

VILNIAUS UNIVERSITETAS
FIZIKOS FAKULTETAS
BENDROSIOS FIZIKOS IR SPEKTROSKOPIJOS KATEDRA

Andrius Lengvinas

Netiesinės dinamikos metodų taikymas DDoS atakų aptikimui

Pagrindinių studijų bakalaурinis darbas

Kompiuterinės fizikos studijų programa

Studentas

Darbo vadovas

Recenzentas

Katedros vedėjas

Andrius Lengvinas

Doc. Feliksas Kuliešius

Doc. Alytis Gruodis

Prof., Dr. (HP) Valdas Šablinskas

Vilnius 2016

Turinys

Įvadas.....	3
1. Atakų aptikimo metodų apžvalga	4
1.1 DoS ir DDoS atakos.....	4
1.2 Pagrindiniai DoS ir DDoS atakų tipai	5
1.2.1 TCP SYN ataka	6
1.2.2 UDP ataka (UDP flood).....	7
1.2.3 ICMP(Ping) flood	7
1.3 DDoS atakų aptikimo metodai: privalumai ir trūkumai	8
1.4 Šaltinio IP adresų anomalijos metodas	10
1.5 Srauto entropijos metodas	10
1.6 Netiesinės dinamikos metodas.....	11
1.6.1 Rekonstruotos erdvės parametrų nustatymas	14
Mutual informacijos algoritmas	14
Autokoreliacija	16
Rekonstruotos erdvės dimensijos nustatymas	17
2. Tyrimo metodai	19
2.1 Tyrime naudojami duomenys	20
3. Rezultatai ir jų aptarimas.....	21
Metodo testavimas.....	21
Interneto srauto analizė.....	24
Pagrindiniai rezultatai ir išvados	32
Literatūros šaltinių sąrašas.....	33
Summary.....	37
Santrauka	38
Priedas	39

Išvadas

Internetas yra neatsiejamą mūsų gyvenimo dalis. Interneto teikiamų paslaugų prireikia kiekvienam iš mūsų atliekant kasdienes darbus. Žmonės internetu naudojami informacijos paieškai, apsiėitimui informacija, darbų valdymui bei priežiūrai, pirkimams, verslui ir t. t. Daugelio svarbių sistemų priežiūra taip pat atliekama internetu. Sparčiai auga ne tik naudojimasis internetu, bet ir neteisėta veikla. Šiandieninėje kibernetinėje erdvėje nuolatos vykdomos kibernetinės atakos. Vienos iš populiariausių yra atsisakymo aptarnauti – DoS (*Denial of service*) ir DDoS (*Distributed Denial of Service*) – atakos [1].

DoS ir DDoS atakų tikslas yra pakenkti aukos darbui arba jį visiškai sutrikdyti. Tai pasiekama apkraunant įrangą ar ryšio kanalą didelio intensyvumo srautu. Šis srautas apkrauna aukos įrangos procesorių ir interneto pralaidą. Priklausomai nuo atakos intensyvumo ir aukos įrangos skaičiavimų resursų, atakos metu gali tapti neįmanoma naudotis aukos įranga arba prie jos nebus galima net prisijungti. Internetiniai nusikaltėliai neapsiriboja vien tik paslaugų blokavimu – DDoS atakas dažnai panaudoja kitos veiklos maskavimui. Puolėjas pradeda DDoS ataką, norėdamas nusiėpti savo tikrąjį tikslą, pavyzdžiui, įsilaužimą arba duomenų vagystę, t. y. vykdydamas DDoS ataką, puolėjas gali nepastebėtas atlikti įsilaužimą į sistemą.

Norint sumažinti DoS ir DDoS atakų žalą, svarbiausia yra greitas ir tikslus jų aptikimas [1]. Aptikus ataką galima imtis prevencinių priemonių, skirtų sumažinti atakos keliamą grėsmę tinklo arba įrangos stabilumui. Atakų aptikimo metodai skirstomi į dvi grupes: elgsenos ir šablonų. Šabloniniai metodai puikiai veikia, kai yra žinomas vykstančios atakos kontrolinis kodas (*fingerprints*) [2]. Jeigu ataka vykdoma pasitelkus nežinomą algoritmą, tada naudojant šabloninius metodus atakos aptikti neįmanoma. Kai nežinomas atakos kontrolinis kodas, naudojami elgsenos metodai, dažniau statistiniai. Naudojant statistinius metodus, DDoS ataka aptinkama, stebint srauto anomalijas – nukrypimus nuo normos.

Darbo tikslas yra ištirti statistinių DDoS atakų aptikimo metodų taikymo galimybes bei jų patikimumą. Siekiant tikslo buvo keliami šie uždaviniai:

1. Išanalizuoti netiesinės dinamikos metodą.
2. Parašyti programą netiesinės dinamikos metodo analizei atlikti.
3. Išanalizuoti gautus rezultatus ir įvertinti metodo pritaikymo galimybes DDoS atakoms aptikti.

1. Atakų aptikimo metodų apžvalga

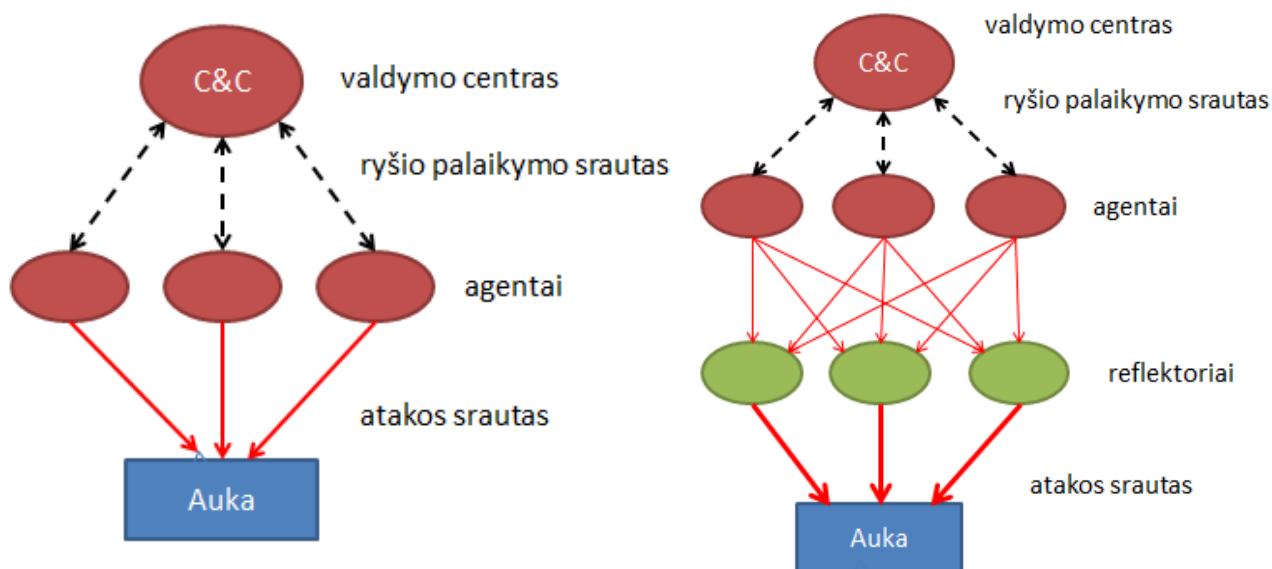
1.1 DoS ir DDoS atakos

Pirmosios DDoS atakos užfiksuotos praeito amžiaus devintajame dešimtmetyje. Yra žinoma, kad 1988 metais buvo įvykdytos tik 6 DDoS atakos. Nuo to laiko atakų skaičius auga eksponentiškai [1]. Auga ne tik atakų skaičius, bet ir jų intensyvumas. 2000 metais atakų aukomis tapo gerai žinomos interneto svetainės Yahoo ir Amazon. Atakos sparta tada buvo apie 1 Gb/s. 2007 metais DDoS atakos sparta siekė jau 70 GB/s, o 2013 metų kovo mėnesį buvo įvykdyta didžiausia DDoS ataka istorijoje. Didžiausio intensyvumo metu ataka pasiekė 300 Gb/s. Ilgiausiai trunkanti ataka, „Kaspersky Lab“ užfiksuota 2015 metų trečiąjį ketvirtį [3], tęsėsi 320 valandų, t. y. beveik dvi savaites. Negalima manyti, kad tai yra riba.

Pagrindinis DoS ir DDoS atakų tikslas yra sutrikdyti normalų aukos darbą. Šiomis atakomis siekiama užblokuoti legalių vartotojų prieigą prie įrangos, internetinio serverio ar kompiuterio. DoS atakomis siekiama išnaudoti puolamos įrangos silpnąsias vietas. Pavyzdžiui, interneto serveris vienu metu gali apdoroti ribotą kiekį užklausų, todėl puolėjui generuojant daugiau užklausų negu serveris gali apdoroti, legalūs vartotojai negali prisijungti. Puolėjui, siekiančiam įvykdyti sėkmingą DoS ataką prieš auką, yra reikalingas intensyvus informacijos srautas, koncentruotas į vieną taikinį. Priklausomai nuo puolamo įrenginio ar tinklo resursų, tai pasiekti naudojant vieną kompiuterį yra sudėtinga arba praktiškai neįmanoma, todėl DoS atakos paprastai rengiamos pasinaudojant kitais užkrėstais kompiuteriais. Jei atsisakymo aptarnauti atakoje dalyvauja keli kompiuteriai, ataka tampa paskirstyta (DDoS) ir gali būti daug kartų intensyvesnė nei DoS ataka. DDoS atakai naudojami kompiuteriai sudaro robotizuotų kompiuterių tinklą (*botnet*) [4].

Nusikaltėliai, norėdami sukurti robotizuotų kompiuterių tinklą DDoS atakoms, naudoja tinklo skenavimo įrankius, skirtus pažeidžiamų kompiuterių suradimui. Puolėjas, skenuodamas tinklą, aptinka įrenginius, kurie neturi antivirusinių programų ar šių programų versijos pasenusios, nenaudoja užkardų, naudoja įrangą su neatnaujinta operacinės sistemos versija. Tada puolėjas bando įsilaužti į šiuos įrenginius ir įdiegti juose atakos įrankius, skirtus DDoS srauto generavimui. Sėkmingai įdiegus programas, užkrėstas kompiuteris tampa agentu arba robotu (bot). Vienas iš jų yra išrenkamas valdymo centru (command and control, C&C) [5]. C&C perduoda informaciją agentams apie taikinį, atakos pradžios laiką ir puolimo algoritmą. Jeigu užkrėstas kompiuteris yra įjungtas į tinklą, tai nurodytu atakos laiku jis pradės generuoti DDoS atakos paketus, skirtus aukai. DDoS atakos metu agentų generuojami paketai siunčiami aukai tiesiogiai. Tokios atakos schema pateikta (1 pav. kairėje). DRDoS (*Distributed Reflection Denial of Service*) ataka pavaizduota (1 pav. dešinėje): šiuo atveju atakos srauto sustiprinimui naudojami reflektoriai. Agentai siunčia

užklaudas su aukos IP adresu (IP antraštės šaltinio lauke) reflektoriams, o šias užklaudas gavę reflektoriai mano, kad jos atėjo iš aukos ir į jas atsako. Taip sukuriamas padidintas atakos srautas. Reflektoriais gali būti įvairūs tinklo įrenginiai: kompiuteriai, serveriai ir t. t.



1 pav. DDoS (kairėje) ir DRDoS (dešinėje) atakų pavyzdžiai.

1.2 Pagrindiniai DoS ir DDoS atakų tipai

DDoS atakos yra tyčinis veiksmas skirtas padaryt tinklą ar jo įrenginius nepasiekiamus vartotojams. Paprastai tai padaroma sutrikdant arba sustabdant aukos normalią veiklą. DDoS atakas galima padalinti į tris tipus: intensyvaus srauto atakos, protokolo atakos ir aplikacijos lygio atakos.

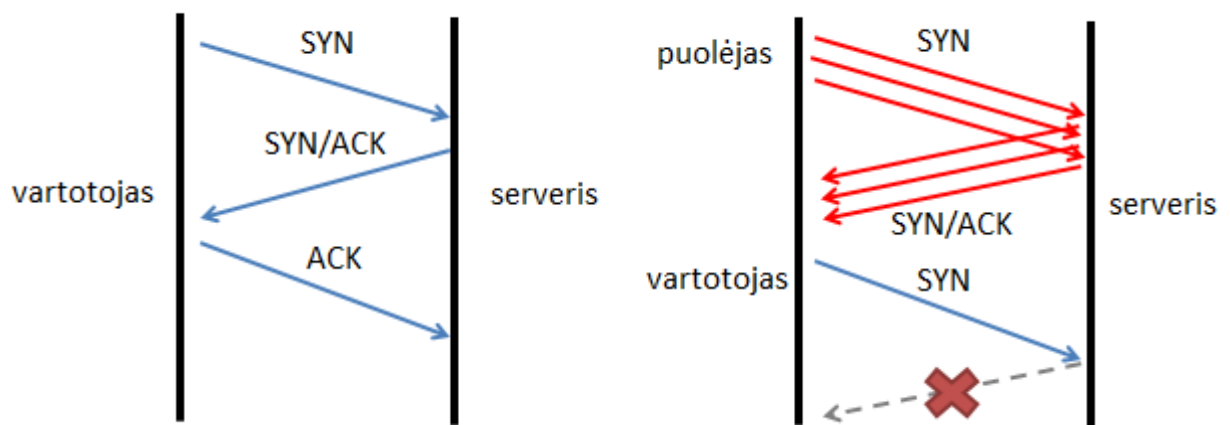
- Intensyvaus srauto atakas sudaro UDP flood, ICMP flood ir kitos atakos. Atakos tikslas yra visą pralaidą prisotinti atakos paketais. Atakos dydis vertinamas bitais per sekundę.
- Protokolo atakas sudaro SYN flood, Ping of Death, Smurf DDoS ir t. t. Šio tipo atakos eikvoja serverio arba tinklo įrenginių, kaip ugniasienės ir srauto lygintuvai, procesorių skaičiavimo resursus. Ataka yra matuojama paketais per sekundę.
- Aplikacijos lygio atakos yra mažo intensyvumo, GET/POST floods, šios atakos taikosi į Apache, Windows ar OpenBSD programinės įrangos pažeidžiamąsias vietas. Sudarytos iš tariamai legalių ir niekuo neišsiskiriančių užklausių. Atakos dydis matuojamas užklausomis per sekundę.

Didžioji dalis DDoS atakų yra nukreiptos prieš serverius, dažniausiai serverio darbui sutrikdyti naudojamos SYN flood, UDP flood, ICMP flood ir t. t.

1.2.1 TCP SYN ataka

TCP (Transmission Control Protocol) yra vienas pagrindinių protokolų iš internetinių protokolų rinkinio (Internet protocol suite). TCP protokolas yra skirtas ryšio užmezgimui tarp dviejų aplikacijų. TCP protokolas ryšiui užmegzti tarp vartotojo ir serverio naudoja „three-way handshake“ (2 pav.) t. y. trys etapai: ryšio sukūrimas, duomenų perdavimas ir ryšio nutraukimas [6]. Tam, kad vartotojas galėtų prisijungti serverio portas turi būti atviras prisijungimams. Kai tai įvykdoma, klientas gali inicijuoti aktyvų ryšį. Jis sukuriamas išsiunčiant paketą su pradiniu sekos skaičiumi (x) ir SYN (sinchronizacija) bitų rinkiniu. Serveris gauna SYN užklausą, nuskaito sekos numerį x ir atsako išsiųsdamas SYN+ACK (sinchronizacija ir patvirtinimas) $ACK = x + 1$. Serveris į paketą įtraukia ir savo sekos numerį atsitiktinį kintamąjį (y). $ACK = 20$ reiškia, kad serveris gavo baitus nuo 0 iki 19 ir tikisi sekančiu siuntimu gauti 20 baitą. Vartotojas priėmęs serverio išsiųstą užklausą jam išsiunčia ACK (patvirtinimas) paketą, kurį sudaro sekantis baitas, kurį vartotojas tikisi gauti ($ACK = y + 1$). Atlikus „three-way handshake“ apsiskeitimą paketais, klientas su serveriu užmezga tiesioginį ryšį, kuriuo gali bendrauti ir keistis duomenimis. Pirmas ir antras punktai inicijuoja ryšį į vieną pusę, o antrasis ir trečiasis – į antrą pusę. Taip sukuriamas pilnai dvipusis ryšys.

„SYN flood“ ataka atliekama pasinaudojus TCP protokolo „three-way handshake“ ryšio užmezgimo algoritmu. Atakos metu be perstojo siunčiamos SYN užklausos į visus atvirus serverio portus [7]. Prasidėjus tokiai atakai serveris atsako į kiekvieną gautą užklausą išsiųsdamas SYN/ACK patvirtinimą. Normalaus ryšio užmezgimo atveju vartotojas į SYN/ACK atsakytą išsiųsdamas ACK paketą. Atakuojantys įrenginiai niekada nesiunčia ACK paketo. Tačiau jie siunčia pakartotinai SYN užklausas, todėl prieš pasibaigiant sesijos laikui („time out“) gaunama nauja SYN užklausa. Puolėjas visada žino, kurie portai atidaryti, kai gauna SYN/ACK paketą. Tokios atakos metu aukos pralaida užpildoma beprasimiais paketais ir normalūs vartotojai negali pasiekti serverio (2 pav.).



2 pav. Kairėje užbaigtas ryšio užmezgimas, dešinėje puolėjo srautu apkrautas serveris negali atsakyti į legalią srauto užmezgimo užklausą.

1.2.2 UDP ataka (UDP flood)

UDP flood ataka naudoja „User Datagram Protocol“ (UDP), kuris, priešingai nei TCP prieš pradėdamas paketų siuntimą neužmezga sesijos [8]. Paketai perduodami UDP protokolu gali pasiekti adresatą ne tokia tvarka, kokia buvo išsiųsti arba gali išvis nepasiekti adresato. Šio tipo ataka siunčia didelius kiekius paketų į atsitiktinius įrenginio portus (virtualius), taip priversdama įrenginį daug kartų tikrinti ar yra aplikacija, kuri naudoja šį portą. Įrenginiui neradus jokios aplikacijos, jis gauto paketo šaltinio adresu išsiunčia „ICMP Destination Unreachable“ paketą. Šis procesas išnaudoja atakuojamo įrenginio resursus ir esant didelio intensyvumo atakai, įrenginys gali būti nepasiekiamas legaliems vartotojams.

1.2.3 ICMP(Ping) flood

Veikimo principas panašus į UDP flood atakos. ICMP ataka vykdoma generuojant be perstojo „ICMP Echo Request (ping)“ paketus ir nelaukiant atsako į juos. Sugeneruotas intensyvus nenutrūkstantis srautas apkrauna aukos interneto liniją ir sumažina jos pralaidą arba ją visiškai „užkemša“. Ši ataka „užkemša“ ne tik įeinančio srauto liniją, bet ir išeinančio srauto liniją, nes auka kiekvienai gautai „ICMP Echo Request (ping)“ užklausai sugeneruoja ir išsiunčia „ICMP Echo Reply“ paketą.

1.3 DDoS atakų aptikimo metodai: privalumai ir trūkumai

Siekiant apsisaugoti nuo DDoS atakų, yra sukurta ir pritaikyta įvairių saugumo mechanizmų. Šiuos metodus galima suskirstyti į tris grupes: aptikimo [9 – 14], apsaugos [11, 15 – 18] ir atakos šaltinio – IP atsekimo [19 – 21]. Pats svarbiausias etapas kovojant prieš DDoS atakas yra DDoS atakos aptikimas.

