metodo tvirtumo tyrimu siekiama išsiaiškinti ar duomenų siuntimas kuriamais slaptais kanalais yra patikimas. Tiriamas slaptų kanalų tvirtumas, kurie sudaryti *DATA*, *PRIORITY* ir *PING* kadruose. Prieš pradedant tvirtumo tyrimą, su *OpenSSL* įrankiu sudaroma atsitiktinė 100000 baitų ilgio žinutė, tada suskaičiuojama jos *md5* santrauka. Žinutė išsiunčiama prototipu nustatytu kadru, ją gavus prototipo serverio pusėje, yra skaičiuojamas gautų duomenų ilgis ir galutinės žinutės *md5* santrauka, jei žinutės ilgis ir santrauka sutampa su originalia, tada daroma išvada, kad sudarytais slaptais kanalais, gauti duomenys yra vientisi. Metodo tvirtumo eksperimentas daromas ir vietiniame ir išoriniame tinkle.

Kadangi, HTTP/2 protokolu siunčiami duomenys pernešami trečiame TCP/IP sluoksnyje esančiu patikimu TCP protokolu, tikimasi, kad ir slapti duomenys patikimai keliaus tinklu be praradimų, tačiau šio tyrimo pagrindinis tikslas patikrinti ar protokolo laukų naudojimas, ne pagal jų paskirtį, yra aptinkamas ir paveikiamas, pavyzdžiui, normalizuojamas.

3.5.1. Slapto kanalo sudaryto *DATA* kadre tvirtumas

Prototipu nustačius naudoti *DATA* kadrą slaptų duomenų siuntimui, kaip matoma 3.9 lentelėje, gautų duomenų ilgis ir žinutės santrauka sutapo su išsiųstosios žinutės. Siunčiant duomenis ir iš vidinio tinklo, ir iš išorinio duomenys išliko vientisi, nebuvo normalizuojami.

3.9 lentelė. Siunčiamų slaptų duomenų vientisumas *DATA* kadru

	Ilgis (baitai)	MD5
Išsiųsta žinutė *DATA* kadru sudarytu slaptu kanalu	100000	21af8212ab2f8407226425eaabb28caf
Gauta žinutė (iš vidinio tinklo)	100000	21af8212ab2f8407226425eaabb28caf
Gauta žinutė (iš išorinio tinklo)	100000	21af8212ab2f8407226425eaabb28caf

3.5.2. Slapto kanalo sudaryto *PRIORITY* kadre tvirtumas

Prototipu nustačius naudoti *PRIORITY* kadrą slaptų duomenų siuntimui, kaip matoma 3.10 lentelėje, siunčiant duomenis ir iš vidinio tinklo, ir iš išorinio duomenys išliko vientisi.

3.10 lentelė. Siunčiamų slaptų duomenų vientisumas *PIORITY* kadru

	Ilgis (baitai)	MD5
Išsiųsta žinutė *PRIORITY* kadru sudarytu slaptu kanalu	100000	21af8212ab2f8407226425eaabb28caf
Gauta žinutė (iš vidinio tinklo)	100000	21af8212ab2f8407226425eaabb28caf
Gauta žinutė (iš išorinio tinklo)	100000	21af8212ab2f8407226425eaabb28caf

3.5.3. Slapto kanalo sudaryto *PING* kadre tvirtumas

Kadangi kelių *PING* kadrų vienu TCP jungimu negalima išsiųsti, sudaroma ir siunčiama maksimalaus ilgio žinutė (8 baitai). Žinutė – „ABCD1234”, jos *md5* santrauka – “361633153a464830a1fe85dec5efab17”. Kaip matoma 3.11 lentelėje, ir *PING* kadro atveju, vidiniame ir išoriniame tinkluose, išsiųsta žinutė išliko vientisa.

3.11 lentelė. Siunčiamų slaptų duomenų vientisumas *PING* kadru

	Ilgis (baitai)	MD5
Išsiųsta žinutė *PING* kadru sudarytu slaptu kanalu	8	361633153a464830a1fe85dec5efab17
Gauta žinutė (iš vidinio tinklo)	8	361633153a464830a1fe85dec5efab17
Gauta žinutė (iš išorinio tinklo)	8	361633153a464830a1fe85dec5efab17

3.5.4. *DATA, PRIORITY, PING* kadruose sudarytų slaptų kanalų tvirtumo tyrimo rezultatai

Iš gautų rezultatų, galima spręsti, kad jokios ypatingos priemonės nėra naudojamos, įtartinam HTTP/2 srauto naudojimo kelio užkirtimui. Visais tirtais kadrais, išsiuntus duomenis sudarytais slaptais kanalais, jie išliko vientisi nepaisant to ar siuntimas vyko iš vidinio tinklo ar iš išorinio.

3.6. HTTP/2 steganografinio metodo tyrimo išvados

Steganografijos panaudojimas HTTP/2 kadruose ištirtas – *DATA*, *PRIORITY*, *PING* kadruose. Visi trys siunčiami kadrai atitiko RFC apibrėžtas kadro struktūras ir kaip tikėtasi, panaudojus *DATA* ir *PRIORITY* apibrėžtus laukus slaptiems duomenims persiųsti gauta 1 baito per kadrą talpa, tuo tarpu *PING* talpina iki 8 baitų per paketą.

Steganalizė – entropijos skaičiavimas, parodė, kad entropiją skaičiuojant kadro rėmuose, pasimato, kad kai kurių kadrų dalių entropija visada sutampa, todėl dažnai keičiant lauko reikšmes, į ten įstatant slaptus duomenis, tai gali pasimatyti vizualiai grafikuose. Skaičiuojant entropiją visam kadro tipo srautui, pasimato didesni entropijos lygių skirtumai. Aukščiausiai entropija pakyla

naudojant *PING*, tada *DATA* ir galiausiai *PRIORITY* kadrą, todėl, šios steganalizės būdų, labiausiai aptinkamas slaptų duomenų persiuntimas *PING* kadruose.

Papildomų funkcijų – šifravimo ir suspaudimo naudojimas, entropiją, bent šiek tiek pakelia, nes padaro duomenis labiau įvairius ir mažiau nuspėjamus, todėl padidinama aptinkamumo tikimybė, tačiau slaptus duomenis aptikusiam asmeniui, duomenų atkūrimas gali būti sunkesnis.

Siunčiant slaptus duomenis dvejetainiu kodavimu, entropijos lygiai žemesni, nei, kad šešioliktainiu, tačiau dėl mažos talpos, pilnos žinutės persiuntimas sudarys bent 8 kartus daugiau persiunčiamų paketų.

Ištyrus steganografinio metodo tvirtumą vietiniame ir išoriniame tinkle nustatyta, kad slapti duomenys nėra normalizuojami ir patikimai keliauja tinklu.

4. IŠVADOS

Iš analizės paaiškėjus, kad HTTP/2 steganografinis metodas, iki šiol, dar nebuvo praktiškai realizuotas, šio darbo metu jis buvo realizuotas. Modeliuojant steganografinio metodo prototipą, pavyko panaudoti tris HTTP/2 kadrus slaptiems duomenims persiųsti – *DATA*, *PRIORITY* ir *PING*. Į šių kadrų, ne įprastiems duomenims, bet kitiems tikslams skirtus laukus, pavyko išskaidyti, sudėti ir persiųsti slaptus duomenis. Kaip iš tyrimo paaiškėjo, slaptų duomenų siuntimas, be duomenų praradimų, sėkmingai keliauja ir vietiniame ir išoriniame tinkle.

Entropijos skaičiavimas kaip steganalizės būdas aptikti šiuos slaptus kanalus gali būti efektyvi priemonė, bet tik tam tikromis sąlygomis:

- Žinoma, kad tinkle HTTP/2 *DATA* kadrų praplėtimo ir *PRIORITY* kadro svorio funkcijos, arba nėra naudojamos, arba jų naudojimas menkas;
- Žinoma, kad *PING* kadras arba nenaudojamas įvairiems, skirtingiems pranešimams siųsti, arba siunčiamas numatytai tuščias.

Jei minėtos sąlygos netenkinamos, reikėtų apmastyti kitokius aptikimo tyrimo metodus. Pavyzdžiui, *PRIORITY* kadro slaptų duomenų siuntimui aptikti, pasiūlymas – stebėti kaip keičiasi kadrų prioritetai, pagal RFC 7540, jie turi keistis proporcionaliai, o siunčiant slaptus duomenis svorio keitimasis kur kas chaotiškesnis, taip pat, numatyta svorio reikšmė yra 15, todėl maža tikimybė, kad nustatomos reikšmės būtų žemesnės. *PING* kadrus, tiesiog turėtų galėti normalizuoti, nebent yra svari priežastis, kodėl *ping* lauko dalių žinutės yra svarbios. Su *DATA* kadrais yra sunkiau – nėra nustatyta, kaip praplėtimas turėtų būti išnaudojamas, jis gali būti ir atsitiktinis ir nustatyto dydžio, todėl tokio tipo slaptus kanalus gali būti gan sunku pašalinti.

Ateičiai, galėtų būti patobulinamas siuntimas – naudoti kelias rūšis kadrų vienu metu, taip padidinant steganografinę talpą. Taip pat, be šiame darbe steganografijai išbandytų HTTP/2 kadrų, reikėtų ištirti kitų kadrų, pavyzdžiui, *SETTINGS* steganografines galimybes. Kadangi, tokie, netradiciniai duomenų persiuntimai dažniausiai nepageidautini, reikėtų patobulinti steganalizės metodą, siekiant aptikti ir sustabdyti slaptų duomenų siuntimą HTTP/2 protokolu.

5. LITERATŪRA

[1] K. Muhammad, J. Ahmad, H. Farman ir M. Zubair, „A Novel Image Steganographic Approach for Hiding Text in Color Images using HSI Color Model," Islamia College Peshawar, Pakistan, 2015.

[2] T. G. Handel ir M. T. S. II, „Hiding Data in the OSI Network Model," Los Alamos National Laboratory, Los Alamos, 1996.

[3] W. Fraczek ir K. Szczypiorski, „StegBlocks: ensuring perfect undetectability of network steganography," Warsaw University of Technology , Warsaw, 2015.

[4] J. Fridrich, „APPLICATIONS OF DATA HIDING IN DIGITAL IMAGES," SUNY Binghamton, Binghamton, 1998.