Atakų aptikimo metodus galima padalinti į dvi grupes: elgsenos ir šabloninius. Šie DDoS aptikimo metodai remiasi aktyviu profiliavimu [22, 23], paketų filtravimu [24, 25] nuosekliu „change-point“ aptikimu [9, 10] ir bangelių analize [26, 27]. Naudojant šabloninius metodus, atakos aptinkamos lyginant srautą su gerai žinomų atakų kontroliniais kodais (*fingerprints*) [2]. Srauto ir atakos šablono atitikimas įspėja apie galimą ataką. Tokio tipo metodais aptinkama dauguma žinomų atakų [19]. Metodas turi mažą klaidingai teigiamų rezultatų procentą (*false positive rate*). Klaidingai teigiami rezultatai fiksuojami, kai iš gaunamų duomenų nustatoma DDoS ataka, nors iš tikrųjų DDoS atakos nėra.

Dauguma dabartinių atakos aptikimo metodų yra šabloniniai, paremti žinomais atakų kontroliniais kodais [10 – 14]. Tokius metodus galime laikyti pasyviais [1], nes juos naudojant negalima aptikti naujų atakų.

Internetiniai nusikaltėliai nuolatos tobulina atakų algoritmus, todėl neįmanoma turėti kontrolinių kodų visoms galimoms DDoS atakoms. Siekiant aptikti naujo tipo DDoS atakas dažniausiai yra naudojami statistiniai metodai. Naudojant statistinius metodus, DDoS atakos yra aptinkamos stebint srauto anomalijas. Anomalija laikoma bet koks nukrypimas nuo legalaus srauto normų [29]. Statistiniuose metoduose dažniausiai srauto pakitimams fiksuoti naudojami šie paketo atributai: šaltinio ir paskirties IP adresai. Srauto anomalija paremta įsibrovimo aptikimo sistema skirstoma į tris stadijas [30]: parametrizavimo, modeliavimo ir aptikimo. Parametrizavimo stadijoje yra apibrėžiami sistemos parametrai. Modeliavimo stadijoje yra sukuriamas legalaus srauto modelis pagal vietinio tinklo srauto parametrus. Aptikimo stadijoje srautas yra lyginamas su antroje stadijoje sukurtu modeliu. Jeigu srautas neatitinka modelio, manoma, kad vyksta DDoS ataka. Statistiniai metodai yra tinkami aptikti atakas, kurios neturi joms sukurto šablono. Tačiau lyginant su šabloniniais metodais statistiniai metodai turi didesnį klaidingai teigiamų rezultatų procentą.

Atsirandant naujiems ir tobulesniems atakų aptikimo algoritmams, puolėjai kartu tobulina savo atakų algoritmus ir stengiasi nusišlėpti atakos šaltinį. Tikrojo atakos šaltinio paslėpimui, dažnai naudojama taktika yra apsimesimas kitu šaltiniu (*IP spoofing*) [2]. Paketo šuolių skaičiavimo filtras gali aptikti atakas su apsimestiniu šaltinio adresu. Žinoma, kad puolėjas gali suklastoti bet kurį lauką IP paketo antraštelėje, viena iš jų yra TTL (*time to live*). TTL yra paketo gyvavimo laikas

tinkle, kuris mažėja ties kiekvienu šuoliu tarp maršrutizatorių. Auka, siekdama aptikti ataką, gali savo įrangoje sudaryti bendrą IP adresų ir atstumo iki jų lentelę (*IP to hop count, IP2HC*), kurioje būtų informacija apie paketo šaltinio IP adresą ir jo TTL vertė. Remiantis šia lentele galima atrinkti apsimestinius IP adresus.

Apsimestinio IP adreso atakos aptinkamos analizuojant IP adresų lentelę. Atakai vykstant iš vieno šaltinio puolėjas naudoja vieną apsimestinį IP adresą, kuris yra realus teisėto vartotojo IP adresas, jį pažymėkime C_i . C_i generuoja n_i legalaus srauto paketų per laiko intervalą. Atakos paketus pažymėkime N_i , kuris tenkina sąlygą $N_i > n_i$. Tuomet aptikimo koeficientas yra:

$$Z_s = 1 - \frac{N_i}{n_i} = 1 - a_i, \quad (1)$$

čia, a_i yra trupmeninė seka (*fraction following*).

Atakai vykstant iš z ($z > 1$) skirtingų šaltinių, hakeris naudoja z legalių vartotojų IP adresų $C_1, C_2, \dots, C_z$, ir kiekvienas iš jų siunčia $N_1, N_2, \dots, N_z$ atakos paketų aukai, o jų susijusios trupmeninės dalys yra $a_1, a_2, \dots, a_z$ iš (1) formulės:

$$Z_m = \frac{\sum_{i=1}^z (1 - a_i) N_i}{\sum_{i=1}^z N_i}. \quad (2)$$

Tuo atveju, jeigu apsimestiniai paketai yra netolygiai išdalinti tarp z IP adresų, aptikimo koeficientą galima įvertinti kaip:

$$Z_m = 1 - \frac{1}{z} \sum_{i=1}^z a_i. \quad (3)$$

Puolėjas, turintis kelis šaltinius, žinodamas apie naudojamą aptikimo metodą, bet neturintis tikslios informacijos, vengdamas aptikimo gali pradėti generuoti pradinę TTL vertę intervale $[h_m, h_n]$, naudojant pasirinktą, pavyzdžiui Gauso pasiskirstymą. Tarkime, kad šuolių skaičiaus h_k pasirinktam pasiskirstymui tikimybė yra p_k . Tada iš lygties (3) seka, kad trupmeninė IP adresų dalis a_k turi šuolių skaičių h_k , o aptikimo koeficientas yra:

$$Z_{ami} = 1 - \sum_{k=m}^z a_k p_k. \quad (4)$$

Jeigu puolėjas išanalizavęs auką geriau, tuomet jis gali pasiekti mažesnį aptikimo koeficientą manipuliudamas atakos srautu ir keisdamas TTL verčių pasiskirstymą.

1.4 Šaltinio IP adresų anomalijos metodas

M. Alenezi ir M. J Reed [28] rašo apie DDoS aptikimo metodą paremtą šaltinio IP adresu. Sistema, užuot stebėjusi srautą, stebi naujai atsirandančius šaltinio IP adresus. Šis metodas paremtas Jung tyrimu [31]. Remiantis tyrimo rezultatais galima teigti, kad atakos metu dauguma šaltinio IP adresų yra nauji. Pavyzdžiui, analizuojant du laiko intervalus, naujais IP adresais laikomi tie IP adresai, esantys antrame laiko intervale, kurie neturi savo atitikmens pirmame intervale. Tikrinti galima daugiau negu su vienu intervalu. Nustatyta, kad staigaus legalaus srauto išaugimo (*flash crowd*) metu dauguma IP adresų nėra nauji. „Flash crowd“ yra ženklus apkrovos padidėjimas tinklo serveriui, sukeltas legalių vartotojų. Todėl šis metodas puikiai tinka atakų, kurios naudoja apsimestinius IP adresus, aptikimui. Jeigu puolėjas pirma išanalizuoja situaciją ir tik tada organizuoja ataką, naudojant realius IP adresus, jam išvengti aptikimo bus nesunku.

1.5 Srauto entropijos metodas

Entropija yra sistemos netvarkos matas rodantis, kaip arti pusiausvyros yra sistema [32]. Todėl ji yra viena iš pagrindinių metrikų, naudojamų informacinėse technologijose [1]. Atsitiktinio kintamojo X entropija apibrėžiama taip:

$$H(X) = -\sum_{x \in \chi} (\text{Pr}\{X = x\}) \log_2(\text{Pr}\{X = x\}), \quad (5)$$

kur x priklauso bandinių erdvei χ . Entropija naudojama norint apskaičiuoti paketų atributų pasiskirstymą intervale.

Tsalio entropija yra supaprastinta Boltzmano–Gibso entropija. Tarkime $P = \{p_1, p_2, \dots, p_N\}$ yra elementų skaičiaus N tikimybių pasiskirstymas. Kintamasis p_i turi tenkinti sąlygas $0 \leq p_i \leq 1$ ir $\sum_i p_i = 1$, entropija H_q apibrėžiama taip:

$$H_q = \frac{1 - \sum_{i=1}^N p_i^q}{q-1}, \quad (6)$$

H_q yra vadinama Tsalio entropija. Čia q yra vadinamas entropiniu parametru. q kitimas keičia atskiros įvykio santykinę įnašą visam srautui. Kai q vertė lygi 1 Tsalio entropija yra lygi Šenono entropijai. N identifikuoja duomenų skaičių, o p_i nurodo jo tikimybę. Entropijos vertė gali kisti nuo

0 iki H_q^{max} , tai parodo maksimalią koncentraciją arba maksimalų išsklaidymą. Maksimali entropija H_q^{max} lygi:

$$H_q^{max} = \frac{1-N^{1-q}}{q-1}. \quad (7)$$

DDoS atakų aptikimui dažniausiai naudojama srauto entropijos analizė. Kadangi DDoS atakos metu aukos IP adresai yra pastovūs ir puolėjas negali pakeisti šio paketo lauko informacijos, skaičiuojama srauto entropija pagal paskirties adresą negali būti pakeista puolėjo, kuris naudoja apsimestinius šaltinio IP adresus ar manipuliavimą TTL vertėmis

Bendru atveju, kol nevyksta DDoS ataka, srauto entropija yra pastovi, prasidėjus atakai, srauto entropija ženkliai sumažėja arba padidėja per keletą sekundžių, nes atakos srautas daug intensyvesnis lyginant su legaliais srautais vietiniame tinkle. Esant normaliam srautui, nustatomos slenkstinės ribos g ir h (minimali ir maksimali entropijos galima vertė, kol nevyksta ataka). Fiksavus srauto entropijos kitimą, kuris peržengia nustatytas ribas (8 formulė), galima teigti, kad vyksta DDoS ataka.

$$|H(X)| < g \text{ arba } |H(X)| > h \quad (8)$$

Svarbiausia yra tinkamai pasirinkti g ir h vertes, nes nustačius per mažus rėžius, padaugėja klaidingų pranešimų apie DDoS atakas, o nustačius per didelius – dalis atakų bus neaptinkamos.

Tais atvejais, kai srauto entropija naudojama ilgo laikotarpio srauto stebėjimui, jos vertės gali peržengti nustatytą slenkstinę ribą, nors DDoS ataka nevyksta. Taip nutinka dėl besikeičiančių legalių vartotojų poreikių, o tai sukuria srauto augimą (*flash crowd*). Todėl patartina naudoti sunormuotą srauto entropiją:

$$E = \frac{(\sum_{i=1}^m p_i \log_b p_i)}{(\sum_{i=1}^n \log_b \frac{1}{n})}, \quad (9)$$

kur m yra unikalių IP adresų skaičius intervale, n yra paketų skaičius intervale, $p_i = \frac{a_i}{n}$, a_i yra unikalaus IP adreso pasikartojimų skaičius intervale, b yra logaritmo pagrindas, dažniausiai naudojama 2 arba 10.

1.6 Netiesinės dinamikos metodas

Nors internete perduodamų informacijos paketų srauto intensyvumą ir pasiskirstymą sunku (turbūt neįmanoma) aprašyti deterministiniu modeliu, šis srautas nėra visiškai atsitiktinis. Todėl informacijos srautą galima traktuoti kaip nestabilią dinaminę sistemą. Tokios sistemos vadinamos

chaotinėmis sistemomis. Nors nėra universalus ir priimto chaoso apibrėžimo, laikoma, kad dinaminė sistema elgiasi chaotiškai, jeigu: 1) ji yra labai jautri pradinėms sąlygoms, 2) ji turi savybę topologiškai maišytis, ir 3) jos periodinė orbita yra koncentruota (turi atraktorių). Taikant netiesinės dinamikos metodus DDoS atakų aptikimui, remiamasi prielaida, kad galima nustatyti, kaip kis nuo laiko priklausanti tiesinė sistema net nežinant analizinės modelio išraiškos [33]. Prognozės gali būti gaunamos analizuojant laikines eksperimento rezultatų sekas. Siekiant nustatyti ar sistema yra visiškai atsitiktinė, ar ji yra chaotiška ir jai galima taikyti netiesinės dinamikos metodus, pirmiausia reikia įvertinti ar tiriamo srauto fazinės erdvės diagrama sudaro atraktorių. Kiti stabilumo požymiai yra maža koreliacinė dimensija ir maksimalios Liapunovo eksponentės vertės [34].

Vystantis chaoso teorijai ir jos pritaikymo galimybėms, buvo pristatomi vis nauji metodai skirti prognozuoti chaotiškas laiko sekas [35-40]. Šiuos metodus galima suskirstyti į dvi klases: globalius ir lokalius. Kitimo funkcijos modeliavimui globalūs metodai naudoja visus laiko serijoje esančius duomenis. Aišku, kad tokio tipo metodai turi akivaizdų trūkumą, visi modelio parametrai gali pasikeisti įtraukus naują informaciją ir jų įvertinimas iš naujo atima daug laiko bei skaičiavimų resursų. Lokalūs metodai neturi šio trūkumo, nes jų modelis sudaromas panaudojant tik dalį informacijos iš surinktų duomenų.

Pagrindas chaotiškų laiko serijų prognozavimui daugeliu atvejų yra rekonstruotos erdvės, kuri yra ekvivalenti originalios sistemos būsenos erdvei, sudarymas iš viendimensinės laikinės eilutės [41]. Taken'as, remiantis Whitney'ų topologine rekonstrukcijos sistema įrodė rekonstrukcijos pagal laiko delką teoremą [42]. Pritaikius ją galima suformuoti m dimensijų dinamiką iš laiko serijos dalies [43-45]. Sakykime užfiksuota skaliarinė laiko serija yra $\{x(n), n = 1, 2, 3, \dots, N\}$. Pagal Taken'o teoremą nežinomo chaotiško atraktoriaus rekonstruota erdvė gali būti sudaryta iš delsos vektorių.

$$X(n) = [x(n), x(n - \tau), \dots, x(n - (m - 2)\tau), x(n - (m - 1)\tau)]^T, \quad (10)$$

kur $n = (m - 1)\tau + 1, (m - 1)\tau + 2, \dots, N$. Iš laiko delsos koordinačių m yra rekonstruotos erdvės dimensijų skaičius, o τ yra laiko delsa. Norint atlikti būsenos erdvės atstatymą svarbiausia yra nustatyti optimalius parametrus. Tais atvejais kai rekonstrukcijos dimensija yra didesnė už minimalią rekonstrukcijos dimensiją sudaryto vektoriaus trajektorija gali atspindėti tikrą chaotinės sistemos būsenos kitimą erdvėje. Iš matematinės pusės laiko delsa τ yra sutartinė, nes duomenys laikomi esą begalinio tikslumo.

Lokaliam tiesiniam prognozavimui atlikti pirmas žingsnis yra atkurtos fazinės erdvės kaimyninių taškų nustatymas iš $X(n)$. Pagal atkurtą trajektoriją apskaičiuojamas atstumas tarp

esamo delsos vektoriaus $X(n)$ ir jo sekančio delsos vektoriaus $X(i)$ ($i = 1, 2, \dots, n - 1$), su įvesta metrika $\|\cdot\|$.

$$d(i) = \|X(i) - X(n)\|, \quad (11)$$

Apskaičiavus atstumus tarp vektorių surandame artimiausių taškų t. y. kaimynų skaičius k , $X(n_i)$ ($i = 1, 2, \dots, k$). Kaimyninių taškų paieškai yra parenkamos atitinkamos rekonstruotos erdvės dimensijų ir laiko delsos vertės m_{opt} ir τ_{opt} , kurios yra optimalios rekonstruotos erdvės sudarymui, k vertė parenkama būtinai didesnė už m_{opt} [35]. Lokalaus tiesinio prognozavimo modelis su m_{opt} dimensijų rekonstruota erdve yra kito m_{opt} autoregresyvus modelis, todėl prognozavimas yra tiesinė delsos vektoriaus m_{opt} elementų superpozicija t. y.

$$\hat{x}(n + 1) = a_0 + \sum_{i=1}^{m_{opt}} a_i x(n - 1(i - 1)\tau_{opt}) = A_1 Y_1(n), \quad (12)$$

čia $A_1 = [a_0, a_1, a_2, \dots, a_{m_{opt}}]$,

$$Y_1(n) = [1, X(n)^T]^T = [1, x(n), x(n - \tau_{opt}), \dots, x(n - (m_{opt} - 1)\tau_{opt})]^T.$$

Deterministiniame prognozavimo modelyje daroma prielaida, kad rekonstruota trajektorija yra nulemta deterministiškai nuolatos sudarinėjant F tarp esamos ir būsimos būsenos. Jeigu būsena laiku j buvo panaši esamai būsenai $X(n)$, F tęstinumas garantuoja, kad būsena $X(j+1)$ taip pat bus artima ateities būsenai $X(n+1)$.

Lokalaus tiesinio prognozavimo modelio vertės nebūtinai turi tenkinti šias sąlygas $m = m_{opt}$ ir $\tau = \tau_{opt}$, kurios yra optimalios būsenos erdvės sudarymui ir kaimyninių taškų nustatymui, bet nėra optimalios atliekant prognozavimą [42]. Parametrus m ir τ turėtų būti parinktos tokios vertės, kurios yra tinkamos prognozavimui atlikimui. Tuomet prognozavimo modelis parametrizuotas su vertėmis $\{m, \tau\}$ suformuluojamas taip:

$$\hat{x}(n + 1) = a_0 + \sum_{i=1}^m a_i x(n - (i - 1)\tau) = A_2 Y_2(n), \quad (13)$$

kur $A_2 = [a_0, a_1, a_2, \dots, a_m]$,

$$Y_2(n) = [1, x(n), x(n - \tau), \dots, x(n - (m - 1)\tau)]^T.$$

Koeficiento vektoriaus A_2 vertės priklauso nuo $X(n)$ būsenos ir jo topologinio kaimyno $X(n_i)$ ($i = 1, 2, \dots, k$). Koeficiento vektoriaus A_2 vertės apskaičiuojamos pagal sekančia lygtį.

$$A_2 B = D, \quad (14)$$

kur B yra $(m + 1) \times k$ dydžio matrica, kurios i -tasis stulpelis yra sudarytas iš $Y_2(n_i)$, o D yra $1 \times k$ vektorius, kurio i -tasis stulpelis yra sudarytas iš $x(n_i + 1)$.

$$Y_2(n_i) = [1, x(n_i), x(n_i - \tau), \dots, x(n_i - (m - 1)\tau)]^T, \quad (15)$$

$$D = [x(n_1 + 1), x(n_2 + 1), x(n_3 + 1), \dots, x(n_k + 1)]. \quad (16)$$

Pasinaudojus tokia pačia schema galima apskaičiuoti $\hat{x}(n + 2)$, kuris seka po paskutinio taško, kuris dabar yra $X(n + 1)$.