[5] J. Lubacz, W. Mazurczyk ir K. Szczypiorski, „Principles and Overview of Network Steganography," IEEE Communications Magazine, Warsaw, 2014.

[6] R.-I. Ciobanu, M.-O. Tirsa, R. Lupu, S. Stan ir M. I. Andreica, „SCONeP: Steganography and Cryptography Approach for UDP and ICMP," Politehnica University of Bucharest, Bucharest, 2011.

[7] W. Frączek, W. Mazurczyk ir K. Szczypiorski, „How Hidden Can Be Even More Hidden?," Warsaw University of Technology , Warsaw, 2011.

[8] C. H. Rowland, „Covert Channels in the TCP/IP Protocol Suite," 1997.

[9] J. Rutkowska, „Rutkowska, J., The Implementation of Passive Covert Channels in the Linux Kernel, Chaos Communication," 2004.

[10] S. J. Murdoch ir S. Lewis, „Embedding covert channels into TCP/IP," Barcelona, Spain, 2005.

[11] A. Mileva ir B. Panajotov, „Covert Channels in TCP/IP Protocol Stack -extended version-," University Goce Delcev, Štip, 2000.

[12] I. S. I. U. o. S. California, „tools.ietf.org," Rugsėjis 1981. [Tinkle]. Available: https://tools.ietf.org/html/rfc791. [Kreiptasi 2019].

[13] K. Ahsan ir D. Kundur, „Practical Data Hiding in TCP/IP," University of Toronto, Toronto, 2002.

[14] H. A. Alsaffar ir D. Johnson, „Covert Channel using the IP Timestamp Option of an IPv4 Packet," Rochester Institute of Technology, New York , 2015.

[15] daemon9, „Project Loki," *Phrack Magazine,* t. 6, nr. 49, 1996.

[16] M. J. Muench. [Tinkle]. Available: http://icmpchat.sourceforge.net. [Kreiptasi 2019].

[17] R. Murphy, „IPv6 / ICMPv6 Covert Channels," 2006.

[18] C. Ma, Y. Fan ir L. Ji, „Covert channel for local area network," IEEE International Conference on Wireless Communications, Networking and Information Security, 2010.

[19] B. Dimitrova ir A. Mileva, „Steganography of Hypertext Transfer Protocol Version 2 (HTTP/2)," University Goce Delčev, Štip, 2017.

[20] K. Szczypiorski, „HICCUPS: Hidden communication system for corrupted networks," Warsaw University of Technology, Warsaw, 2003.

[21] G. Nagy ir T. SZERB, 22 Rugsėjo 2002. [Tinkle]. Available: http://savannah.nongnu.org/projects/nstx. [Kreiptasi 2019].

[22] „code.kryo.se," 16 Birželio 2014. [Tinkle]. Available: https://code.kryo.se/iodine/. [Kreiptasi 2019].

[23] W. Mazurczyk, M. Smolarczyk ir K. Szczypiorski, „Retransmission steganography and its detection," Springer-Verlag, Warsaw, 2009.

[24] M. Belshe, BitGo, R. Peon, I. Google, E. M. Thomson ir Mozilla, „tools.ietf.org,“ Gegužė 2015. [Tinkle]. Available: https://tools.ietf.org/html/rfc7540. [Kreiptasi 2019].

[25] E. Melendez, „Steganography Detection Using Entropy MeasuresTechnical,“ Puerto Rico, 2012.

[26] J. Gin, R. Greenstadt, P. Litwack ir R. Tibbetts, „Covert Messaging Through TCP Timestamps,“ Massachusetts, 2003.

[27] P. Clifford ir I. A. Cosma, „A simple sketching algorithm for entropy estimation over streaming data,“ 2013.

[28] J. P. Black, „TECHNIQUES OF NETWORK STEGANOGRAPHY AND COVERT CHANNELS,“ San Diego, 2013.

[29] A. Aviv, G. Shah ir M. Blaze, „Steganographic Timing Channels,“ Pennsylvania, 2011.

[30] G. Berg, I. Davidson, M.-Y. Duan ir G. Paul, „Searching For Hidden Messages: Automatic Detection of Steganography,“ Albany, 2003.

[31] AnjanK, SrinathNK ir J. Abraham, „ENTROPY BASED DETECTION ANDBEHAVIORAL ANALYSIS OF HYBRID COVERT CHANNELIN SECURED COMMUNICATION,“ Pune.

[32] S. Gianvecchio ir H. Wang, „Detecting Covert Timing Channels: An Entropy-Based Approach,“ Williamsburg, 2007.

[33] Z. Wang, L. Huang, W. Yang ir Z. He, „An Entropy-based Method for Detection of Covert Channels over LTE,“ Nanjing, 2018.

[34] „rosettacode,“ [Tinkle]. Available: https://rosettacode.org/wiki/Entropy. [Kreiptasi 2019].

[35] W. Mazurczyk, S. Wendzel, S. Zander, A. Houmansadr ir K. Szczypiorski, Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures, Wiley-IEEE Press; 1 edition, 2016.

6. PRIEDAI

6.1. priedas. HTTP/2 steganografinio metodo prototipo programinis kodas

Paleidimo programa (http2steg.py)

```
import argparse
import hashlib
from steg_client import Client
from steg_server import Server

# Argument parser
parser = argparse.ArgumentParser(description="HTTP/2 Steganography Client/Server")
group = parser.add_mutually_exclusive_group()
group.add_argument("-c", "--client", action="store_true", help="run as steganography cli
ent")
group.add_argument("-s", "--server", action="store_true", help="run as steganography ser
ver")
parser.add_argument("-a", "--address", help="source or destination depending on whether
client or server mode is used")
parser.add_argument("-m", "--message", help="secret message")
parser.add_argument("-p", "--port", type=int, help="server port, default = 80")
parser.add_argument("-z", "--compress", action="store_true", help="compress parts of sec
ret message")
parser.add_argument("-M", "--header-method", choices=["GET", "POST"], help="set method,
e.g. GET, POST")
parser.add_argument("-A", "--header-authority", help="set authority", metavar=('AUTH'))
parser.add_argument("-S", "--header-scheme", choices=["http", "https"], help="set connec
tion scheme, e.g. http, https")
parser.add_argument("-P", "--header-path", help="set path that method will connect to, e
.g. /", metavar=('PATH'))
parser.add_argument("-U", "--header-user-agent", help="spoof user agent", metavar=('USER
-AGENT'))
parser.add_argument("-d", "--data-frame", action="store_true", help="use padding of data
 frame as a covert channel")
parser.add_argument("-r", "--priority-frame", action="store_true", help="use priority fr
ame as a covert channel")
parser.add_argument("-o", "--ping-frame", action="store_true", help="use ping frame as a
 covert channel")
# parser.add_argument("-y", "--promise-frame", action="store_true", help="use promise fr
ame as a covert channel")
parser.add_argument("-x", "--hex", action="store_true", help="send hex arrays instead of
 bit arrays (faster throughput)")
parser.add_argument("-k", "--enc-key", help="set DES symmetric encryption key", metavar=
('KEY'))
parser.add_argument("-f", "--fake-data", help="set fake data that will be sent through n
on covert channels, default = '1'", metavar=('DATA'))
parser.add_argument("-t", "--delay", help="set delay time between sent stego bits in sec
onds, float is supported", metavar=('TIME'))
args = parser.parse_args()

## STEGANOGRAPHY CLIENT
if args.client:
    # Print hash sum of steg message
    print (hashlib.md5(args.message).hexdigest())

    # Initialize steg HTTP/2 client
    sc = Client(args.address)

    # Compress steg message
    if args.compress:
        # Encrypt steg message
```

```
        if args.enc_key:
            # Compress then encrypt
            # https://superuser.com/questions/257782/compress-and-then-encrypt-or-vice-v
ersa
            compressed_msg = sc.compress_steg_messagge(args.message)
            enc_msg = sc.encrypt_secret_message(sc.new_des(args.enc_key), sc.pad_secret_
message(compressed_msg))
            # Split steg message into chunks that don't go over default 65535 TCP window
 limit
            steg_array = sc.get_steg_parts(enc_msg, args.hex)
        else:
            compressed_msg = sc.compress_steg_messagge(args.message)
            steg_array = sc.get_steg_parts(compressed_msg, args.hex)
    else:
        if args.enc_key:
            enc_msg = sc.encrypt_secret_message(sc.new_des(args.enc_key), sc.pad_secret_
message(args.message))
            steg_array = sc.get_steg_parts(enc_msg, args.hex)
        else:
            steg_array = sc.get_steg_parts(args.message, args.hex)

    # Send each steg part (less than or equal to 65535 bytes in length) with new instanc
e of TCP socket and HTTP/2 connection
    for steg_part in steg_array:
        socket = sc.new_socket(args.port)
        conn = sc.new_http2_connection()
        header = sc.new_http2_header(args.header_method, args.header_authority, args.hea
der_scheme, args.header_path, args.header_user_agent)
        flow_id = sc.set_http2_header(conn, header)

        # DATA and PRIORITY frames
        if args.data_frame and args.priority_frame:
            # EXPERIMENTAL
            # sc.send_all(args.hex, steg_part, socket, conn, flow_id, args.fake_data, ar
gs.delay)
            # sc.end_http2_stream(socket, conn, flow_id)
            pass
        # DATA frame
        elif args.data_frame:
            sc.send_data_steg(args.hex, steg_part, socket, conn, flow_id, args.fake_data
, args.delay)
            sc.end_http2_stream(socket, conn, flow_id)
        # PRIORITY frame
        elif args.priority_frame:
            sc.send_priority_steg(args.hex, steg_part, socket, conn, flow_id, args.delay
)
            sc.end_http2_stream(socket, conn, flow_id)
        # PING frame
        elif args.ping_frame:
            if args.enc_key:
                sc.send_ping_steg(enc_msg, socket, conn)
            else:
                sc.send_ping_steg(args.message, socket, conn)
        # PROMISE frame. Sending PROMISE frames is for SERVER side only. This will not w
ork.
        # elif args.promise_frame:
            # sc.send_promise(args.message, socket, conn)
            # sc.send_promise(steg_part, socket, conn, flow_id)

## STEGANOGRAPHY SERVER
elif args.server:
    # Initialize steg HTTP/2 server
```

```
    ss = Server(args.enc_key)
    socket = ss.new_socket(args.port)
    while True:
        ss.handle_connections(socket.accept()[0], args.hex, args.compress)

## SELECT PROGRAM MODE HELP MESSAGE
else:
    print ('Use either -c or -s flag, to start program as a client or a server')
```