1.6.1 Rekonstruotos erdvės parametrų nustatymas

Būsenos erdvės atstatymas yra neišvengimas žingsnis laiko serijų analizavime. Jeigu duomenys yra sugeneruoti sistemos su atraktoriumi, kurį sudaro d dimensijų. Atraktorius yra aibė taškų kuriais juda dinaminė sistema praėjus pakankamai daug laiko [46]. Tada pasinaudojus Taken'o teorema, kuri teigia, kad galima sudaryti rekonstruotą fazinę erdvę artimą originaliam atraktoriui iš vienos laiko serijos, kai ją sudarot $(2d + 1)$ dimensijų [47]. Alternatyvūs atstatymo metodai: išvestinės arba tiesinio nepriklausomumo koordinatės surastos atlikus esminių komponentų analizę, gali būti traktuojami, kaip laiko postūmio vektoriaus transformacija [47]. Pasitelkiant vieną iš šių rekonstrukcijos procedūrų yra sudaroma nauja fazinė erdvė, kuri yra ekvivalenti originaliai fazinei erdvei. Rekonstrukcijai atlikti reikia nuspręsti kokius parametrus naudoti rekonstruotai erdvei sudaryti, t. y. laiko delsos vertė tarp koordinatžių ir dimensijų skaičiaus.

Tiesinėms ir netiesinėms laiko serijoms tirti bei jų fazinėms erdvėms sudaryti reikalingas metodas skirtas nustatyti tinkamą laiko delsos vertę τ . Jeigu delsos vertė bus per didelė sudarytoje fazinėje erdvėje verčių koordinatės iš esmės bus nepriklausomos viena nuo kitos ir negalėsime gauti tikslios informacijos. Optimaliai delsos vertei nustatyti naudojami bendrosios (*mutual*) informacijos ir autokoreliacijos metodai.

Bendrosios informacijos algoritmas

Bendroji (*mutual*) informacija tikimybių ir informacijos teorijoje yra dviejų atsitiktinių kintamųjų rodiklis bendros priklausomybės tarp šių dviejų kintamųjų įvertinimui. Ji kiekybiškai nustato informacijos kiekį (vienetais, t. y. bitais) gautą iš vieno atsitiktinio kintamojo per kitą atsitiktinį kintamąjį [48].

Bendroji informacija yra plačiai naudojama siekiant nustatyti vieną iš svarbiausių fazinės erdvės rekonstrukcijai iš netiesinių procesų laiko eilučių reikalingų parametrų t. y. laiko delsą (*lag*).

Jeigu $\{s(t_i)\}(i = 1, 2 \dots N)$ yra laiko eilutė gauta atlikus skaičiavimus ar eksperimentą, kurioje nuskaitymo intervalas yra Δt tada duomenys gali būti perstatyti taip:

[illegible]

čia τ yra laiko delsa. Bendroji informacija tarp $S = \{s(t_1), (t_2), \dots, s(t_N)\}$ ir $Q = \{(t_1 + \tau), (t_2 + \tau), \dots, s(t_N + \tau)\}$ yra $I(S, Q)$, kur S gali būti prognozuojamas naudojant Q matavimus. $I(S, Q)$ galima išreikšti taip:

$$I(S, Q) = H(Q) + H(S) + H(S, Q), \quad (18)$$

čia $H(Q)$ ir $H(S)$ yra Q ir S entropijos atitinkamai, o $H(S, Q)$ yra bendra entropija tarp S ir Q . Dažniausiai laiko serijos fazinės erdvės atvaizdavimui, kaip optimali laiko delsa naudojamos pirmos bendrosios informacijos minimumas.

Siekiant nustatyti bendrąją informaciją tarp dviejų laiko serijų S ir Q , jų skaitinės vertės surašomoms ant dviejų statmenų ašių, sudaroma speciali matrica. Pirmiausia iš Q serijos nustatoma mažiausia Q vertė ir ji yra atimama iš visų Q imties verčių. Tada surandama didžiausia Q vertė ir iš jos padalinamos visos Q vertės - taip atliekamas sunormavimas į rėžį nuo 0 iki 1. Iš šių sunormuotų verčių sudarome seka qq_j , $j = 1, 2, 3, \dots, N$. Tuomet, vertės yra vėl pakeičiamos pagal formules aprašytas žemiau ir gaunama nauja seka.

$$qqq_j = \begin{cases} (int)qq_jb, & qq_j < 1 \\ b, & qq_j = 1 \end{cases} \quad (19)$$

$$b = (\text{int}) (\log_2 N + 0.5), \quad (20)$$

čia b yra optimali matricos dimensija, o qqq_j apibrėžia matricos kintamojo indeksą. Atlikus tokius pat veiksmus su S galime užpildyti $b \times b$ matricą vertėmis. Iš pradžių visos matricos vertės lygios 0. Matricos užpildymui pritaikomas toks algoritmas, kur qqq_j vertės yra matricos eilutės indeksas, o sss_j yra stulpelio indeksas. Tada superpoziciją tarp s_i ir q_i galima išreikšti, kaip (s_i, q_i) ($i = 1, 2, 3, \dots, b$), kuri yra lygi sumai visų pasikartojančių qqq_j ir sss_j , kurių vertės yra lygios i (3pav.). Tuomet taškų, kurie atitinka duomenų poras iš q_i tikimybės pasiskirstymas $P(q_i)$,

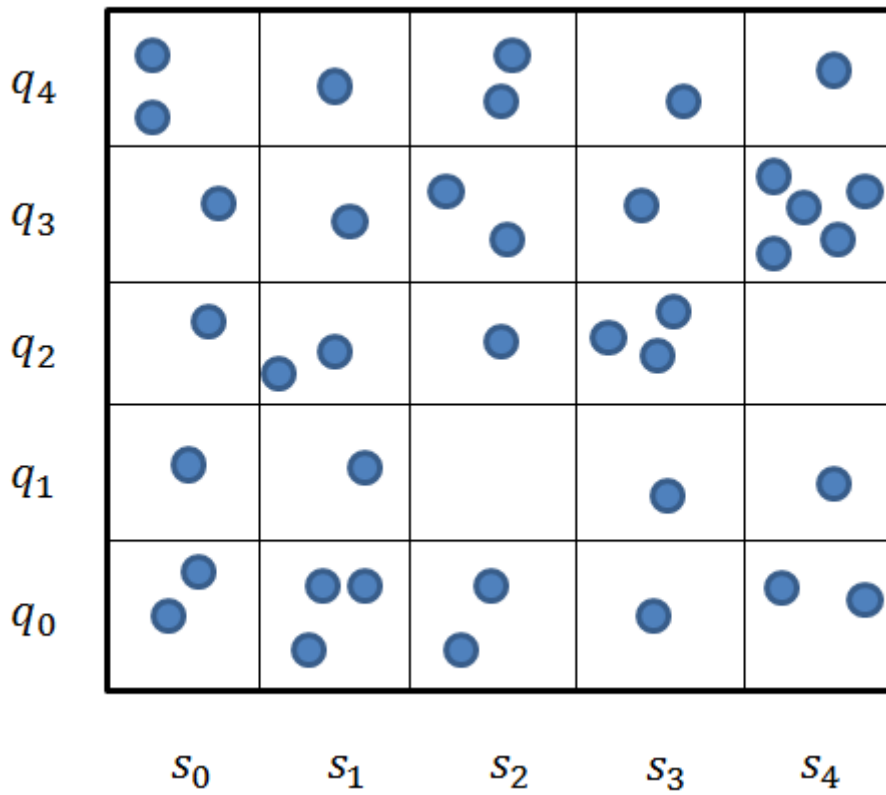
gali būti nustatytas padalinant q_i iš duomenų serijos dydžio N . (s_i, q_i) , tikimybės pasiskirstymas $P(s_i, q_i)$ lygus santykiui tarp (s_i, q_i) taškų skaičiaus ir N . Todėl bendroji informacija tarp S ir Q gali būti apskaičiuojama pasinaudojant lygtimis (21 – 24).

$$H(Q) = -\sum P(q_i) \log_2 P(q_i), \quad (21)$$

$$H(S) = -\sum P(s_i) \log_2 P(s_i), \quad (22)$$

$$H(Q, S) = H(S, Q) = -\sum P(s_i, q_i) \log_2 P(s_i, q_i), \quad (23)$$

$$I(Q, S) = \sum_i \sum_j P_{sq}(s_i, q_j) \log_2 \left[\frac{P_{sq}(s_i, q_j)}{P_s(s_i)P_q(q_j)} \right]. \quad (24)$$



3 pav. Lygaus tikimybės pasiskirstymo kiekvienam elementui matrica.

Autokoreliacija

Koreliacija tikimybių teorijoje ir statistikoje yra statistinis ryšys tarp kintamųjų, parodantis ar šie duomenys yra susiję ir jeigu susiję, tai kaip stipriai jie susiję [49]. Koreliacijos koeficientas yra koreliacijos stiprumo matas. Jeigu dviejų kintamųjų koreliacijos koeficientas lygus nuliui, tai tie kintamieji yra statistiškai nepriklausomi. Pasinaudojus koreliacijos teorija, galima nustatyti delką netiesinėms sekoms.

Koreliaciją tarp duomenų x_i $i = 1, 2, 3, \dots, N$ ir paslinktų duomenų x_{i+j} galima apskaičiuoti pagal formulę.

$$R_j = \frac{\sum (x_i - \bar{x})(x_{i+j} - \bar{x})}{\sum (x_i - \bar{x})^2}, \quad (25)$$

čia $\bar{x}$ yra sekos vidurkis R_j yra auto koreliacijos koeficientas, kai delsa lygi j . Optimali laiko delsos vertė nustatoma pagal gautos kreivės pirmą vertę, kuri kerta x-ą ašį.

Rekonstruotos erdvės dimensijos nustatymas

Rekonstruotos erdvės dimensijų parinkimas yra toks pats svarbus erdvės atstatymui, kaip ir laiko delsos vertės nustatymas. Pagal Whitney's teoremą galima m dimensijų atstatyti sudarant $(2m + 1)$ dimensijų Euklidinę erdvę [50]. Kartais reikalingas dydis rekonstruoti fazinei erdvei gali būti mažesnis priklausomai nuo skaičiavimų tikslumo ir triukšmų.

Pasirinkti galimai minimalią rekonstruotos erdvės dimensiją yra sudėtinga, kai originalios sistemos laisvės laipsnių skaičius yra nežinomas. Erdvė sudaryta parinkus per mažą dimensijų skaičių nepaaiškina trajektorijos, o per didelė rekonstruota erdvė gali sukelti persidengimus. Įprastai tinkamos dimensijos paieška atliekama žingsnis po žingsnio plečiant rekonstrukcijos erdvę ir kartu stebint tam tikras vertes, kurios turėtų išlikti nepakitusios pasiekus tinkamo dydžio atstatytos erdvės dimensiją. Atstatymo metu pasirinktinai stebima viena iš šių verčių: koreliacijos dimensija, didžiausia Lyapunov'o eksponentė, nuspėjamumo indeksas arba netikri artimiausi kaimynai.

Klaidingų artimiausių kaimynų algoritmas

Klaidingų artimiausių kaimynų (*false nearest neighbours*, *FNN*) algoritmas yra populiariausias įrankis pasirenkant minimalią rekonstrukcijos dimensiją [51]. Šis metodas yra paremtas prielaida, kad du taškai, kurie arti vienas kito tinkamoje rekonstruotoje dimensijoje m turi išlikti artimi dimensijai didėjant. Tačiau, esant per mažai rekonstrukcijos dimensijai taškai, kurie yra toli vienas nuo kito gali pasirodyti esą kaimynai. Įvairios metodo modifikacijos pritaiko geometrinius samprotavimus: vienos iš jų didina rekonstruojamos erdvės dimensijų skaičių ir tikrina kaimynus didėjančioje rekonstruotoje erdvėje, kol suranda nedidelį iš anksto nustatytą netikrų kaimynų skaičiaus pakitimą tarp dimensijų m ir $m + 1$. Tuomet m yra parenkamas, kaip minimalus rekonstruotos erdvės dimensijų skaičius. Darome prielaidą, kad su šita m verte turėsime rekonstrukciją be persidengimų.

Cao's algoritmas

Viena iš *FNN* metodo problemų yra subjektyvus kelių parametrų pasirinkimas. Pavyzdžiui Kennle's algoritme yra naudojamos R ir A atskirti tikrus kaimynus nuo netikrų bei kitos vertės skirtos nustatyti, kada surastų netikrų kaimynų skaičius yra pakankamai mažas. Su skirtingomis parametrų vertėmis Kennle's algoritmas gražina skirtingas optimalios dimensijos vertes. Cao pristatė modifikuotą algoritmą, kuris dar vadinamas klaidingų kaimynų vidurkio metodu [52]. Užuo testavus ar kaimynai yra tikri ar ne, Cao skaičiuoja, kaip vidutiniškai pasikeitė atstumas tarp artimiausių kaimynų perėjus nuo dimensijos m prie $m + 1$. Dimensija, ties kuria atstumas nebesikeičia, yra parenkama kaip optimali. Pagrindinis Cao metodo privalumas yra sumažintas subjektyviai parenkamų parametrų skaičius.

2. Tyrimo metodai

Tiriant netiesinės dinamikos metodo patikimumą DDoS atakų aptikimui bakalauro darbo metu buvo parašyta programa, skirta šio metodo analizei. Programa parašyta JAVA programavimo kalba. Ši kalba pasirinkta todėl, nes jau yra parašyti bibliotekų failai, skirti komunikacijai su duomenų baze „PostgreSQL9.5“, kurioje laikomi tiriamo srauto duomenys. Duomenų bazė naudojama siekiant paspartinti skaičiavimus, išnaudojant duomenų bazės optimizuotus algoritmus, skirtus duomenų atrinkimui pagal tam tikras sąlygas.

Duomenų bazėje saugoma tokia paketo informacija

1. Unikalus paketo laikas „timestamp“.
2. Paketo šaltinio IP adresas.
3. Paketo paskirties IP adresas.
4. Paketo perdavimo protokolas.

Parašytas programinis kodas (priede optimalios delsos nustatymo klasė) skirtas apskaičiuoti dviejų srautų tarpusavio autokoreliacijos vertes keičiant laiko delsos vertę. Pirmiausia paimami duomenys iš duomenų bazės, tada skaičiavimai atliekami pagal 25 formulę. Atlikus skaičiavimus gauname autokoreliacijos verčių seką r_i ($i = 1, 2, 3, \dots, M$), kurios vertės yra suvidurkinamos, taip sumažinamas triukšmas. Vidurkinimas atliekamas su pasirinktu vidurkinamų verčių skaičiumi $vidSk$ pagal 26 formulę. Atlikus verčių suvidurkinimą, naujos sekos rezultatai atidedami grafike ir nustatoma delsos vertė pagal kreivės pirmą tašką, kuris kerta x ašį. Jeigu $vidSk = 0$, vidurkinimas neatliekamas.

$$rr_i = \frac{r_i + \sum_{j=1}^{vidSk} (r_{i-j} + r_{i+j})}{2 * vidSk + 1} \quad (26)$$

Programos kodas, skirtas apskaičiuoti bendrajai informacijai veikia labai panašiai, lyginant su autokoreliacijos kodu, tačiau optimali deltos vertė nustatoma iš verčių atidėtų grafike, pagal pirmą bendrosios informacijos minimumo vertę.

Nustačius optimalią delsos vertę, galima atlikti srauto prognozavimą. Pirmiausia, reikia nurodyti tiriamo srauto pradžios ir pabaigos laikus, įvesti nustatytą delsos vertę ir norimą prognozuojamų verčių skaičių. Skaičiavimai atliekami pagal 13 – 16 formules, programos kodas pateiktas priede (srauto verčių prognozė).

2.1 Tyrime naudojami duomenys

Skaičiavimai atliekami su oficialiai publikuojamais srauto duomenimis. Duomenys paimti iš MIT (*Massachusetts Institute of Technology*) Linkolno laboratorijos (*Lincoln Laboratory*) [53] ir CAIDA (*Center for Applied Internet Data Analysis*) [54] Kalifornijos universiteto tyrimų skyriaus. Tyrime iš MIT laboratorijos buvo analizuojamas 1999 metų DARPA Intrusion Detection Evaluation Data Set [55]. Iš CAIDA duomenų bazėje publikuojamų duomenų pasirinktas „DDoS Attack 2007“ [56] duomenų failas. Būtent šie duomenų rinkiniai pasirinkti, nes jie yra pateikiami su papildoma svarbia informacija apie srautą: srauto fiksavimo laikotarpis, DDoS atakų užfiksuotų sraute pradžios ir pabaigos laikai, atakų pobūdžio tipus bei atakų šaltinių IP adresus.

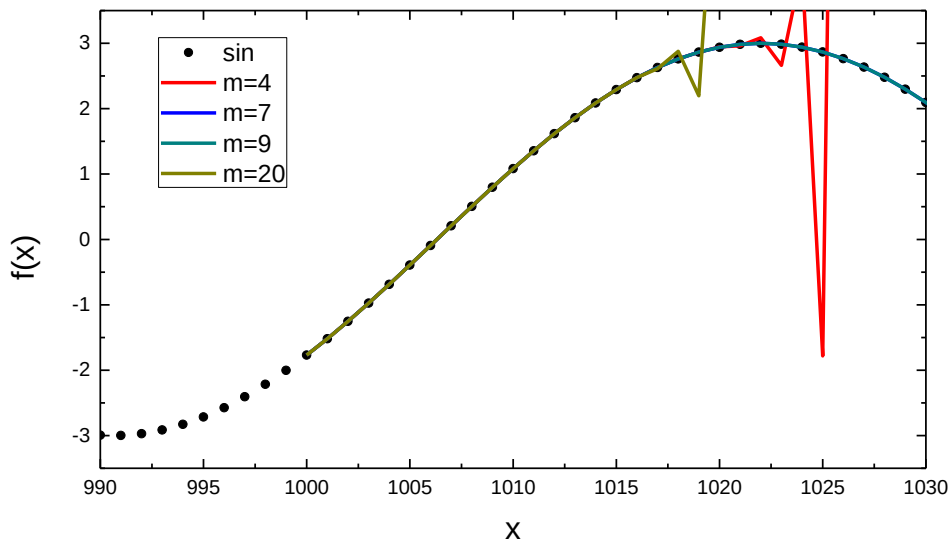
1999 metų srauto duomenų rinkinį sudaro duomenys surinkti fiksuojant srautą penkias savaites. Per penkias savaites surinktus duomenis galima padalinti į dvi grupes: mokomieji duomenys (*training data*) ir testavimo duomenys (*testing data*). Pirmą grupę sudaro pirmos [57] antros [58] ir trečios [59] savaitės duomenų srautai. Šiuos duomenų failus sudaro tik legalus srautas, failai yra be DDoS atakų paketų. Antrą grupę sudaro ketvirta [60] ir penkta [61] savaitės. Šiuo srauto stebėjimo laikotarpiu buvo užfiksuotos 56 skirtingos atakos, o iš viso užfiksuota 201 ataka [54]. Srauto duomenų aprašo faile [62] pateikiama papildoma meta informacija: atakos laikas, atakos IP adresas, trukmė, atakos tipas ir t. t. Darbe naudojamas šio duomenų rinkinio pirmos savaitės pirmos dienos duomenų srautas [63], toliau įvardijamas kaip MIT1999.

CAIDA duomenų faile „DDoS Attack 2007“ [56] yra užfiksuoti apytiksliai vienos valandos ilgio srauto duomenys su didelio intensyvumo DDoS ataka. Duomenų failas sudarytas iš srauto duomenų, užfiksuotų 2007 metų rugpjūčio 4 dieną, nuo 20:50:08 iki 21:56:16, fiksuojant puolamo serverio paketų srautą. Užfiksuota DDoS ataka prasideda 21:13. Srauto duomenys yra padalinti į dvi dalis ir saugomi atskirai. Pirmą dalį sudaro paketai, kuriuos gavo auka, t. y. paketai, kurių paskirties IP adresas yra aukos IP adresas. Antrąją dalį sudaro visi paketai, kuriuos auka išsiuntė, t. y. paketai, kurių šaltinio IP adresas yra aukos IP adresas. Bendras duomenų rinkinio dydis yra 21 Gb, todėl duomenys yra sudalinti į mažesnius failus, kuriuose saugomi penkių minučių ilgio srauto duomenys. Šie duomenų failai saugomi (tcpdump) formatu ir jų peržiūrai gali būti naudojamos: CoralReef Software Suite, tcpdump, Wireshark, ir kitos programos. Šiame tyrime naudojamus duomenis sudaro paketai, kurių paskirties IP adresas yra aukos IP adresas. Toliau šie duomenis įvardinami CAIDA2007.