Klientas (steg_client.py)

```
import socket
import h2.connection
import h2.events
import hyperframe.frame
import zlib
import base64
from Crypto.Cipher import DES
import time

class Client:

## TCP SOCKET AND HTTP/2 CONNECTIONS

    # Create new TCP socket connection
    def new_socket(self, port):
        sock = socket.socket()
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.connect((self.dest, port if port != None else 80))
        return sock

    # Create new HTTP/2 connection
    def new_http2_connection(self):
        # Configure HTTP/2 client to validate and normalize headers
        conf = h2.config.H2Configuration(client_side = True, validate_outbound_headers =
 True, normalize_outbound_headers = True)
        conn = h2.connection.H2Connection(config = conf)
        conn.initiate_connection()
        return conn

    # HTTP/2 header constructor
    def new_http2_header(self, method, authority, scheme, path, user_agent):
        if user_agent == None:
            return [
                (':method', method if method != None else 'GET'),
                (':authority', authority if authority != None else 'localhost'),
                (':scheme', scheme if scheme != None else 'https'),
                (':path', path if path != None else '/')
            ]
        else:
            return [
                (':method', method if method != None else 'GET'),
                (':authority', authority if authority != None else 'localhost'),
                (':scheme', scheme if scheme != None else 'https'),
                (':path', path if path != None else '/'),
                ('user-agent', str(user_agent))
            ]

    # Set available flow ID and set HTTP/2 header
    def set_http2_header(self, conn, header):
        flow_id = conn.get_next_available_stream_id()
        conn.send_headers(flow_id, header)
        return flow_id
```

```
    # End HTTP/2 stream of set flow ID
    def end_http2_stream(self, socket, conn, flow_id):
        conn.end_stream(flow_id)
        data_to_send = conn.data_to_send()
        socket.sendall(data_to_send)

## HTTP/2 STEGANOGRAPHY

    # Return array of split [compressed | uncompressed / binary encoded | hex encoded ]
message
    def get_steg_parts(self, message, encode_hex):
        steg_array = []
        for steg_part in self.split_steg(message):
            if encode_hex:
                steg_array.append(self.steg_to_dec(steg_part))
            else:
                steg_array.append(self.steg_to_bin(steg_part))
        return steg_array

    # Return steg message chunk
    def split_steg(self, steg_msg):
        # print(sum(bytearray(steg_msg))+(2*len(steg_msg)))
        steg_msg_arr = []
        self.split_message(steg_msg, steg_msg_arr)
        return [steg_msg_arr[i] for i, msg in enumerate(steg_msg_arr)]

    # Split steg message into parts so that it won't go over default 65535 TCP window li
mit
    def split_message(self, steg_msg, steg_msg_arr):
        frame_size = 0
        steg_part = ''
        last_steg_part = len(steg_msg) - 1
        for i, char in enumerate(steg_msg):
            if frame_size != 65535:
                frame_size += ord(char)+2
                steg_part += char
                if i == last_steg_part:
                    steg_msg_arr.append(steg_part)
                if frame_size >= 65535:
                    if frame_size != 65535:
                        steg_part = steg_part[:-1]
                    steg_msg_arr.append(steg_part)
                    return self.split_message(steg_msg.replace(steg_part, ''), steg_msg_
arr)

    # Send steg message in [binary | hex ] encoding through DATA frame
    def send_data_steg(self, encode_hex, steg_part, socket, conn, flow_id, fake_data, de
lay):
        # Send HEX array
        if encode_hex:
            for steg_char in steg_part:
                conn.send_data(flow_id, fake_data if fake_data != None else '1', False,
int(steg_char))
                # No STEG
                # conn.send_data(flow_id, fake_data if fake_data != None else '1', False
)
                data_to_send = conn.data_to_send()
                socket.sendall(data_to_send)
                if delay != None:
                    time.sleep(float(delay))
        # Send BITS array
```

```
        else:
            for steg_char_item in steg_part:
                for char in steg_char_item:
                    conn.send_data(flow_id, fake_data if fake_data != None else '1', Fal
se, int(char))
                    # No STEG
                    # conn.send_data(flow_id, fake_data if fake_data != None else '1', F
alse)
                    data_to_send = conn.data_to_send()
                    socket.sendall(data_to_send)
                    if delay != None:
                        time.sleep(float(delay))

    # Send steg message in [binary | hex ] encoding through PRIORITY frame