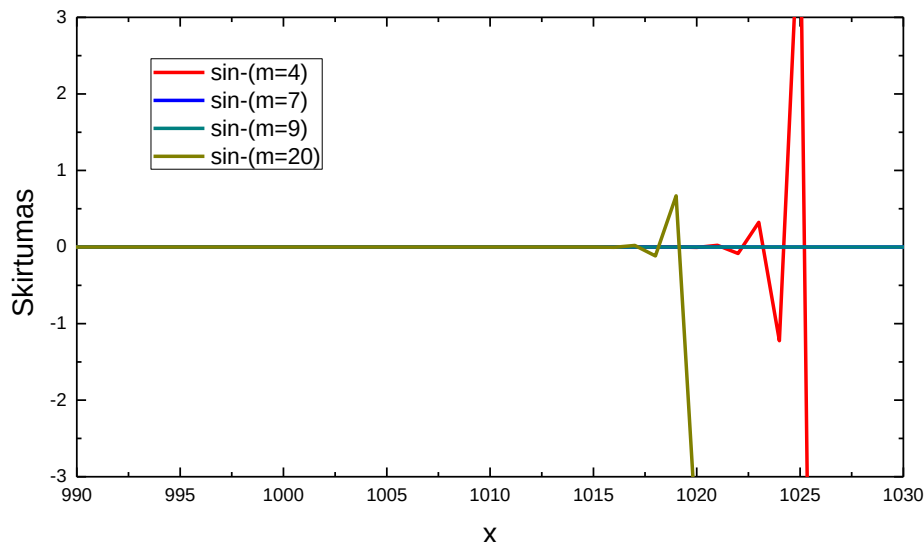
3. Rezultatai ir jų aptarimas

Metodo testavimas

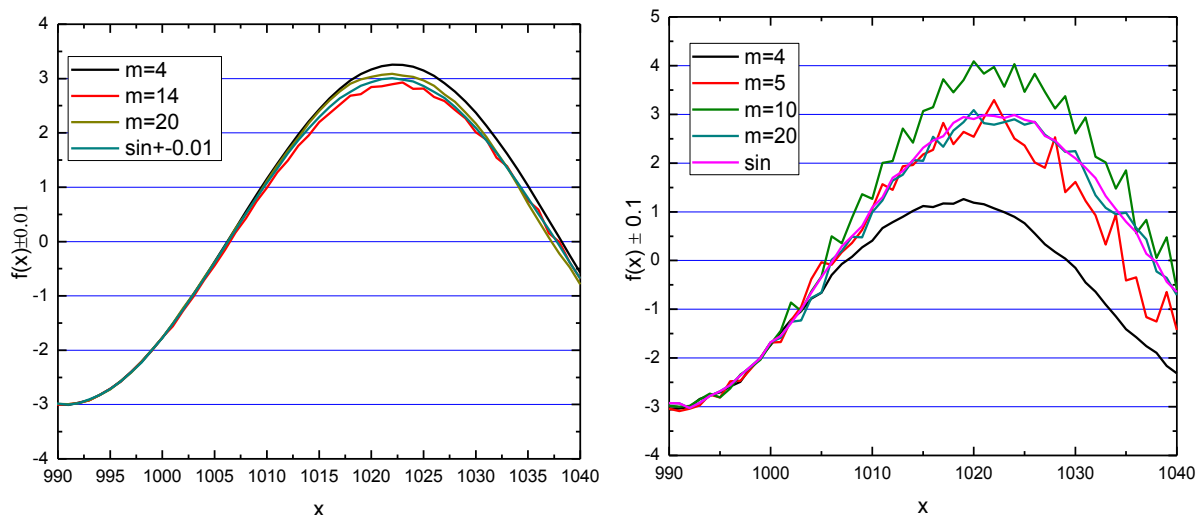
Šiame darbe parašytos programos testavimui buvo naudojamos periodinė $f(x) = 3\sin(x * 0.1)$ ir chaotinė – $g(y_i) = 4y_{i-1}(1 - y_{i-1})$, logistinė funkcija su pradine verte $y_1 = 0.1$, sekos. Testavimas atliktas sugeneruojant 1000 verčių duomenų seką. Fazinės erdvės rekonstrukcijai reikalingos optimalios delsos, nustatytos taikant pirmojo bendrosios informacijos minimumo metodą. Priede pateikta (optimalus delsos nustatymas) programinio kodo dalimi nustatytos funkcijų optimalios delsos, sinuso optimali delsa 16, funkcijos $g(y_i)$ optimali delsa 9. Gauti prognozės rezultatai pavaizduoti 4 – 10 paveiksluose. Prognozės vertės pavaizduotos nuo 1000-osios vertės x ašyje.



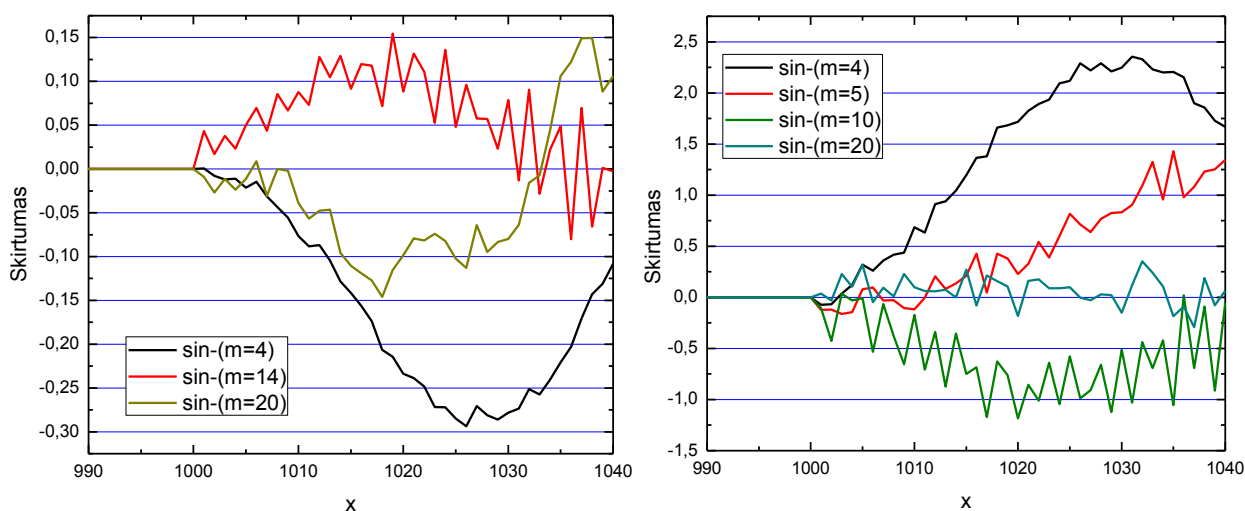
4 pav. Funkcijos $f(x)$ ir jos prognozuojamų verčių priklausomybės nuo x grafikas. Prognozė prasideda nuo 1000-osios vertės x ašyje.



5 pav. Skirtumo tarp funkcijos $f(x)$ ir prognozuojamų verčių priklausomybės nuo x grafikas.

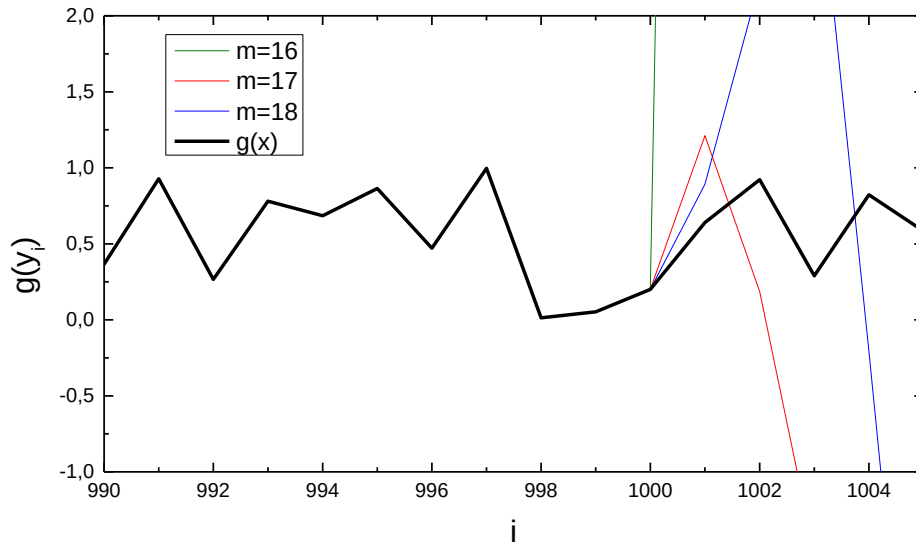


6 pav. Funkcijų $f(x)$ su triukšmu ± 0.01 (kairėje) ir ± 0.1 (dešinėje) prognozuojamų verčių priklausomybės nuo x grafikai. Intarpe pažymėta modeliavime naudotas dimensijų skaičius. Prognozė prasideda nuo 1000-osios vertės x ašyje.

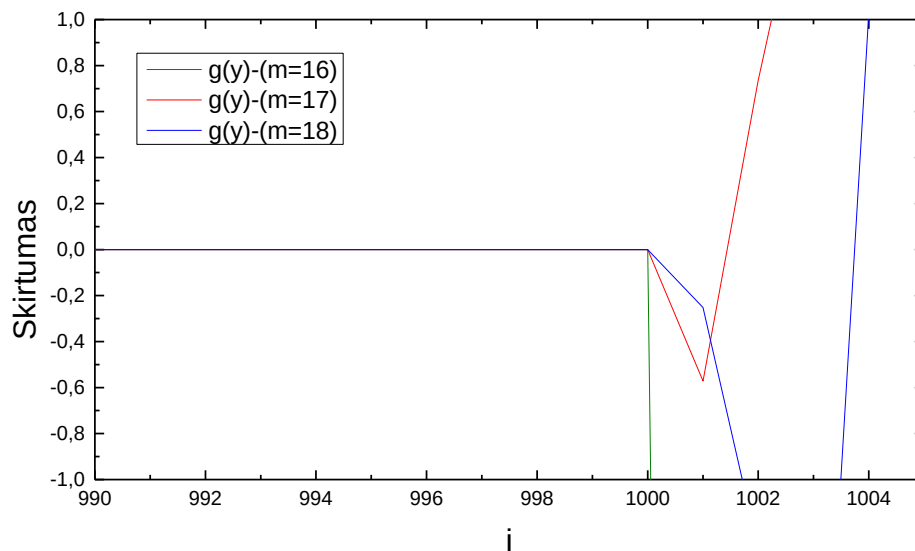


7 pav.. Skirtumo tarp prognozuojamų verčių priklausomybės nuo x grafikas, kai pradinės vertės yra su (kairėje) ± 0.01 ir ± 0.1 (dešinėje) amplitudės triukšmu. Intarpe pažymėta modeliavime naudotas dimensijų skaičius.

Išanalizavus duomenis nustatyta, kad nesant triukšmo teisingai prognozuojamos pirmos 15 verčių, nepriklausomai nuo naudotų dimensijų skaičiaus. Esant dimensijų skaičiui, lygiam $m=7$ ir $m=9$, teisingai prognozuojamos 30 verčių. Kai prie pradinių rezultatų pridedamas triukšmas, (∓ 0.01 ir ∓ 0.1) prognozuojamų verčių tikslumas priklauso nuo dimensijų skaičiaus: didinant dimensijų skaičių prognozuojama vertė yra artimesnė realiai funkcijos vertei.

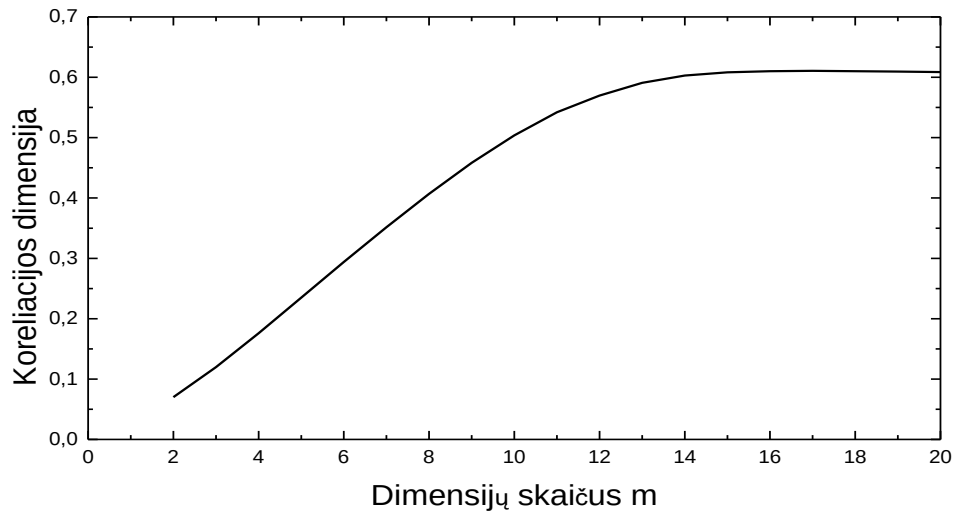


8 pav. Logistinės funkcijos $g(y)$ ir jos prognozuojamų verčių priklausomybės nuo i grafikas. Intarpe pažymėta modeliavime naudota dimensija.



9 pav. Skirtumo tarp logistinės funkcijos $g(y_i)$ ir jos prognozuojamų verčių priklausomybės nuo i grafikas. Intarpe pažymėta modeliavime naudotas dimensijų skaičius.

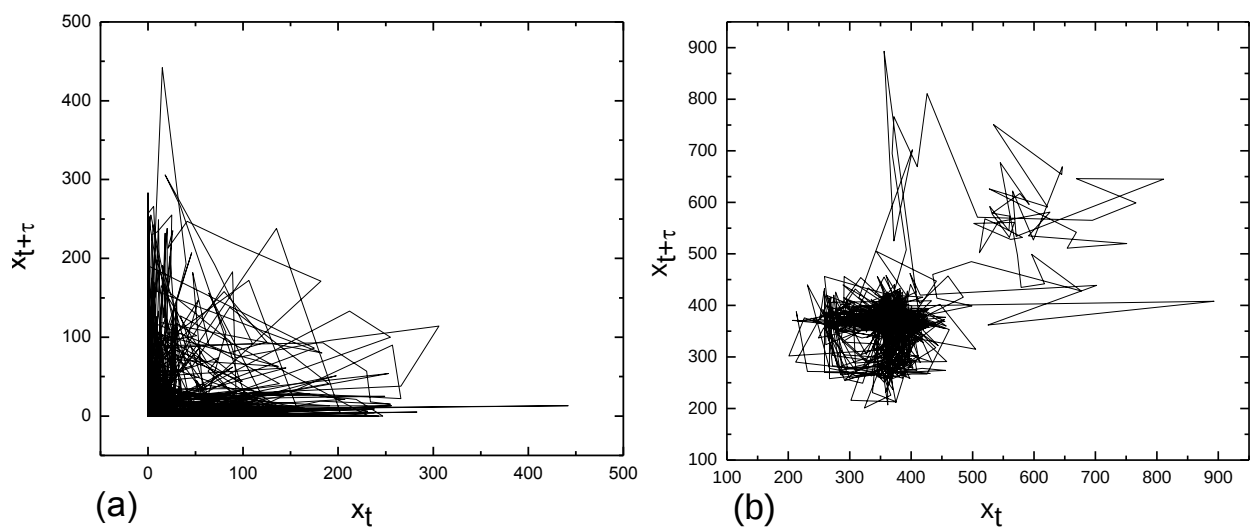
Įvertinus logistinės funkcijos skaičiavimo duomenis nustatyta, kad prognozavimas yra tiksliausias parinkus optimalų dimensijų skaičių $m = 17$, jį nustatome iš koreliacijos dimensijos įsisotinimo vertės (10 pav.). Lyginant su periodine funkcija, padidinus rekonstruotos erdvės dimensijų skaičių, prognozavimo tikslumas nepadidėja, o sumažėja. Todėl, tiriant chaotines sistemas yra svarbu nustatyti optimalų dimensijų skaičių.



10 pav. Logistinės funkcijos $g(y_i)$ koreliacijos dimensijos priklausomybė nuo dimensijų skaičiaus, kai $\tau = 9$.

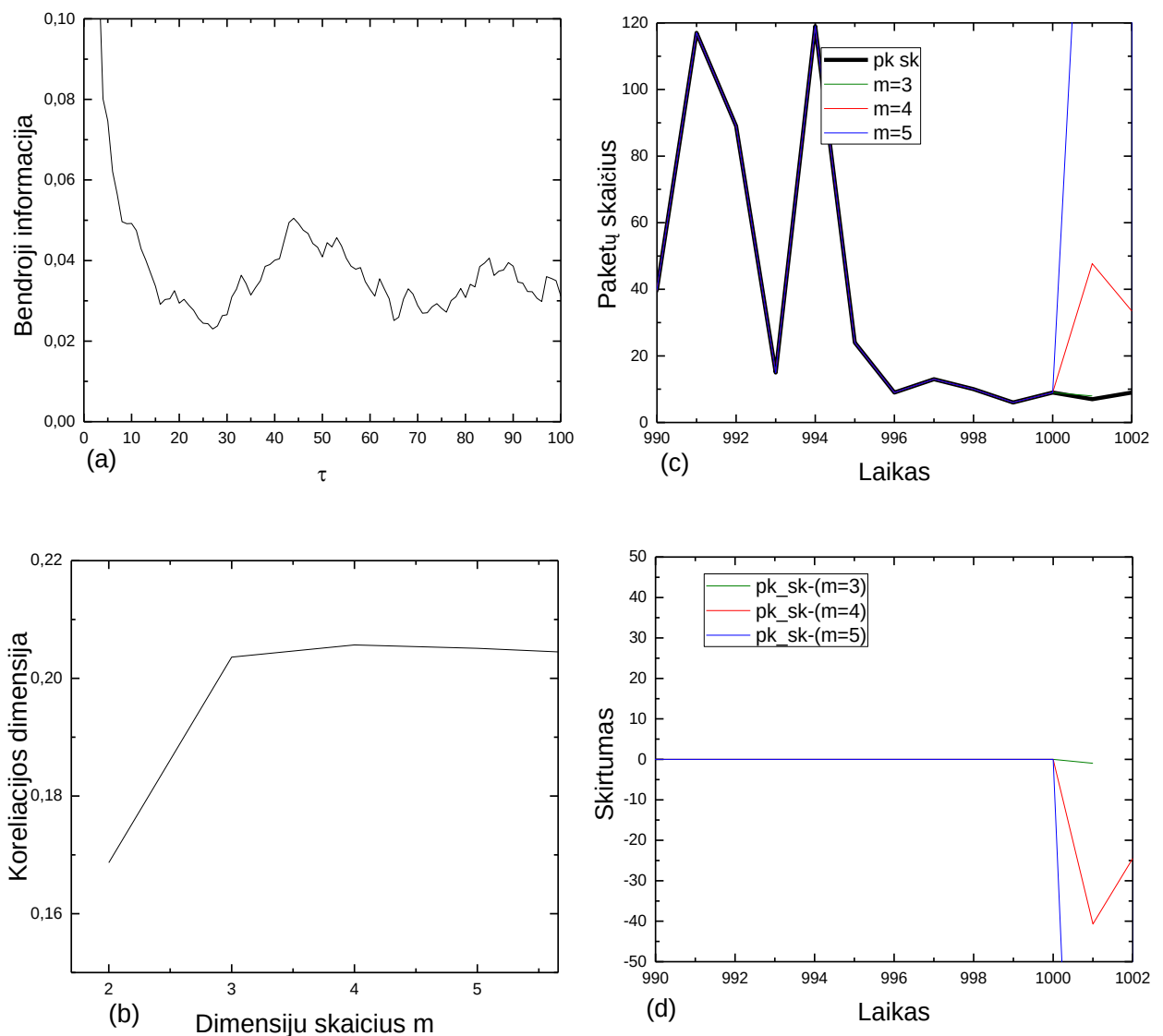
Interneto srauto analizė

Proceso tinkle stabilumo ir apibrėžtumo įvertinimui naudoti duomenys MIT1999 [63] ir CAIDA2007 [54]. Kadangi tokius duomenis teoriškai aprašyti sunku, naudotas laiko eilučių analizės metodas [42]. Šių laiko eilučių fazinės erdvės rekonstrukcijos, kai srauto vertinimui naudojamas paketų skaičius 1 s ilgio intervaluose, grafikai pateikti 11 pav. Analizuojant 11 pav., galime pastebėti, kad laiko eilutės elementai yra koncentruoti apibrėžtuose matavimuose ir ribojami atitinkamo atraktoriaus. Todėl galima teigti, kad tiriamus duomenų srautus galima vertinti, kaip netiesines dinamines sistemas. Atliekant fazinės erdvės rekonstrukciją, svarbūs parametrai yra delsa ir rekonstruotos erdvės dimensijų vertė.