    def send_priority_steg(self, encode_hex, steg_part, socket, conn, flow_id, delay):
        if encode_hex:
            for steg_char in steg_part:
                # Weight must be between 1 and 256
                if steg_char == '0':
                    steg_char = 256
                conn.prioritize(flow_id, int(steg_char))
                # No STEG
                # conn.prioritize(flow_id)
                data_to_send = conn.data_to_send()
                socket.sendall(data_to_send)
                if delay != None:
                    time.sleep(float(delay))
            # EXPERIMENTAL
            # Use both HEADER and PRIORITY frame weights to send steg message
            # iter_steg_part = iter(steg_part)
            # for steg_char in iter_steg_part:
            #     if steg_char == '0':
            #         steg_char = 256
            #     next_id = conn.get_next_available_stream_id()
            #     conn.send_headers(next_id, self.new_http2_header('GET', 'localhost', '
https', '/', ''), priority_weight = int(steg_char))
            #     try: conn.prioritize(next_id, int(next(iter_steg_part)))
            #     except:
            #         pass
            #     data_to_send = conn.data_to_send()
            #     socket.sendall(data_to_send)
        else:
            for steg_char in steg_part:
                for char in steg_char:
                    if int(char) == 1:
                        conn.prioritize(flow_id, int(char))
                        # No STEG
                        # conn.prioritize(flow_id, None)
                    else:
                        conn.prioritize(flow_id, None)
                    data_to_send = conn.data_to_send()
                    socket.sendall(data_to_send)
                    if delay != None:
                        time.sleep(float(delay))

    # Send steg message in hex encoding through PING frame
    def send_ping_steg(self, steg_message, socket, conn):
        steg_message_arr = []
        if len(steg_message) != 8:
            steg_message_arr = [steg_message[i:i+8] for i in range(0, len(steg_message),
 8)]
        else:
```

```
            steg_message_arr.append(steg_message)
        pf = hyperframe.frame.PingFrame(stream_id = 0, opaque_data = steg_message_arr[0]
)
        # No STEG
        # pf = hyperframe.frame.PingFrame(stream_id = 0, opaque_data = '')
        pf.flags.add('ACK')
        data_to_send = conn._data_to_send
        data_to_send += pf.serialize()
        socket.sendall(data_to_send)

    # EXPERIMENTAL
    # Send steg using multiple frames
    # def send_all(self, encode_hex, steg_part, socket, conn, flow_id, fake_data, delay)
:
    #     if encode_hex:
    #         iter_steg_part = iter(steg_part)
    #         for steg_char in iter_steg_part:
    #             # Weight must be between 1 and 256
    #             if steg_char == '0':
    #                 steg_char = 256
    #             next_id = conn.get_next_available_stream_id()
    #             conn.send_headers(stream_id = next_id, headers = self.new_http2_header
('GET', 'localhost', 'https', '/', ''), priority_weight = int(steg_char))
    #             try: conn.prioritize(stream_id = next_id, weight = int(next(iter_steg_
part)))
    #             except:
    #                 pass
    #             try: conn.send_data(stream_id = next_id, data = fake_data if fake_data
 != None else '1', end_stream = False, pad_length = int(next(iter_steg_part)))
    #             except:
    #                 pass
    #             socket.sendall(conn.data_to_send())
    #             if delay != None:
    #                 time.sleep(float(delay))

    # Sending PROMISE frames is for server side only.
    # def send_promise(self, steg_message, socket, conn):
    #     pf = hyperframe.frame.PushPromiseFrame(stream_id=1, promised_stream_id=1)
    #     pf.flags.add('PADDED')
    #     hyperframe.frame.Padding.parse_padding_data(pf, "A")
    #     data_to_send = conn._data_to_send
    #     data_to_send += pf.serialize()
    #     socket.sendall(data_to_send)

## ENCODING

    # Return binary encoded message
    def steg_to_bin(self, steg_part):
        return [format(ord(c), '08b') for c in steg_part]

    # Return decimal encoded message
    def steg_to_dec(self, steg_part):
        return [format(ord(c)) for c in steg_part]

## COMPRESSION

    def compress_steg_messagge(self, message):
        return zlib.compress(message, 9)

## ENCRYPTION

    def pad_secret_message(self, message):
```

```
        while len(message) % 8 != 0:
             message += ' '
        return message

    def new_des(self, key):
        return DES.new(key, DES.MODE_ECB)

    def encrypt_secret_message(self, des, padded_message):
        return des.encrypt(padded_message)

## INIT

    def __init__(self, dest):
        self.dest = dest
```