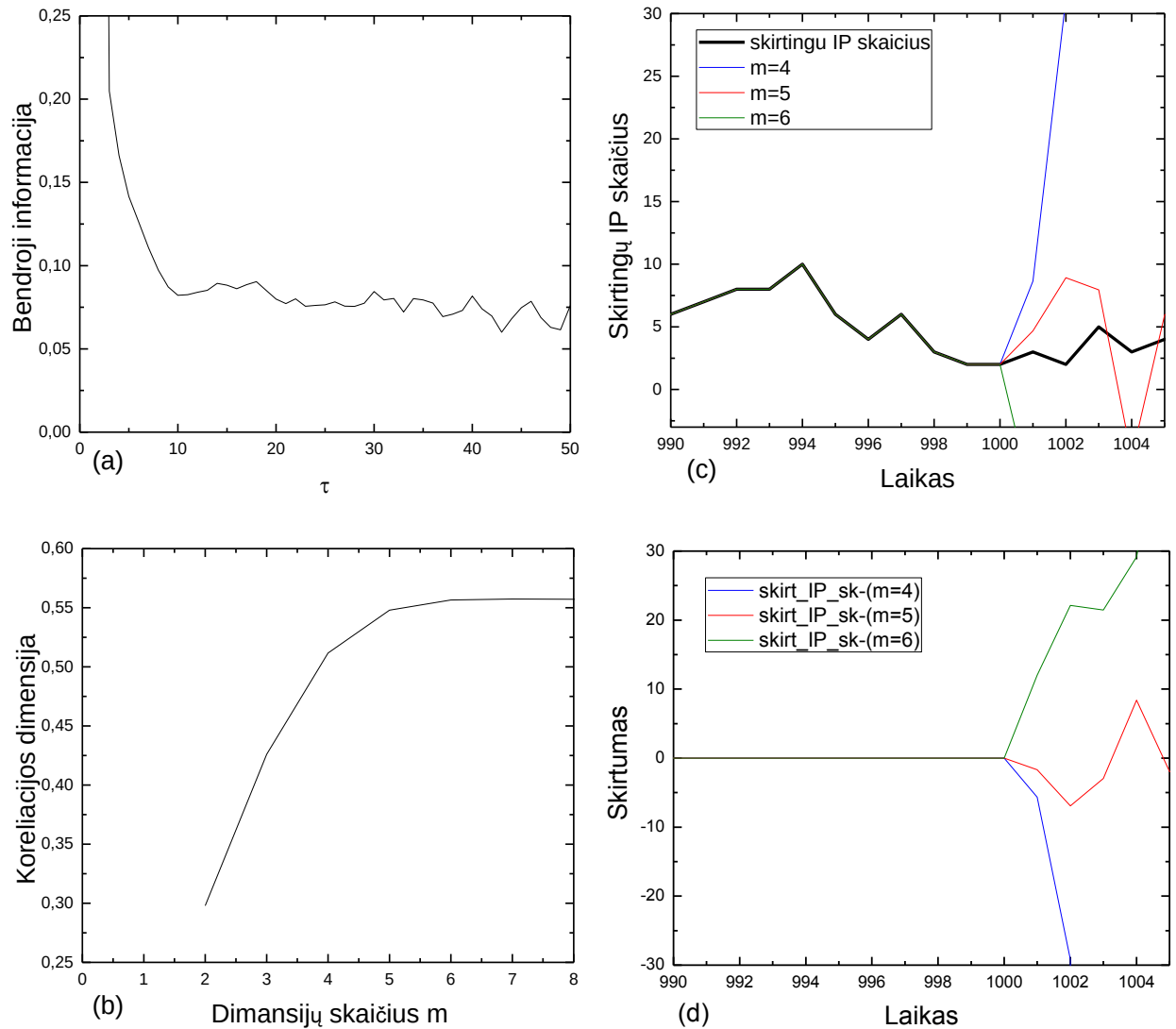
11 pav. MIT1999 (a) ir CAIDA2007 (b) paketų skaičiaus 1 s intervaluose fazinės erdvės rekonstrukcijos grafikai.

Taikant netiesinės dinamikos metodus DDoS atakų aptikimui, remiamasi prielaida, kad galima nustatyti, kaip kis nuo laiko priklausianti sistema net nežinant analizinės modelio išraiškos [33]. Srauto prognozė gali būti atlikta analizuojant laikines eksperimento rezultatų sekas, iš kurių yra sudaroma rekonstruota erdvė, kuri yra ekvivalenti originaliai sistemos būsenos erdvei. Rekonstruotos erdvės sudarymui yra svarbiausi du parametrai τ ir m rekonstruotos erdvės dimensijų skaičius. Nustačius optimalias m ir τ vertes iš laikinės eilutės galima sudaryti rekonstruotą erdvę, ekvivalenčią originaliai, pagal 10 formulę. Tuomet pasinaudojus 13 – 16 formulėmis galima prognozuoti sekančią srauto vertę.



12 pav. Paketų skaičiaus: bendrosios informacijos priklausomybės nuo τ (a), koreliacijos dimensijos nuo dimensijų skaičiaus m , kai $\tau = 27$ (b), prognozuojamų verčių priklausomybė nuo laiko (c), skirtumo tarp prognozuojamos ir realios vertės (d) grafikai. Tarpuose pažymėta modeliavime naudota dimensija.

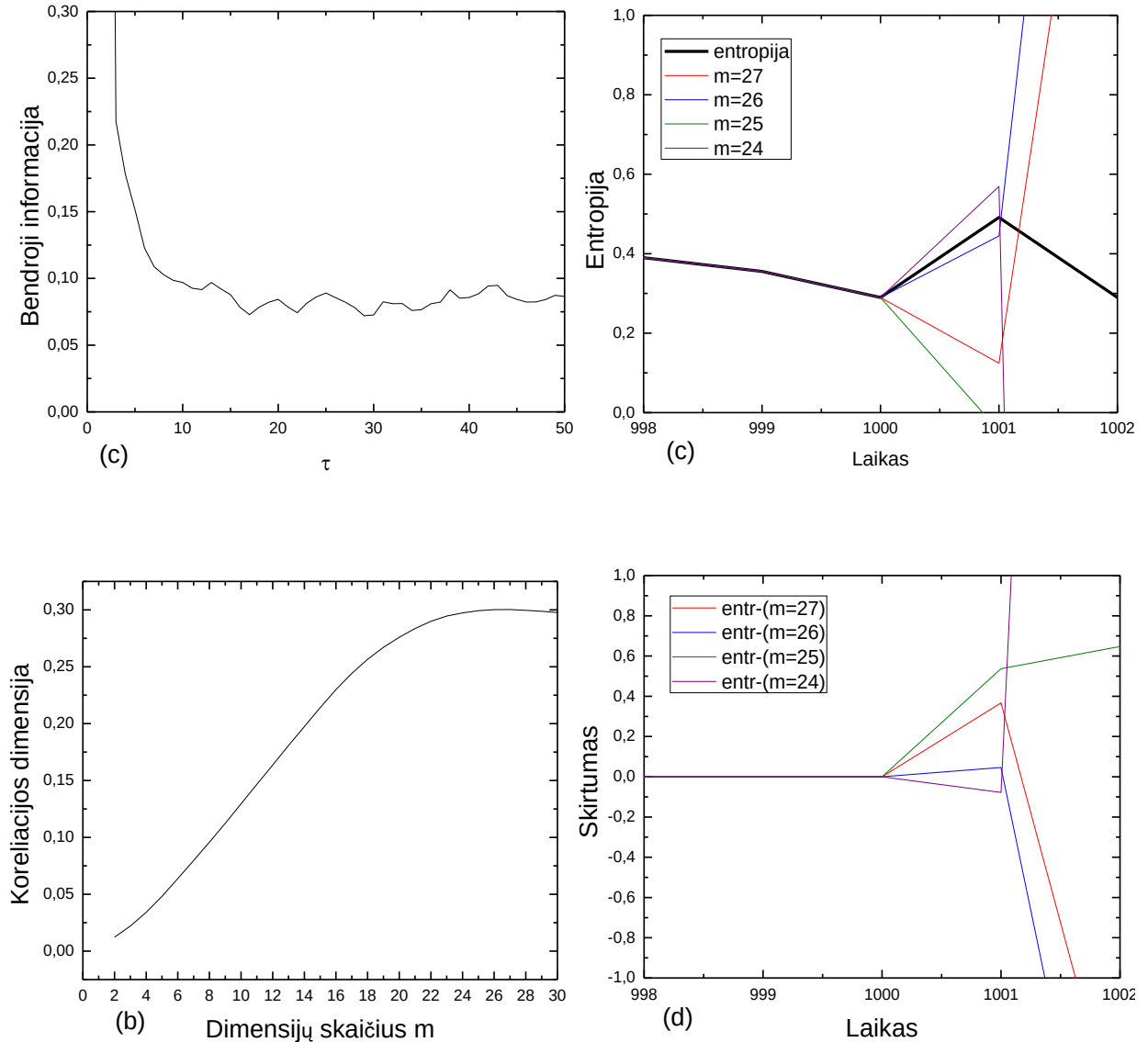
Analizuojant MIT1999 duomenis pagal paketų skaičių 1 s ilgio intervaluose, optimali delsa $\tau = 27$ (12 pav. a) nustatyta naudojant bendrosios informacijos metodą (pagal pirmąjį minimumą). Išanalizavus 12 pav. b grafiką pagal įsisotinimo vertę, nustatyta optimali koreliacinės dimensijos vertė $m = 3$. Atliktos srauto prognozės rezultatai pavaizduoti 12 pav. c nuo 1000-osios vertės x ašyje. Įvertinus 12 pav. d rezultatus, nustatyta, kad skirtumas tarp realios ir pirmos prognozuojamos vertės yra -0,97 (13%), ($m=3$) ir -40,69 (581%), ($m=4$).



13 pav. Skirtingų IP adresų skaičiaus: bendrosios informacijos priklausomybės nuo τ (a), koreliacijos dimensijos nuo dimansijų skaičiaus m , kai $\tau = 10$ (b), prognozuojamų verčių priklausomybė nuo laiko (c), skirtumo tarp prognozuojamos ir realios vertės (d) grafikai. Intarpuose pažymėta modeliavime naudota dimensija.

Analizuojant duomenis, pagal skirtingų šaltinio IP adresų skaičių 1 s ilgio intervaluose, nustatyta optimali delsa $\tau = 10$ (12 pav. a) ir optimali koreliacijos dimensijos vertė $m = 5$ (12 pav. b). Srauto prognozės vertės pavaizduotos 12 pav. c. Išanalizavus 12 pav. d nustatyta, kad pirma

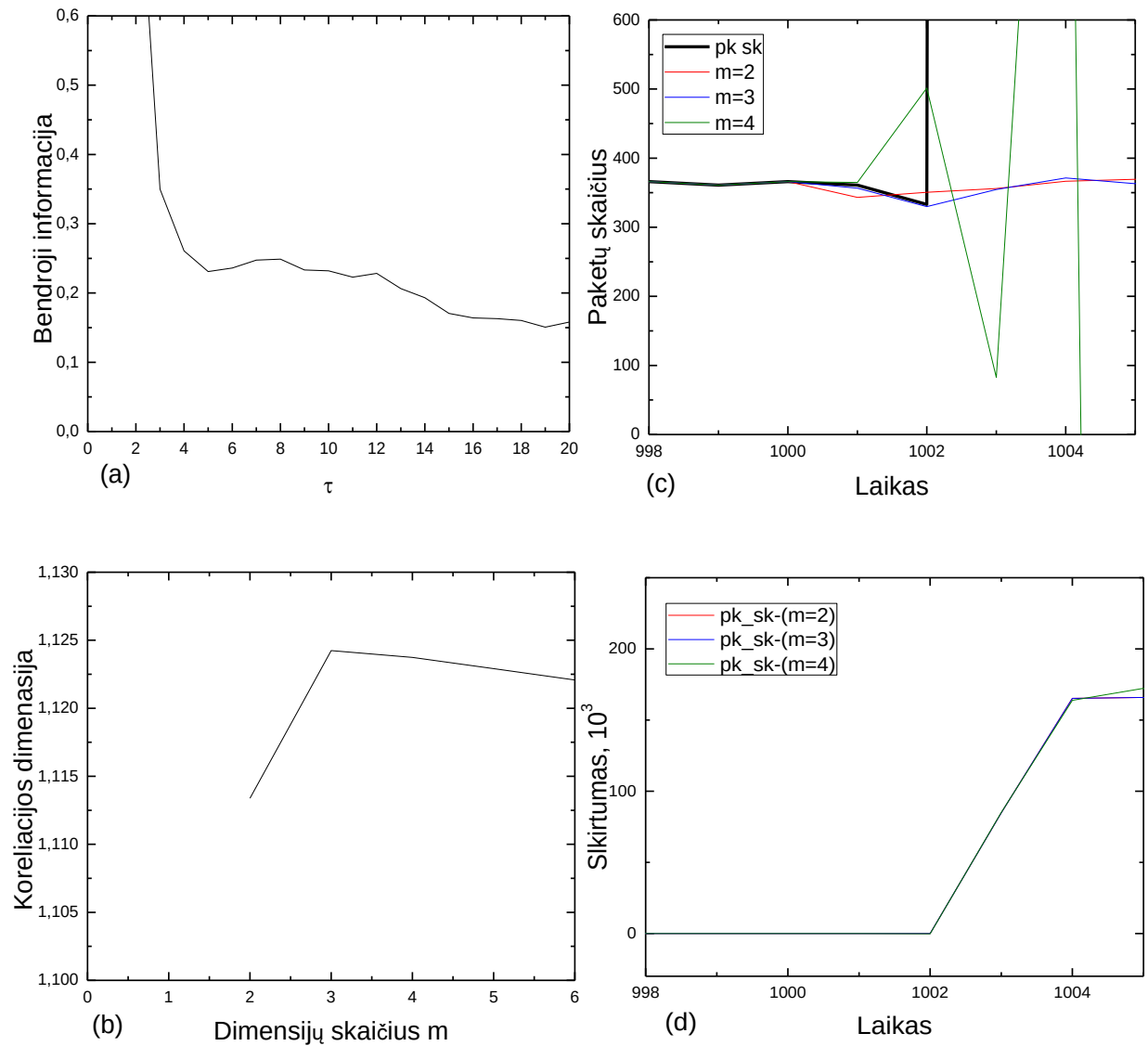
prognozuojama vertė nuo realios vertės skiriasi -5,65 (188%), ($m = 4$), -1,68 (56%), ($m = 5$) ir 12 (400%), ($m = 6$).



14 pav. Entropijos: bendrosios informacijos priklausomybės nuo τ (a), koreliacijos dimensijos nuo dimensijų skaičiaus m , kai $\tau = 17$ (b), prognozuojamų verčių priklausomybė nuo laiko (c), skirtumo tarp prognozuojamos ir realios vertės (d) grafikai. Intarpuose pažymėta modeliavime naudota dimensija.

Analizuojant duomenis, pagal šaltinio IP adreso entropiją 1 s ilgio intervaluose, nustatytos optimalios $\tau = 17$ ir $m = 26$ vertės rekonstruotos erdvės sudarymui (14 pav. a, b). Tinkamai parinkus rekonstruotos erdvės dimensijų skaičių, galima apytiksliai prognozuoti vieną vertę į priekį (14 pav. c). Išanalizavus 14 pav. d grafiką nustatyta, kad šaltinio IP adresų entropijos skirtumas tarp realios vertės ir pirmos prognozuojamos vertės yra -0,08 (16%), ($m = 24$), 0,54 (109%), ($m = 25$), 0,05 (9%), ($m = 26$) ir 0,37 (75%), ($m = 27$).

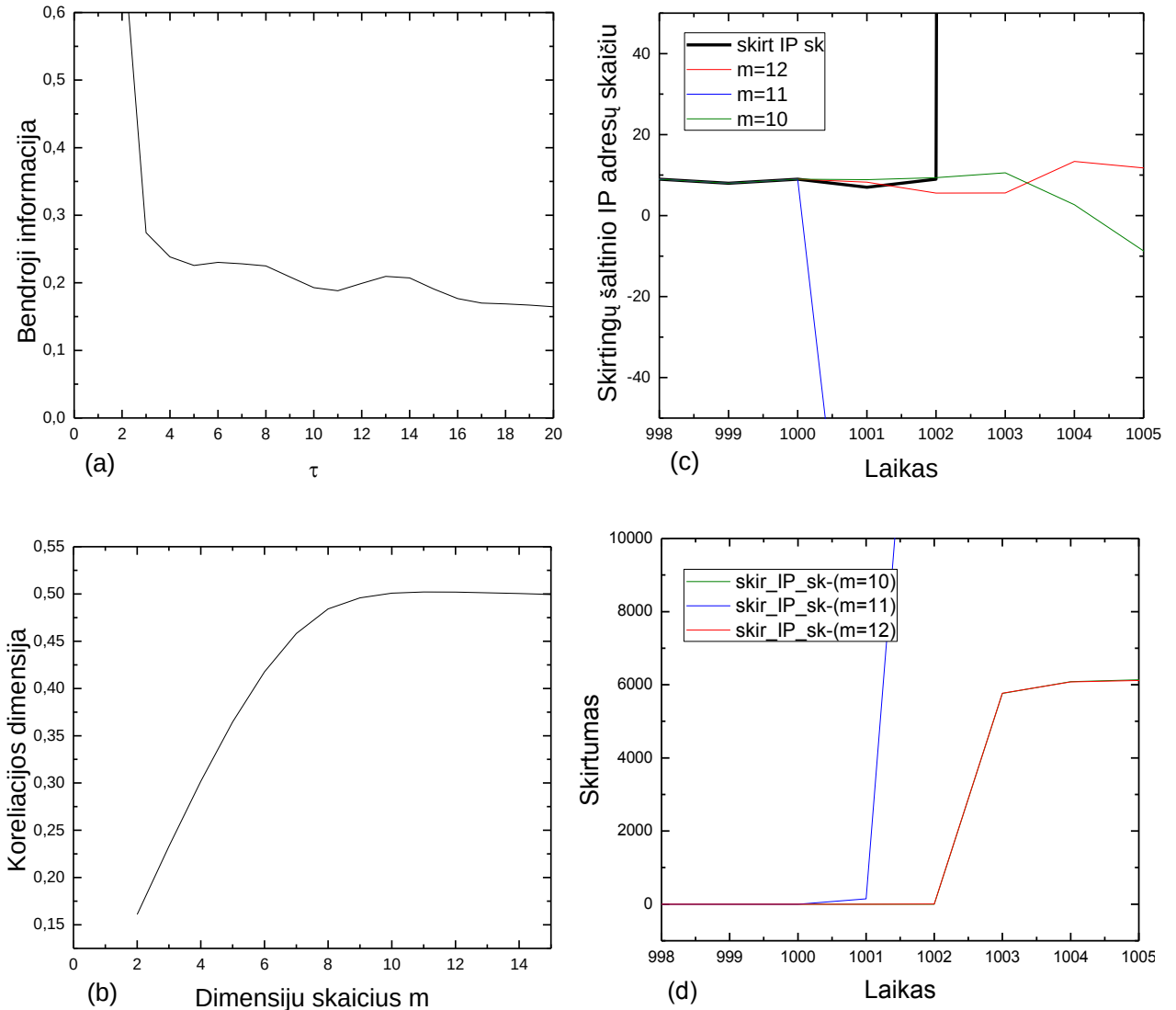
Siekiant geriau išanalizuoti netiesinės dinamikos metodą, skaičiavimai buvo pakartoti su CAIDA2007 duomenų failu, kuriame užfiksuotas srautas yra daug intensyvesnis, lyginant su MIT1999 duomenų failu. Programa sudaro 1000 verčių seką iš CAIDA2007 duomenų srauto iki atakos pradžios likus 2 sekundėms. Rezultatai pavaizduoti 15 – 17 pav. Srauto prognozė prasideda nuo 1000-osios vertės x ašyje (15-17 pav. c).



15 pav. Paketų skaičiaus: bendrosios informacijos priklausomybės nuo τ (a), koreliacijos dimensijos priklausomybės nuo dimensijų skaičiaus m , kai $\tau = 5$ (b), prognozuojamų verčių priklausomybės nuo laiko (c), skirtumo tarp prognozuojamos ir realios vertės (d) grafikai. Interpuose pažymėta modeliavime naudota dimensija.

Analizuojant CAIDA2007 duomenis, pagal paketų skaičių 1 s ilgio intervaluose optimali delsa $\tau = 5$ (15 pav. a), nustatyta naudojant bendrosios informacijos metodą (pagal pirmąjį minimumą). Įvertinus 15 pav. b grafiką parinkta optimali koreliacijos dimensijos vertė $m = 3$.

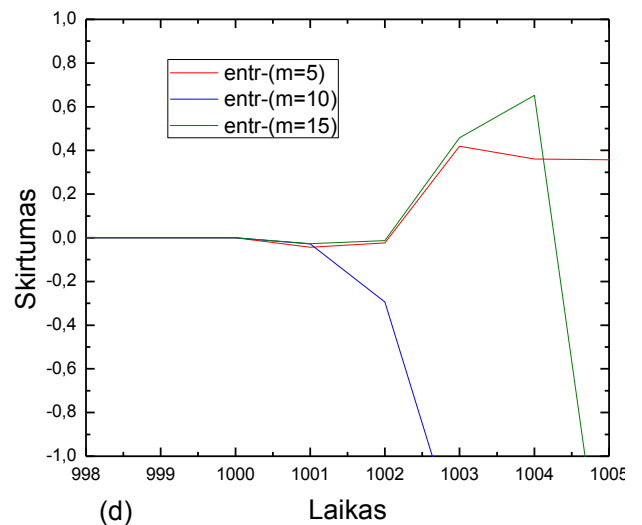
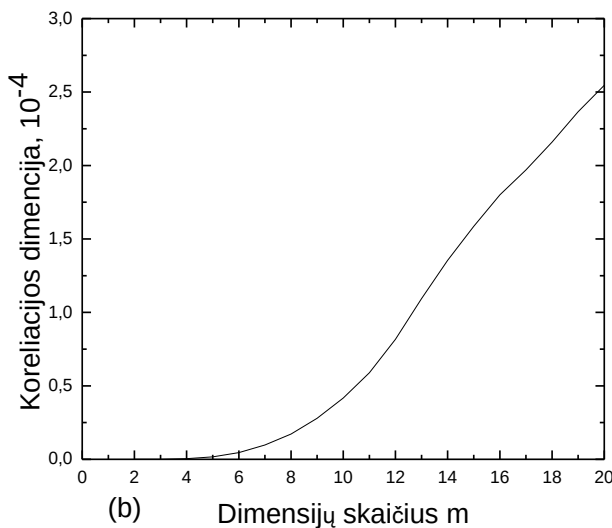
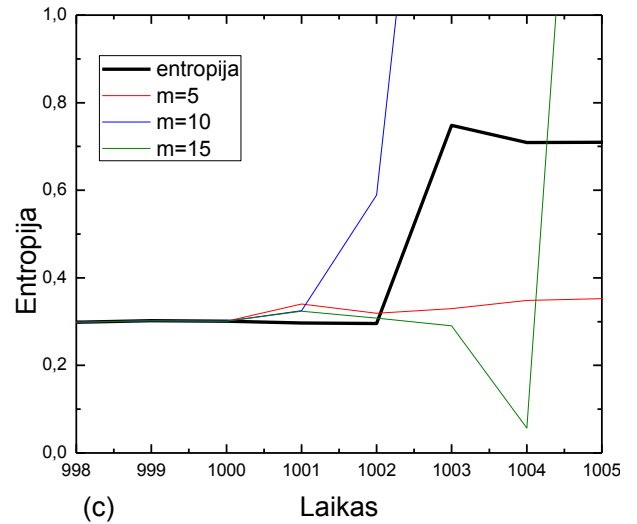
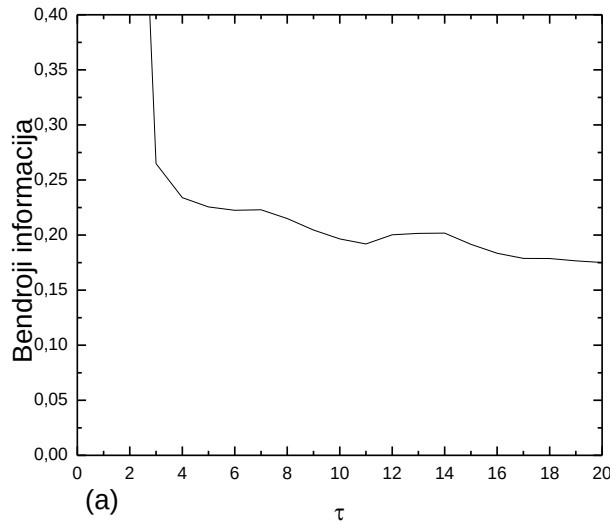
Srauto prognozės rezultatai parinkus optimalius parametrus pavaizduoti 15 pav. c grafike. Išanalizavus skirtumo tarp realios ir prognozuojamos vertės grafiką (15 pav. d) nustatyta, kad iki atakos pradžios (laikas 1002) skirtumas tarp prognozuojamos vertės ir realios vertės kinta nuo -17 iki 17 (5%), kai $m = 2$ ir nuo 3 iki 5 (1%), kai $m = 3$. Prasidėjus atakai skirtumas išauga ir peržengia rėžius.



16 pav. Skirtingų IP adresų skaičiaus: bendrosios informacijos priklausomybės nuo τ (a), koreliacijos dimensijos priklausomybė nuo dimensijų skaičiaus m , kai $\tau = 11$ (b), prognozuojamų verčių priklausomybė nuo laiko (c), skirtumo tarp prognozuojamos ir realios vertės (d) grafikai. Intarpuose pažymėta modeliavime naudota dimensija.

Analizuojant CAIDA2007 duomenis pagal skirtingų šaltinio IP adresų skaičių 1 s ilgio intervaluose nustatyti optimalūs rekonstruotos erdvės parametrai $\tau = 11, m = 12$, grafikai atvaizduoti 16 pav. a, b. Srauto prognozės rezultatai pavaizduoti 16 pav. c nuo 1000-osios vertės x ašyje. Išanalizavus skirtumo tarp realios ir prognozuojamos vertės grafiką (16 pav. d) nustatyta, kad

iki atakos pradžios (laikas 1002) skirtumas tarp spėjamos vertės ir realios vertės kinta intervale - 1,25 – 3,43 (18% – 38%), ($m = 12$) ir -1,88 – -0,37 (4% – 27%), ($m = 10$). Prasidėjus atakai išauga ir peržengia režius.



18 pav. Entropijos: bendrosios informacijos priklausomybės nuo τ (a), koreliacijos dimensijos priklausomybė nuo dimensijų skaičiaus m , kai $\tau = 11$ (b), prognozuojamų verčių priklausomybė nuo laiko (c), skirtumo tarp prognozuojamos ir realios vertės (d) grafikai. Tarpuose pažymėta modeliavime naudota dimensija.

Atlikus CAIDA2007 duomenų analizę nustatytos šios optimalios laiko delsos ir rekonstruotos erdvės dimensijų skaičiaus vertės $\tau = 11, m = 5$, kai srautas analizuojamas pagal šaltinio IP adreso entropijos vertę 1 s ilgio intervaluose (17 pav. a, b). Atlikus verčių spėjimą su optimaliomis delsos ir rekonstruotos erdvės dimensijų vertėmis, galime pastebėti, kad pirmos dvi prognozuojamos vertės yra artimos realioms vertėms (17 pav c). Skirtumo vertės tarp realios ir

prognozuojamos vertės pagal entropijos atributą iki atakos pradžios (laikas 1002) svyruoja intervale $-0,04 - 0,02$ (8% – 15%), ($m = 5$) (4% – 9%), ($m = 15$), o prasidėjus atakai išauga ir svyruoja intervale $0,35 - 0,47$ (49% – 56%), kai ($m = 5$) (17 pav. d).

Įvertinus tyrimo duomenis, galima teigti, kad su MIT1999 duomenimis galime prognozuoti 1 vertę su 9% – 56% paklaida, parinkus optimalias delsos ir rekonstruotos erdvės vertes, priklausomai nuo skaičiavimams naudojamo atributo. Su CAIDA2007 duomenimis galime prognozuoti apie 5 vertes su 1% – 27% paklaida, priklausomai nuo naudojamo atributo, nustatčius optimalias τ ir m vertes.

Pagrindiniai rezultatai ir išvados

1. Darbo metu išanalizuotos netiesinės dinamikos metodų taikymo galimybės, nagrinėjant duomenų srautus kompiuterių tinkluose.
2. Parašyta ir su modelinėmis funkcijomis ištestuota programa statistinei chaotinės sistemos analizei.
3. Ištyrus MIT1999 ir CAIDA2007 srauto duomenis nustatyta, kad juose užfiksuoto tinklo srauto laikinių sekų rekonstruotos fazinės erdvės diagramos konverguoja link baigtinio (nors ir neapibrėžto atraktoriaus), turi apibrėžtą koreliacinę dimensiją, todėl sistemą galima vertinti kaip netiesinės dinamikos procesą.
4. Išanalizavus skaičiavimų rezultatus nustatyta, kad parinkus optimalią delką ir rekonstruotos erdvės dimensijų skaičių su 1% – 27% paklaida, galima prognozuoti 5 vertes analizuojant CAIDA2007 duomenis ir 1 vieną vertę su 9% – 56% paklaida analizuojant MIT1999 duomenis. Remiantis tyrimo rezultatais, galime teigti, kad reikalingas papildomas netiesinės dinamikos metodo taikymo, DDoS atakoms aptikti, tyrimas, siekiant gauti tikslesnius rezultatus.

Literatūros šaltinių sąrašas

- [1] SHUI YU, Distributed Denial of Service Attack and Defence, (2013).
- [2] ims.mii.lt/ALKŽ/, (2015-12-20).
- [3] <https://securelist.com/analysis/quarterly-malware-reports/72560/kaspersky-ddos-intelligence-report-q3-2015/>, (2015-11-15).
- [4] <http://www.lietuviuzodynas.lt/anglu-lietuviu-zodynas/Botnet>, (2016-01-15).
- [5] http://www.lietuviuzodynas.lt/anglu-lietuviu-zodynas/Command_control, (2016-01-16).
- [6] <http://searchsecurity.techtarget.com/definition/SYN-flooding>
- [7] <https://enotepad.wordpress.com/2008/08/27/tcp-connection-establishment/>
- [8] <https://www.incapsula.com/ddos/ddos-attacks/>
- [9] Y. Chen and K. Hwang, “Collaborative detection and filtering of shrew ddos attacks using spectral analysis” *Journal of Parallel Distributed Computing*, vol. 66, no. 9, pp. 1137–1151, (2006).
- [10] H.Wang, C. Jin, and K. G. Shin, “Defense against spoofed ip traffic using hop-count filtering” *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 40–53, 2007.
- [11] B. Al-Duwairi and G. Manimaran, “Novel hybrid schemes employing packet marking and logging for ip traceback” *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 5, pp. 403–418, (2006).
- [12] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao, “Packetscore: A statistics-based packet filtering scheme against distributed denial-of-service attacks” *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 2, pp. 141–155, (2006).
- [13] R. R. Kompella, S. Singh, and G. Varghese, “On scalable attack detection in the network,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 14–25, (2007).
- [14] P. E. Ayres, H. Sun, H. J. Chao, and W. C. Lau, “Alpi: A ddos defense system for high-speed networks” *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1864– 1876, (2006).
- [15] Y. Xiong, S. Liu, and P. Sun, “On the defense of the distributed denial of service attacks: an on-off feedback control approach” *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 31, no. 4, pp. 282–293, (2001).
- [16] R. Chen, J.-M. Park, and R. Marchany, “A divide-and-conquer strategy for thwarting distributed denial-of-service attacks” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 5, pp. 577–588, (2007).

- [17] A. Yaar, A. Perrig, and D. Song, "Stackpi: New packet marking and filtering mechanisms for ddos and ip spoofing defense" *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1853–1863, (2006).
- [18] A. Bremler-Barr and H. Levy, "Spoofing prevention method", *Proceedings of INFOCOM*, pp. 536–547, (2005).
- [19] H. Aljifri, "Ip traceback: A new denial-of-service deterrent?" *IEEE Security and Privacy*, vol. 1, pp. 24–31, 2003.
- [20] A. Yaar, A. Perrig, and D. X. Song, "Fit: fast internet traceback" in *Proceedings of the INFOCOM*, pp. 1395–1406, (2004).
- [21] M. Sung, J. Xu, J. Li, and L. Li, "Large-scale ip traceback in high-speed internet: practical techniques and information-theoretic foundation" *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1253–1266, (2008).
- [22] L. Feinstein and D. Schnackenberg, "Statistical approaches to ddos attack detection and response", *Proceedings of the DARPA Information Survivability Conference and Exposition*, pp. 303–314, (2003).
- [23] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denialof- service activity" *ACM Transactions on Computer Systems*, vol. 24, no. 2, pp. 115–139, (2006).
- [24] A. El-Atawy, E. Al-Shaer, T. Tran, and R. Boutaba, "Adaptive early packet filtering for defending firewalls against dos attacks", *Proceedings of INFOCOM*, pp. 2437–2445, (2009).
- [25] F. Soldo, A. Markopoulou, and K. J. Argyraki, "Optimal filtering of source address prefixes:Models and algorithms", *Proceedings of INFOCOM*, pp. 2446–2454, (2009).
- [26] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies", *Proceedings of the Internet Measurement Workshop*, pp. 71–82, (2002).
- [27] J. Tang and Y. Cheng, "Quick detection of stealthy sip flooding attacks in voip networks", *Proceedings of the IEEE ICC*, pp. 1–5, (2011).
- [28] Mohammed Alenezi and Martin J Reed „Methodologies for detecting DoS/DDoS attacks against network servers“ *ICSNC: The Seventh International Conference on Systems and Networks Communications*, (2012).
- [29] N. Ye, *Secure computer and network systems: modeling, analysis and design*: Wiley, 2008.
- [30] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges" *computers & security*, vol. 28, pp. 18-28, (2009).

- [31] W. Liu, P. P. Pokharel, and J. C. Principe, “Correntropy: Properties and applications in nongaussian signal processing”, *IEEE Transactions on Signal Processing*, vol. 55, no. 11, pp. 5286–5298, (2007).
- [32] <https://lt.wikipedia.org/wiki/Entropija>
- [33] Elizabeth Bradley, Holger Kantz, Nonlinear time-series analysis revisited, *Chaos* 25, 097610-1-10, (2015); <http://dx.doi.org/10.1063/1.4917289/>
- [34] Stane Kodba, Matjaž Perc and Marko Marhl, Detecting chaos from a time series, *Eur. J. Phys.* 26 205–215 (2005) doi:10.1088/0143-0807/26/1/021
- [35] J.D. Farmer, J.J. Sidorowich, *Phys. Rev. Lett.* 59 845, (1987).
- [36] G. Sugihara, R.M. May, *Nature* 334 734, (1990).
- [37] M. Giona, F. Lentini, V. Cimagalli, *Phys. Rev. A* 44 3496, (1991).
- [38] D. Kugiumtzis, O.C. Lingjarde, N. Christophersen, *Physica D* 112 344, (1998).
- [39] A.W. Jayawardena, W.K. Li, P. Xu, *J. Hydro.* 258 4,, (2002).
- [40] M. Ragwitz, H. Kantz, *Phys. Rev. E* 65 056201, (2002).
- [41] H. Kantz, T. Schreiber, *Nonlinear Time Series Analysis*, second ed., *Cambridge Univ. Press, Cambridge*, (2003).
- [42] Qingfang Meng *, Yuhua Peng, A new local linear prediction model for chaotic time series, *School of Information Science and Engineering, Shandong University, Jinan 250100, PR China*, (2006).
- [43] N. Packard, et al., *Phys. Rev. Lett.* 45 712, (1980).
- [44] F. Takens, in: D. Rand, L.-S. Young (Eds.), *Dynamical Systems and Turbulence*, *Springer, Berlin*, , p. 366, (1981).
- [45] T. Sauer, J.A. Yorke, M. Casdagli, *J. Stat. Phys.* 65 579, (1991).
- [46] <https://lt.wikipedia.org/wiki/Atraktorius> (2016).
- [47] Anna Krakovská, Kristína Mezeiová, and Hana Budáčová, Use of False Nearest Neighbours for Selecting Variables and Embedding Parameters for State Space Reconstruction, *Journal of Complex Systems*, (2015).
- [48] Ai-Hua Jiang, Xiu-ChangHuang, Zhen-HuaZhang, JunLi, Zhi-Yi Zhang, Hong-XinHua, Mutual information algorithms, *Mechanical Systems and Signal Processing* 24 2947–2960, (2010).
- [49] <https://lt.wikipedia.org/wiki/Koreliaciija> (2016).
- [50] H. Whitney, “Differentiable manifolds,” *The Annals of Mathematics*, vol. 37, no. 3, pp. 645–680, (1936).