Serveris (steg_server.py)

```
import socket
import h2.connection
import h2.events
import zlib
import base64
from Crypto.Cipher import DES

class Server:

    bit_array = []
    hex_array = []
    compressed_steg = ''
    encrypted_steg = ''
    frame_type = ''

## SOCKET AND HTTP/2 CONNECTIONS

    # Bind TCP socket
    def new_socket(self, port):
        sock = socket.socket()
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.bind(('0.0.0.0', port if port != None else 80))
        sock.listen(5)
        return sock

    # Handle incoming connections
    def handle_connections(self, sock, encode_hex, compress):
        # Configure as HTTP/2 server, validate headers
        conf = h2.config.H2Configuration(client_side=False, validate_inbound_headers=Tru
e)
        conn = h2.connection.H2Connection(config=conf)

        while True:
            data = sock.recv(65535)
            if not data:
                break

            events = conn.receive_data(data)
            for event in events:
                if isinstance(event, h2.events.RequestReceived):
                    pass
                if isinstance(event, h2.events.RemoteSettingsChanged):
                    # self.clear_arrays()
                    pass
                if isinstance(event, h2.events.DataReceived):
                    if event.flow_controlled_length != 0:
```

```
                        frame_type = 'Data'
                    if encode_hex:
                        self.construct_hex_array(event.flow_controlled_length)
                    else:
                        self.construct_bit_array(event.flow_controlled_length)
                elif isinstance(event, h2.events.StreamEnded):
                    if frame_type == 'Data':
                        if encode_hex:
                            self.decode_hex_steg(self.hex_array, compress if compress !=
 None else False)
                        else:
                            self.decode_bin_steg(self.bit_array, compress if compress !=
 None else False)
                    elif frame_type == 'Priority':
                        if encode_hex:
                            self.decode_hex_steg(self.hex_array, compress if compress !=
 None else False)
                        else:
                            self.decode_bin_steg(self.bit_array, compress if compress !=
 None else False)
                    self.clear_arrays()
                elif isinstance(event, h2.events.PriorityUpdated):
                    frame_type = 'Priority'
                    if encode_hex:
                        self.construct_hex_array_priority(event.weight)
                    else:
                        self.construct_bit_array(event.weight)
                elif isinstance(event, h2.events.PingAcknowledged):
                    if self.key != None:
                        print (self.decrypt_secret_message(self.new_des(self.key), event
.ping_data))
                    else:
                        print (event.ping_data)

## HTTP/2 STEGANOGRAPHY

    # Puts incoming bit stream into array
    def construct_bit_array(self, flow_length):
        if flow_length == 1:
            self.bit_array.append(0)
        elif flow_length != 0:
            self.bit_array.append(1)

    # Puts incoming hex stream into array
    def construct_hex_array(self, flow_length):
        if (flow_length > 1):
            self.hex_array.append("{:02X}".format(flow_length - 2))
        elif (flow_length == 1):
            self.hex_array.append("{:02X}".format(flow_length - 1))

    def construct_hex_array_priority(self, flow_length):
        self.hex_array.append("{:02X}".format(flow_length))

    # Clears bit and hex arrays
    def clear_arrays(self):
        del self.bit_array[:]
        del self.hex_array[:]

    # Decodes bin encoded secret message
    def decode_bin_steg(self, bit_arr, compression):
        try:
            stegram=''
```

```
            for b in bit_arr:
                if b == 1:
                    stegram += "1"
                elif b == 0:
                    stegram += "0"
            if self.key != None:
                print (self.decrypt_secret_message(self.new_des(self.key), zlib.decompre
ss(('%x' % int(stegram, 2)).decode('hex'))))
            else:
                print (zlib.decompress(('%x' % int(stegram, 2)).decode('hex')))
        except:
            # Stegram does not appear to be compressed...
            try:
                stegram=''
                for b in bit_arr:
                    if b == 1:
                        stegram += "1"
                    elif b == 0:
                        stegram += "0"
                if self.key != None:
                    print (self.decrypt_secret_message(self.new_des(self.key), ('%x' % i
nt(stegram, 2)).decode('hex')))
                else:
                    print (('%x' % int(stegram, 2)).decode('hex'))
            except:
                print ("Error decoding stegram")

    # Decodes hex encoded secret message
    def decode_hex_steg(self, hex_arr, compression):
        stegram=''
        for hx in hex_arr:
            # Fix for PRIORITY frame
            if hx == '100':
                hx = '00'
            char=chr(int(hx, 16))
            stegram+=char
        if self.key != None:
            if compression:
                self.encrypted_steg += stegram
                try:
                    print (zlib.decompress(self.decrypt_secret_message(self.new_des(self
.key), self.encrypted_steg)))
                    self.encrypted_steg = ''
                except:
                    pass
            else:
                self.encrypted_steg += stegram
                try:
                    self.decrypt_secret_message(self.new_des(self.key), self.encrypted_s
teg)
                except:
                    pass
                else:
                    print (self.decrypt_secret_message(self.new_des(self.key), self.encr
ypted_steg))
                    self.encrypted_steg = ''
        else:
            if compression:
                try:
                    print (zlib.decompress(stegram).decode('utf-8'))
                except zlib.error as e:
                    if (e.args[0] == 'Error -5 while decompressing data: incomplete or t
```

```
runcated stream'):
                        if (len(self.compressed_steg) != 0):
                            self.compressed_steg = stegram
                    try:
                        print (zlib.decompress(self.compressed_steg+stegram).decode('utf
-8'))
                        self.compressed_steg = ''
                    except zlib.error as e:
                        if (e.args[0] == 'Error -5 while decompressing data: incomplete
or truncated stream'):
                            self.compressed_steg += stegram
            else:
                print (stegram)

## ENCRYPTION

    def new_des(self, key):
        return DES.new(key, DES.MODE_ECB)

    def decrypt_secret_message(self, des, encrypted_message):
        return des.decrypt(encrypted_message)

# INIT

    def __init__(self, key):
        self.key = key
```