- [51] M. B. Kennel, R. Brown, and H. D. I. Abarbanel, “Determining embedding dimension for phase-space reconstruction using a geometrical construction,” *Physical Review A*, vol. 45, no. 6, pp. 3403–3411, (1992).
- [52] L. Cao, “Practical method for determining the minimum embedding dimension of a scalar time series,” *Physica D: Nonlinear Phenomena*, vol. 110, no. 1-2, pp. 43–50, (1997).
- [53] <https://www.ll.mit.edu/ideval/data/index.html>, (2016-01-16).
- [54] <http://www.caida.org/home/>, (2016-01-13).
- [55] <https://www.ll.mit.edu/ideval/data/1999data.html>, (2016-01-13).
- [56] <https://data.caida.org/datasets/security/ddos-20070804/>, (2016-01-13).
- [57] <https://www.ll.mit.edu/ideval/data/1999/training/week1/index.html>, (2016-01-13).
- [58] <https://www.ll.mit.edu/ideval/data/1999/training/week2/index.html>, (2016-01-13).
- [59] <https://www.ll.mit.edu/ideval/data/1999/training/week3/index.html>, (2016-01-13).
- [60] <https://www.ll.mit.edu/ideval/data/1999/training/week4/index.html>, (2016-01-13).
- [61] <https://www.ll.mit.edu/ideval/data/1999/training/week5/index.html>, (2016-01-13).
- [62] <https://www.ll.mit.edu/ideval/files/master-listfile-condensed.txt>, (2016-01-13).
- [63] <https://www.ll.mit.edu/ideval/data/1999/testing/week1/monday/inside.tcpdump.gz>, (2016-01-13).

Summary

Andrius Lengvinas

APPLICATION OF NONLINEAR DYNAMICS TO DDoS ATTACK DETECTION

DoS (Denial of service) and DDoS (Distributed Denial of Service) cybernetic attacks are an increasing threat to our security. The main objective of these attacks is to disrupt effectiveness of the net or hardware. During the attack the target device is affected by high intensity of DDoS attack flow and cannot provide its intended service. **In this work application of nonlinear dynamic method to detect DDoS attacks is investigated.** For the analysis, a programme was developed. The purpose of this programme is to calculate necessary values needed for state phase reconstruction and predict future evolution of traffic. The programme was tested using periodic (sin) and chaotic (logistic) functions. Internet traffic data was taken from MIT (Massachusetts Institute of Technology) Linkolno laboratorijos (Lincoln Laboratory) and CAIDA (Center for Applied Internet Data Analysis). Calculations were executed using analysis of the traffic according to these characteristics number of packets, number of different source IP addresses and source IP entropy. **After conducting data analysis it is determined that using MIT1999 data it is possible to approximately determine 1 future value, whereas using CAIDA2007 data it is possible to approximately determine multiple values. It is assumed that further investigations are necessary.**

Santrauka

Andrius Lengvinas

NETIESINĖS DINAMIKOS METODŲ TAIKYMAS DDoS ATAKŲ APTIKIMUI

DoS (Denial of service) ir DDoS (Distributed Denial of Service) kibernetinės atakos kelia vis didesnį pavojų saugumui. Šiomis atakomis siekiama sutrikdyti stabilų tinklo ar įrangos veikimą. Atakos metu auka apkraunama intensyviu paketų srautu, kas ir priverčia ją išnaudoti savo resursus, t. y. procesoriaus ciklus, atmintį ar interneto pralaidą, todėl įranga labai sulėtėja arba tampa neįmanoma ja naudotis. Darbe tiriamas netiesinės dinamikos metodo pritaikymo galimybės, aptikti DDoS atakas. Tyrimui atlikti parašyta programa, skirta: optimalių parametrų reikalingų rekonstruotos erdvės sudarymui, nustatymui bei srauto spėjimui atlikti. Programa ištestuota naudojant periodinę (sin) ir chaotinę (logistinę) funkcijas. Tyrimui naudoti interneto srauto duomenys iš MIT (Massachusetts Institute of Technology) Linkolno laboratorijos (Lincoln Laboratory) ir CAIDA (Center for Applied Internet Data Analysis) Kalifornijos universiteto tyrimų skyriaus. Tyrimo metu skaičiavimai atlikti analizuojant srautą pagal šiuos atributus: paketų skaičių, skirtingų IP adresų skaičių, šaltinio IP adreso entropiją. Atlikus rezultatų analizę nustatyta, kad naudojant MIT1999 duomenis galima apytiksliai prognozuoti 1 vertę, o CAIDA2007 duomenis – kelias vertes. Todėl reikalingas tolimesnis netiesinės dinamikos metodo tyrimas.

Priedas

1. Programos pagrindinė klasė.

```
package bakalauras;

/**
 *
 * @author Faustas
 */
public class Bakalauras {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws Exception {
        // TODO code application logic here
        System.out.println("Hello");

        String DB_table_name = "packets";
        String DBname = "postgres";

        FazineErdve fazE = new FazineErdve();
        Duomenys duom = new Duomenys();
        SrautoNuspejimas spejimas = new SrautoNuspejimas();
        DimensijuNustatymas dimmensija = new DimensijuNustatymas();
        DelsosNustatymas tau = new DelsosNustatymas();

        int diena = 2007;
        int outIntervals = 1000;
        double intervalas = 1;
        int pradziosLaikas = 17;
        int paprastosKorVidurkinamiTaskai = 7;
        int delta = 0;
        int m = 13, spejamuVerciuSk = 40;
        double SINtankis = 0.1;
        int patikra = 1;
        //////////////////////////////////////
        double[] pirmaEilute=null;

        int duomenuTipas = 0;
        int norimasVeiksmas = 0;
        switch (duomenuTipas) {
            case 1:
                pirmaEilute = duom.funkcijosSinVertes(outIntervals, SINtankis);
                break;
            case 2:
                pirmaEilute = duom.netisinesFunkcijosVertes(outIntervals);
                break;
            case 3:
```

```

        pirmaEilute = duom.paketuSkaicius(DBname, DB_table_name, intervalas,
outIntervals, -1, diena, pradziosLaikas);
        break;
    case 4:
        pirmaEilute = duom.skirtinguIpAdresuSkaicius(DBname, DB_table_name,
"source", intervalas, outIntervals, 25, diena, pradziosLaikas);
        break;
    case 5:
        pirmaEilute = duom.entropija(DBname, DB_table_name, "source", intervalas,
outIntervals, 1, diena, pradziosLaikas);
        break;
    default:
        System.out.println("Nepasinkti jokie duomenys!!!");
        System.exit(2);
    }
    switch (norimasVeiksmas) {
        case 11: tau.mutual(pirmaEilute, outIntervals, 0, SINtankis); break;
        case 12: tau.koreliacijaTikra(pirmaEilute, outIntervals); break;
        case 13: fazE.fzineErdve(pirmaEilute, outIntervals, delta); break;
        case 14: dimmensija.embeddingDimencijuSkaicius(pirmaEilute, delta); break;
        case 15: spejimas.verciuSpejimas(pirmaEilute, m, delta, outIntervals,
spejamuVerciuSk, SINtankis); break;
        default: System.out.println("Nepasinktas veiksmas!!!");
    }
}

```

2. Duomenų nuskaitymo klasė

```

package bakalauras;

import java.io.PrintWriter;
import java.sql.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import static java.lang.Math.log;

/**
 * Created by Faustas on 2016-05-18.
 */
public class Duomenys {
    public double[] funkcijosSinVertes(int outIntervals, double tankis) {
        double[] pirmaEilute = new double[outIntervals];
        //
        for (int w = 0; w < pirmaEilute.length; w++)
            pirmaEilute[w] = Math.sin(w * tankis);
        return pirmaEilute;
    }
    public double[] netisinesFunkcijosVertes(int outIntervals){

```

```

double xx = 0.1;
double[] pirmaEilute = new double[outIntervalas];
for (int w = 0; w < pirmaEilute.length; w++) {
    pirmaEilute[w] = 4 * (xx) * (1 - xx);
    xx = 4 * (xx) * (1 - xx);
}
//System.out.println(pirmaEilute[w]);
return pirmaEilute;
// }
}

public static double[] paketuSkaicius(String DBname, String tablename, double
intervalas, int outIntervalas,
                                int tipas, int diena, int valanda) {
double[] grazinamiDuomenys = new double[outIntervalas];
System.out.println("class Duomenys paketuSkaicius" + outIntervalas);
try {
    Class.forName("org.postgresql.Driver");
    Connection con = DriverManager.getConnection("jdbc:postgresql://localhost:5432/"
+ DBname,
        "postgres", "postgres");

    if (con != null) {
        System.out.println("Connected");
        //////////////////////////////////
        Laikai laik = new Laikai();

        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date date = dateFormat.parse(laik.prLaikas(diena));
        Date dateEnd = dateFormat.parse(laik.pbLaikas(diena));

        Timestamp timestamp = new Timestamp(date.getTime());
        Timestamp timestampEnd = new Timestamp(dateEnd.getTime());

        if(valanda > 8 && diena < 2007) { //nustatome pradini laika
            timestamp.setTime(timestamp.getTime() + ( (valanda-8) * 60 * 60 * 1000) );
        }
        Date dateS = dateFormat.parse("2007-08-05 00:14:36");
        Timestamp timestampSuolis = new Timestamp(dateS.getTime());

        System.out.println(timestamp);
        //////////////////////////////////
        double[] pirmaEilute;

        pirmaEilute = new double[outIntervalas];
        int kint = -1, pkSkSumaInt = 0;
        int numeris = 0;
        Statement stmt = con.createStatement();
        Pavadinimas pav = new Pavadinimas();
        String textas;
        //////////////////////////////////

```

```

String failoPav2 = diena + "d" + valanda + "val." + pav.Failo_varas(tipas) +
"_int"
    + intervalas + "Verciu" + 2*outIntervalas + ".txt";

PrintWriter outwrite2 = new PrintWriter(failoPav2);
outwrite2.println("Nr. " + valanda);
////////////////////////////////////
//System.out.println("Connected");
while (!timestampEnd.before(timestamp)) {
    //////////////////////////////////
    Calendar cal = Calendar.getInstance();
    cal.setTimeInMillis(timestamp.getTime());
    cal.add(Calendar.MILLISECOND, (int) (intervalas * 1000));
    Timestamp timestampsSecond = new Timestamp(cal.getTime().getTime());
    //////////////////////////////////
    textas = "SELECT COUNT(timestamp) FROM " + tablename + " where
timestamp >= '" + timestamp
        + "' and timestamp < '" + timestampsSecond + "'";
    //System.out.println("do");
    ResultSet result = stmt.executeQuery(textas);
    result.next();
    timestamp = timestampsSecond;
    if (diena == 2007)
        if(timestamp.equals(timestampSuolis))
            timestamp.setTime(timestamp.getTime() + ( 5 * 60 * 1000) );
    kint++;
    pirmaEilute[kint] = result.getInt(1);
    pkSkSumaInt += pirmaEilute[kint];
    //System.out.println("tai " + pirmaEilute[kint]);
    outwrite2.println( (numeris*pirmaEilute.length+kint+1) + " " +
pirmaEilute[kint]);
    if (kint == outIntervalas - 1) {
//        for (int y=0; y< pirmaEilute.length; y++)
//            outwrite2.println( (numeris*pirmaEilute.length+y+1) + " " +
pirmaEilute[y]);
        if (numeris == 0) {
            for (int y=0; y< pirmaEilute.length; y++)
                grazinamiDuomenys[y] = pirmaEilute[y];
            //System.out.println("sdfsdfsdfsdf" + pirmaEilute[25] +
grazinamiDuomenys[25]);
        }
        numeris++;
        if(numeris == 2 )
            break;
        pkSkSumaInt = 0;
        kint = -1;
    }
}
    outwrite2.close();
} else {

```

```

        System.out.println("NOT Connected");
    }
} catch (Exception ee) {
    ee.printStackTrace();
}
//System.out.println("sdfsdfsdfsdf " + grazinamiDuomenys[25]);
return grazinamiDuomenys;
}

public static double[] skirtinguIpAdresuSkaicius(String DBname, String tablename,
String column_name, double intervalas, int outIntervalas,
int tipas, int diena, int valanda) {
    double[] grazinamiDuomenys = new double[outIntervalas];
    System.out.println("class Duomenys skirtinguIpAdresuSkaicius" + outIntervalas);
    try {
        Class.forName("org.postgresql.Driver");
        Connection con = DriverManager.getConnection("jdbc:postgresql://localhost:5432/"
+ DBname,
        "postgres", "postgres");

        if (con != null) {
            System.out.println("Connected");
            ///////////////////////////////////////////////////
            Laikai laik = new Laikai();

            DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            Date date = dateFormat.parse(laik.prLaikas(diena));
            Date dateEnd = dateFormat.parse(laik.pbLaikas(diena));

            Timestamp timestamp = new Timestamp(date.getTime());
            Timestamp timestampEnd = new Timestamp(dateEnd.getTime());

            if(valanda > 8 && diena < 2007) { //nustatome pradini laika
                timestamp.setTime(timestamp.getTime() + ( (valanda-8) * 60 * 60 * 1000) );
            }
            Date dateS = dateFormat.parse("2007-08-05 00:14:36");
            Timestamp timestampSuolis = new Timestamp(dateS.getTime());
            System.out.println(timestamp);
            ///////////////////////////////////////////////////
            double[] pirmaEilute;

            pirmaEilute = new double[outIntervalas];
            int kint = -1, pkSkSumaInt = 0;
            int numeris = 0;
            Statement stmt = con.createStatement();
            Pavadinimas pav = new Pavadinimas();
            String textas;
            ///////////////////////////////////////////////////
            String failoPav2 = diena + "d" + valanda + "val." + pav.Failo_varas(tipas) +
            "_int"
            + intervalas + "Verciu" + 2*outIntervalas + ".txt";

```

```

PrintWriter outwrite2 = new PrintWriter(failoPav2);
outwrite2.println("Nr. " + valanda);

////////////////////////////////////
//System.out.println("Connected");
while (!timestampEnd.before(timestamp)) {
    //////////////////////////////////
    Calendar cal = Calendar.getInstance();
    cal.setTimeInMillis(timestamp.getTime());
    cal.add(Calendar.MILLISECOND, (int) (intervalas * 1000));
    Timestamp timestampsSecond = new Timestamp(cal.getTime().getTime());
    //////////////////////////////////
    textas = "SELECT COUNT(DISTINCT " + column_name + ") FROM " +
    tablename + " where timestamp >= '"
        + timestamp + "' and timestamp < '" + timestampsSecond + "';";

    //System.out.println("do");
    ResultSet result = stmt.executeQuery(textas);
    result.next();
    timestamp = timestampsSecond;
    if (diena == 2007)
        if(timestamp.equals(timestampSuolis))
            timestamp.setTime(timestamp.getTime() + ( 5 * 60 * 1000) );
    kint++;
    pirmaEilute[kint] = result.getInt(1);
    pkSkSumaInt += pirmaEilute[kint];
    outwrite2.println( (numeris*pirmaEilute.length+kint+1) + " " +
    pirmaEilute[kint]);
    //System.out.println("tai " + pirmaEilute[kint]);
    if (kint == outIntervalas - 1) {

        if (numeris == 0) {
            for (int y=0; y< pirmaEilute.length; y++)
                grazinamiDuomenys[y] = pirmaEilute[y];
        }
        numeris++;
        if(numeris == 2 )
            break;
        pkSkSumaInt = 0;
        kint = -1;
    }
}
outwrite2.close();
} else {
    System.out.println("NOT Connected");
}
} catch (Exception ee) {
    ee.printStackTrace();
}

```

```

        return grazinamiDuomenys;
    }
    public double[] entropija(String DBname, String tablename, String stulpelio_pav, double
intervalas,
        int outIntervalas, int tipas, int diena, int valanda){
        double[] grazinamiDuomenys = new double[outIntervalas];
        try{
            Class.forName("org.postgresql.Driver");
            Connection con = DriverManager.getConnection("jdbc:postgresql://localhost:5432/"
+ DBname, "postgres", "postgres");
            if(con != null){
                System.out.println("Connected");
                //////////////////////////////////////////////////
                Laikai laik = new Laikai();
                DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                Date date = dateFormat.parse(laik.prLaikas(diena));
                Date dateEnd = dateFormat.parse(laik.pbLaikas(diena));
                Timestamp timestamp = new Timestamp(date.getTime());
                Timestamp timestampEnd = new Timestamp(dateEnd.getTime());
                if(valanda > 8 && diena < 2007) { //nustatome pradini laika
                    timestamp.setTime(timestamp.getTime() + ( (valanda-8) * 60 * 60 * 1000) );
                }
                Date dateS = dateFormat.parse("2007-08-05 00:14:36");
                Timestamp timestampSuolis = new Timestamp(dateS.getTime());
                System.out.println(timestamp);
                //////////////////////////////////////////////////
                double[] pirmaEilute = new double[outIntervalas];
                int kint = -1 ;
                double pkSkSumaInt = 0.0;
                int numeris = 0;
                double entropija;
                double etr1 =0.0, etr2 = 0.0, etr3 = 0.0;
                int normuoti = 1;
                //////////////////////////////////////////////////
                Statement stmt = con.createStatement();
                Pavadinimas pav = new Pavadinimas();
                String textas, failoPav = "bloga info";
                failoPav = diena + "d" +valanda + "val" + pav.Failo_varas(tipas) + "_int" +
intervalas
                    + "Verciu" + 2*outIntervalas + ".txt";
                PrintWriter outwrite = new PrintWriter(failoPav);
                outwrite.println("Nr. " + valanda);
                while(!timestampEnd.before(timestamp)){
                    //////////////////////////////////////////////////
                    Calendar cal = Calendar.getInstance();
                    cal.setTimeInMillis(timestamp.getTime());
                    cal.add(Calendar.MILLISECOND, (int) (intervalas*1000));
                    Timestamp timestampsSecond = new Timestamp(cal.getTime().getTime());
                    //////////////////////////////////////////////////
                    entropija = 0.0;

```

```

        textas = "SELECT COUNT(timestamp) FROM " + tablename + " where
timestamp >= '" + timestamp
        + "' and timestamp < '" + timestampsSecond + "'";

        ResultSet eil_sk = stmt.executeQuery(textas); // grazina duomenu skaiciu tame
intervale.
        eil_sk.next();
        double eiluciu_sk = eil_sk.getDouble(1);
        textas = "SELECT " + stulpelio_pav + ", COUNT(*) FROM " + tablename + "
where timestamp >= '"
        + timestamp + "' and timestamp < '" + timestampsSecond + "' GROUP BY
"
        + stulpelio_pav + " ORDER BY " + stulpelio_pav + " ";

        if(eiluciu_sk != 0){
            ResultSet result = stmt.executeQuery(textas); //grazina kiek tam tikru ip
adresu yra tame intervale.
            while(result.next()){
                entropija = entropija + (result.getDouble("count")/eiluciu_sk)
*(log(result.getInt("count")/eiluciu_sk)/log(2));
            }
            if(normuoti == 1 && entropija != 0){
                entropija = entropija / (log(1.0/eiluciu_sk)/log(2));
            }
        }else{
            entropija = (etr1 + etr2 + etr3)/3;
            //System.out.println(entropija);
        }
        timestamp = timestampsSecond;
        if (diena == 2007)
            if(timestamp.equals(timestampSuolis))
                timestamp.setTime(timestamp.getTime() + ( 5 * 60 * 1000) );
        etr3 = etr2;
        etr2 = etr1;
        etr1 = entropija;

        kint++;
        pirmaEilute[kint] = entropija;
        pkSkSumaInt += pirmaEilute[kint];
        outwrite.println( (numeris*pirmaEilute.length+kint+1) + " " +
        pirmaEilute[kint]);
        if (kint == outIntervalas - 1) {
            if (numeris == 0) {
                for (int y=0; y< pirmaEilute.length; y++)
                    grazinamiDuomenys[y] = pirmaEilute[y];
            }
            numeris++;
            if(numeris == 2 )
                break;
            pkSkSumaInt = 0;