6.2. priedas. Scenarijus HTTP/2 paketų entropijai tirti

http2entropy.py

```
#!/usr/bin/python
# -*- coding: utf=8 -*-
import sys
import argparse
import math
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np
from scapy.all import *

parser = argparse.ArgumentParser(description="HTTP/2 Frame Entropy Calculator")
parser.add_argument("-p", "--packet-file", nargs='+', help="packet file or multiple file
s")
parser.add_argument("-b", "--block-size", type=int, help="size of chunks")
parser.add_argument("-l", "--label", nargs='+', help="line label or multiple labels")
parser.add_argument("-f", "--frame", help="frame name used in title")
parser.add_argument("-d", "--data-frame", action="store_true", help="calculate entropy o
f data frame flow")
parser.add_argument("-r", "--priority-frame", action="store_true", help="calculate entro
py of data frame flow")
parser.add_argument("-o", "--ping-frame", action="store_true", help="calculate entropy o
f data frame flow")
args = parser.parse_args()

LINE = ['r-', 'b--', 'g-.', 'y-', 'o-']

# http://blog.dkbza.org/2007/05/scanning-data-for-entropy-anomalies.html
def H(data):
  if not data:
    return 0
  entropy = 0
  for x in range(256):
    p_x = float(data.count(chr(x)))/len(data)
    if p_x > 0:
      entropy += - p_x*math.log(p_x, 2)
  return entropy

def entropy_scan(data, block_size) :
  for block in (
    data[x:block_size+x]
    for x in range (len (data) - block_size) ):
      yield H (block)

def packet_entropy(filepath, block_size):
  packets = rdpcap(os.path.join(filepath))
  ea = []
  ld = 0
  for px, packet in enumerate(packets):
    if packet.getlayer(Raw):
        http = packet[Raw].load
        if (px > 4):
          data = ''.join(x.encode('hex') for x in http)
          # data = ''.join(x.encode('hex') for x in http[-17:])
          ld += len(data)
          npa = np.asarray(list(entropy_scan(data, block_size)))
          ea.append(npa)
  pea = np.zeros([len(ea), len(max(ea, key = lambda x: len(x)))])
```

```
  for i, j in enumerate(ea):
    pea[i][0:len(j)] = j
  print (ld)
  return np.mean(pea, axis=0), np.maximum.reduce(pea)

def packet_entropy_data(filepath, block_size):
  packets = rdpcap(os.path.join(filepath))
  data = ''
  for packet in packets:
    if packet.getlayer(Raw):
        http = packet[Raw].load
        data += ''.join(x.encode('hex') for x in http[:11])
  print (len(data))
  npa = list((entropy_scan(data, block_size)))
  return npa

def packet_entropy_priority(filepath, block_size):
  packets = rdpcap(os.path.join(filepath))
  data = ''
  for packet in packets:
    if packet.getlayer(Raw):
        http = packet[Raw].load
        data += ''.join(x.encode('hex') for x in http[:15])
  print (len(data))
  npa = list((entropy_scan(data, block_size)))
  return npa

def packet_entropy_ping(filepath, block_size):
  packets = rdpcap(os.path.join(filepath))
  data = ''
  for packet in packets:
    if packet.getlayer(Raw):
        http = packet[Raw].load
        data += ''.join(x.encode('hex') for x in http[-17:])
  print (len(data))
  npa = list((entropy_scan(data, block_size)))
  return npa

def block_coordinates(packet_entropy_array, block_size):
  pea = packet_entropy_array
  coordinates = []
  x = 0
  for d in pea[::block_size]:
    coordinates.append([x, d])
    x += args.block_size
  coordinates.append([len(pea)-1, pea[len(pea)-1]])
  return coordinates

def plot_whole_flow_entropy():
  print (np.mean(ped))
  print (np.amax(ped))
  print (np.amin(ped))
  print ('')
  if args.label != None:
    plt.plot(ped, LINE[i], label=args.label[i].decode("utf-8"))
  else:
    plt.plot(ped, LINE[i])

def configure_plot():
  # Legend and labels
  leg = plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3, ncol=2, mode="expand", bo
rderaxespad=0., framealpha=1, fancybox=True)
```

```
  leg.get_frame().set_linewidth(0)
  plt.grid(linestyle='dotted')
  plt.ylabel(u'Entropija (bitai per baitą)')
  plt.xlabel(u'Duomenų ilgis (baitai)')

if __name__ == "__main__":
  ax = plt.axes()
  ax.xaxis.set_major_locator(ticker.MultipleLocator(args.block_size if args.block_size !
= None else 32))
  # ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
  # ax.yaxis.set_major_locator(ticker.MultipleLocator(0.1))
  if (len(args.packet_file) <= len(LINE)):
    if (args.data_frame):
      for i, packetfile in enumerate(args.packet_file):
        ped = packet_entropy_data(packetfile, args.block_size if args.block_size != None
 else 32)
        plot_whole_flow_entropy()
    elif (args.priority_frame):
      for i, packetfile in enumerate(args.packet_file):
        ped = packet_entropy_priority(packetfile, args.block_size if args.block_size !=
None else 32)
        plot_whole_flow_entropy()
    elif (args.ping_frame):
      for i, packetfile in enumerate(args.packet_file):
        ped = packet_entropy_ping(packetfile, args.block_size if args.block_size != None
 else 32)
        plot_whole_flow_entropy()
    else:
      for i, packetfile in enumerate(args.packet_file):
        pe = packet_entropy(packetfile, args.block_size if args.block_size != None else
32)
        print ("Average Entropy")
        print (block_coordinates(np.around(pe[0], decimals=3), args.block_size if args.b
lock_size != None else 32))
        print ("Maximum Entropy")
        print (block_coordinates(np.around(pe[1], decimals=3), args.block_size if args.b
lock_size != None else 32))
        if args.label != None:
          plt.plot(pe[0], LINE[i], label=args.label[i].decode("utf-8"))
        else:
          plt.plot(pe[0], LINE[i])
    configure_plot()
    plt.show()
```