```

```

        kint = -1;
    }
    }
    outwrite.close();
} else{
    System.out.println("NOT Connected");
}
}
catch(Exception ee){
    ee.printStackTrace();
}
return grazinamiDuomenys;
}
}

```

3. Optimalios delsos nustatymo klasė.

```

package bakalauras;

import java.io.PrintWriter;
import java.util.Arrays;

import static java.lang.StrictMath.log;

/**
 * Created by Faustas on 2016-05-19.
 */
public class DelsosNustatymas {

    public void mutual(double[] pirmaEilute, int outIntervalas, int vidurkTsk, double tankis)
    {
        String failoPav = "mutual" + outIntervalas + "mutualVidTsk" + vidurkTsk + ".txt";
        try{
            PrintWriter outwrite = new PrintWriter(failoPav);

            double[] rArray = mutualFunk(pirmaEilute, outIntervalas, vidurkTsk);

            for (int w =0; w<pirmaEilute.length; w++)
                outwrite.println( (w+1) + " " + rArray[w]);

            outwrite.close();
        }
        catch (Exception ee) {
            ee.printStackTrace();
        }
    }

    public static double[] mutualFunk(double[] array, int arrayIlgis, int vidurkTsk){
        double[] dArray;
    }
}

```

```

dArray = arrayParuosimas(array); //gaunami paruostis duomenys tolesniems
skaiciavimams
double[] mutualArray = new double[arrayIlgis];
int[] ddArray = new int[arrayIlgis];
Arrays.fill(mutualArray, 0);
Arrays.fill(ddArray, 0);

int bins = (int) (1 + log(arrayIlgis)/log(2) + 0.5);
for (int i = 0; i < array.length; i++)
    if (dArray[i] < 1) {
        ddArray[i] = (int) (dArray[i] * bins);
    }
    else
        array[i] = bins - 1;
double[][] matrica = new double[bins][bins];
double[] x = new double[bins];
double[] y = new double[bins];
int sk = 0;
////////////////////////////////////
for (int i = 0; i < arrayIlgis; i++) {
    for (int n1 = 0; n1 < bins; n1++) //bins matrica uzpildoma 0
        for (int n2 = 0; n2 < bins; n2++)
            matrica[n1][n2] = 0;
    Arrays.fill(x, 0);
    Arrays.fill(y, 0);
    //System.out.println("4");
    for (int j = 0; j < arrayIlgis; j++) {
        if (j + i < arrayIlgis)
            matrica[ddArray[j+i]][ddArray[j]] = matrica[ddArray[j+i]][ddArray[j]] + 1.0;
        else {
            matrica[ddArray[sk]][ddArray[j]]++;
            sk++;
        }
    }
    sk = 0;
    ////normavimas
    for (int k1 = 0; k1 < bins; k1++) {
        for (int k2 = 0; k2 < bins; k2++) {
            matrica[k1][k2] /= arrayIlgis;
            matrica[k1][k2] += Math.pow(2.22, -16);
        }
    }
    for (int k1 = 0; k1 < bins; k1++)
        for (int k2 = 0; k2 < bins; k2++) {
            x[k1] += matrica[k1][k2]; //eilutes suma
            y[k1] += matrica[k2][k1]; //stulpelio suma
        }
    for (int h1 = 0; h1 < bins; h1++)
        if (x[h1] > 0)
            for (int h2 = 0; h2 < bins; h2++)

```

```

        if (y[h2] > 0)
            if (matrica[h1][h2] > 0)
                mutualArray[i] += matrica[h1][h2] * ( log( matrica[h1][h2] / (x[h1] *
y[h2]) )/log(2) );
    }
    int laik=1;
    double[] mutualArray2 = new double[arrayIlgis];
    Arrays.fill(mutualArray2, 0);
    mutualArray2[0] = mutualArray[0];
    for (int k = 1; k < arrayIlgis-laik; k++) {
        if(k<=(vidurkTsk-1)/2) //nurodo kiek tasku is sono
            laik = k;

        mutualArray2[k] = mutualArray[k];
        for (int l = 1; l < laik + 1; l++)
            mutualArray2[k] += mutualArray[k - l] + mutualArray[k + l];
        mutualArray2[k] /= 2*laik+1;
    }
    return mutualArray2;
}
////////////////////////////////////
public static double[] arrayParuosimas(double[] array){
    double min = array[0];
    for (int i = 1; i < array.length; i++) {
        if (array[i] < min) {
            min = array[i];
        }
    }
    for (int i = 0; i < array.length; i++)
        array[i] -= min;
    double max = array[0];
    for (int i = 1; i < array.length; i++) {
        if (array[i] > max) {
            max = array[i];
        }
    }
    double[] dArray = new double[array.length];
    Arrays.fill(dArray, 0);
    for (int i = 0; i < array.length; i++) {
        dArray[i] = array[i] / max;
    }
    return dArray;
}
public static double[] korSumavimas(double[] array, int arrayIlgis, int vidurkTsk){
    double normavimoKonst;
    double laikSuma = 0;
    double corSuma = 0;
    double[] corVerte = new double[arrayIlgis];
    for(int i = 0; i < arrayIlgis; i++)
        laikSuma += Math.pow(array[i], 2);

```

```

normavimoKonst = 1.0 / ( laikSuma);
int sk = 0;
for(int i = 0; i < arrayIlgis; i++){
    for(int j = 0; j < arrayIlgis; j++){
        if(j+i < arrayIlgis)
            corSuma += array[j] * array[j+i];
        else {
            corSuma += array[j] * array[sk];
            sk++;
        }
    }
    corVerte[i] = normavimoKonst * corSuma;
    corSuma = 0;
    sk = 0;
}
int laik=1;
double[] corVerte2 = new double[arrayIlgis];
Arrays.fill(corVerte2, 0);
corVerte2[0]=corVerte[0];
for (int k = 1; k < arrayIlgis-laik; k++) {
    if(k<=(vidurkTsk-1)/2 ) //nurodo kiek tasku is sono
        laik = k;
    corVerte2[k] = corVerte[k];
    for (int l = 1; l < (laik +1); l++)
        corVerte2[k] += corVerte[k - l] + corVerte[k + l];
    corVerte2[k] /= 2*laik+1;
}
return corVerte2;
}

public static void koreliacijaTikra(double[] array, int arrayIlgis){
    double normavimoKonst;
    double laikSuma = 0;
    double corSuma = 0;
    double vidurkis = 0.0;
    double[] corVerte = new double[arrayIlgis];
    for(int i = 0; i < arrayIlgis; i++)
        vidurkis += array[i];
    vidurkis = vidurkis/(double)arrayIlgis;
    for(int i = 0; i < arrayIlgis; i++)
        laikSuma += Math.pow(array[i] - vidurkis, 2);

    normavimoKonst = 1.0 / ( laikSuma);
    int sk = 0;
    for(int i = 0; i < arrayIlgis; i++){
        for(int j = 0; j < arrayIlgis; j++){
            if(j+i < arrayIlgis)
                corSuma += (array[j] - vidurkis) * (array[j+i] - vidurkis);
            else {
                corSuma += (array[j] - vidurkis) * (array[sk] - vidurkis);
                sk++;
            }
        }
        corVerte[i] = normavimoKonst * corSuma;
        corSuma = 0;
        sk = 0;
    }
}

```

```

        }
    }
    corVerte[i] = normavimoKonst * corSuma;
    corSuma = 0;
    sk = 0;
}
String failoPav = "koreliacija" + ".txt";
try{
    PrintWriter outwrite = new PrintWriter(failoPav);

    for (int w =0; w<corVerte.length; w++)
        outwrite.println( (w+1) + " " + corVerte[w] + " " + 0.368);

    outwrite.close();
}
catch (Exception ee) {
    ee.printStackTrace();
}
}
}

```

4. Fazinės erdvės sudarymas.

```

package bakalauras;

import java.io.PrintWriter;
import java.sql.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import static java.lang.Math.log;

/**
 * Created by Faustas on 2016-04-20.
 */
public class FaizneErdve {
    public void fzineErdve(double[] pirmaEilute, int outIntervalas, int delta){
        String failoPav = "FE" + "_int" + outIntervalas + "delta" + delta + ".txt";
        String failoPav1 = "FE" + "_int" + outIntervalas + "delta" + delta + "Eilute=Delta" +
        ".txt";
        String failoPav2 = "FE" + "_int" + outIntervalas + "delta" + delta + "Stulpelis=Delta" +
        ".txt";
        try{
            PrintWriter outwrite = new PrintWriter(failoPav);
            PrintWriter out = new PrintWriter(failoPav1);
            PrintWriter out2 = new PrintWriter(failoPav2);
            int numeris = 0;

```

```

        for (int z1 = 0; z1 < outIntervalas; z1 += delta) {
            //System.out.println(pirmaEilute[z1]);
            for (int z2 = 0; z2 < delta; z2++) {
                if (z2 == 0)
                    outwrite.print((numeris++));
                if ((z2 + z1) < outIntervalas)
                    outwrite.print(" " + pirmaEilute[z2 + z1]);
            }
            outwrite.println();
        }
        outwrite.close();
        /*FE pirma eilute vertes is veiles iki delta antra prasideda nuo delta ir eina iki
2*delta*/
        numeris = 0;
        for (int j=0; j < (pirmaEilute.length-delta); j+=delta) {
            for (int i = 0; i < delta; i++) {
                if (i == 0)
                    out.print((numeris++));
                out.print(" " + pirmaEilute[i + j]);
            }
            out.println();
        }
        out.close();
        /*////////////////////////////////////*/
        numeris = 0;
        for (int j=0; j<pirmaEilute.length; j++) {
            for (int i = 0; i < pirmaEilute.length - j; i += delta) {
                if (i == 0)
                    out2.print((numeris++));
                out2.print(" " + pirmaEilute[i + j]);
            }
            out2.println();
        }
        out2.close();
    }
    catch (Exception ee) {
        ee.printStackTrace();
    }
}
}

```

5. Rkonstuotos erdvės dimansijų nustatymo klasė.

package bakalauras;

```

import java.io.PrintWriter;
import java.util.ArrayList;

```

```

/**

```

```

 * Created by Faustas on 2016-05-18.

```

```

 */

```

```

public class DimensijuNustatymas {
    public static void embeddingDimencijuSkaicius (double[] pirmaEilute, int delta){
        System.out.println("class DimensijuNustatymas");
        int count = 0;
        double[][] masyvasD = new double[50][2];
        for (int m=2 ; m < masyvasD.length+2; m++){
            int stulpSk = pirmaEilute.length -(m-1)*delta;
            if (stulpSk<1){
                System.out.println("permazai DUOMENU tokiam m");
                break;
            }
            masyvasD[m-2][0]= m;
            masyvasD[m-2][1] = erdvesRekontravimas(pirmaEilute, m, delta);
            count++;
        }
        spauzdinimas(masyvasD, count, delta);
    }
    private static void spauzdinimas(double[][] data, int ilgis, int delta){
        String pav = "Optimalus m delta" + delta + ".txt";
        try{
            PrintWriter out = new PrintWriter(pav);
            for (int n1 = 0; n1 < ilgis; n1++) {
                out.println(data[n1][0] + " " + data[n1][1]);
            }
            out.close();
        } catch (Exception ee) {
            ee.printStackTrace();
        }
    }
    public static double erdvesRekontravimas(double[] pirmaEilute, int m, int delta){
        int stulpSk = pirmaEilute.length -(m-1)*delta;
        double[][] rekonstruotaErdve = new double[m][stulpSk];

        for (int i=0; i < m; i++){
            for (int j=0; j < stulpSk; j++)
                rekonstruotaErdve[i][j] = pirmaEilute[(j+(m-1)*delta) - i*delta];
        }
        return EuklidinisAtstumas(rekonstruotaErdve, stulpSk, delta);
    }
    private static double EuklidinisAtstumas(double[][] rErdve,int stulpSk, int delta ){
        double atstumuKvadSuma;
        double h=0;
        double radius = 0.2;
        double[] masyvasC = new double[(int)(delta/radius*2)];
        for (int r=1; r <= (int)(delta/radius*2); r++) {
            h = 0;
            for (int ii = 0; ii < stulpSk; ii++) {
                for (int i = 0; i < stulpSk; i++) {
                    atstumuKvadSuma = 0.0;
                    for (int j = 0; j < rErdve.length; j++)

```

```

        atstumuKvadSuma += Math.pow((rErdve[j][ii] - rErdve[j][i]), 2);
        if ((r*radius - Math.sqrt(atstumuKvadSuma)) >= 0)
            h++;
    }
}
masyvasC[r-1] = h/Math.pow(stulpSk, 2);
}
return koeficientoRadimas(masyvasC, radius);
}
private static double koeficientoRadimas (double[] masyvasC, double radius ) {
    double k = 0.0;
    double kLaik= 0.0;
    double Const = 0.000001;
    int c =0;
    for (int r=0; r< masyvasC.length; r++) {
        kLaik = Math.log(masyvasC[r]) / Math.log( (r+1)*radius );
        if (kLaik > Const )
            k += kLaik;
    }
    k = k/masyvasC.length;
    return k;
}
private static double[][] vidurkinimasD(double[][] masyvasD){
    double[][] d =new double[masyvasD.length][2];
    d[0][0]=masyvasD[0][0];
    d[0][1]=masyvasD[0][1];
    d[masyvasD.length-1][0]=masyvasD[masyvasD.length-1][0];
    d[masyvasD.length-1][1]=masyvasD[masyvasD.length-1][1];
    for (int i = 1; i < masyvasD.length-2; i++) {
        d[i][0] = masyvasD[i][0];
        d[i][1] = (masyvasD[i-1][1] +masyvasD[i][1] + masyvasD[i+1][1])/3;
    }
    return d;
}
}
}

```

6. Srauto verčių prognozavimo klasė.

```

private double[][] getCoeficientMatrix(double[] duomenys, Integer m, Integer k, Integer
delta) {
    Integer pointCountas = m + 1;
    double[][] matrixB = new double[pointCountas][pointCountas];
    double[][] matrixD = new double[1][pointCountas];
    List<Integer> neighbourList = getLastPointNeighbours(duomenys, m, k, delta);
    //sudarom B matricą
    for (int i = 0; i < m + 1; i++) {
        for (int j = 0; j < neighbourList.size(); j++) {
            double[] column = new double[pointCountas];
            double[] vector = getKey(duomenys, m, delta, neighbourList.get(j));
            column[0] = 1;

```

```

        System.arraycopy(vector, 0, column, 1, vector.length);

        for (int kk = 0; kk < column.length; kk++) {
            matrixB[kk][j] = column[kk];
        }
    }
    //sudarom D matricą
    for (int i = 0; i < neighbourList.size(); i++) {
        matrixD[0][i] = duomenys[neighbourList.get(i) + 1];
    }
    return getCoefficientMatrix(matrixB, matrixD);
}

private List<Integer> getLastPointNeighbours(double[] valueArray, int dimensionCount,
int neighbourPointLimit,
                                int optimalTimeDelay) {
    List<Integer> neighbours=null;
    try {
        KDTree<Integer> kdTree = new KDTree<Integer>(dimensionCount);
        int lastIndex = valueArray.length - 2; // -2 nes mums reikia priešpaskutinio indekso
        for (int i = 0; i < neighbourPointLimit; i++) {
            double[] key = getKey(valueArray, dimensionCount, optimalTimeDelay,
lastIndex - i);
            if (kdTree.search(key) == null) {
                kdTree.insert(key, lastIndex - i);
            }
        }
        double[] lastKey = getKey(valueArray, dimensionCount, optimalTimeDelay,
lastIndex);
        neighbours = kdTree.nearest(lastKey, dimensionCount + 1); //gražinam m + 1
kaimynų
        return neighbours;
    } catch (Exception e) {
        //throw new GeneralException("KDTree failed!", e);
    }
    return neighbours;
}

private double[] getKey(double[] valueArray, int dimensionCount, int optimalTimeDelay,
int index) {
    double[] key = new double[dimensionCount];
    for (int i = 0; i < dimensionCount; i++) {
        key[i] = valueArray[index - (i * optimalTimeDelay)];
    }
    return key;
}

private double[][] getCoefficientMatrix(double[][] matrixB, double[][] matrixD) {
    QRDecomposition qrDecomposition = new QRDecomposition(
        new BlockRealMatrix(matrixB)

```

```

    );

    RealMatrix matrixQT = qrDecomposition.getQT();
    RealMatrix matrixR = qrDecomposition.getR();

    RealMatrix matrixRinverse = new
QRDecomposition(matrixR).getSolver().getInverse();

    RealMatrix coefficientMatrix = new BlockRealMatrix(matrixD)
        .multiply(matrixRinverse)
        .multiply(matrixQT);

    return coefficientMatrix.getData();
}
private double[][] multiplyMatrixes(double[][] first, double[][] second) {
    RealMatrix firstMatrix = new BlockRealMatrix(first);
    RealMatrix secondMatrix = new BlockRealMatrix(second);

    return firstMatrix.multiply(secondMatrix).getData();
}
//public void sin(int m, int delta, int outIntervalas, int spejamuVerciuSk, double tankis){
public void verciuSpejimas(double[] duomenys, int m, int delta, int outINT, int
spejamiTsk, double tankis) {
    int k = 2 * m + 1;
    String failoPavadinimas = "Spejimas" + "delta" + delta + "m" + m + ".txt";
    try {
        PrintWriter outwrite2 = new PrintWriter(failoPavadinimas);
        for (int n1 = 0; n1 < duomenys.length; n1++) {
            outwrite2.println(n1 + 1 + " " + duomenys[n1] + " " + duomenys[n1]);
        }
        outwrite2.flush();
        for (int i = 0; i < spejamiTsk; i++) {

            double[][] coefficientMatrix = getCoefficientMatrix(duomenys, m, k, delta);
            double[][] Y2Matrix = new double[m + 1][1];

            double[] vector = getKey(duomenys, m, delta, duomenys.length - 1);
            Y2Matrix[0][0] = 1.0;
            for (int kk = 0; kk < vector.length; kk++) {
                Y2Matrix[kk + 1][0] = vector[kk];
            }

            double[][] predictedMatrix = multiplyMatrixes(coefficientMatrix, Y2Matrix);
            double predictedValue = predictedMatrix[0][0];
            System.out.println(predictedValue);

            System.arraycopy(duomenys, 1, duomenys, 0, duomenys.length - 1);
            duomenys[duomenys.length - 1] = predictedValue;

```

```

        outwrite2.println(duomenys.length+i+1 + " " + predictedValue);
        outwrite2.flush();
    }

    outwrite2.close();

} catch (Exception ee) {
    ee.printStackTrace();
}
}

```

7. Laikų klasė.

```

package bakalauras;

/**
 * Created by Faustas on 2016-04-18.
 */
public class Laikai {
    public String prLaikas (int diena){
        String return_val = null;
        switch (diena) {
            case 11:
                return_val = "1999-03-01 08:00:05";
                break;
            case 12:
                return_val = "1999-03-02 08:00:01";
                break;
            case 13:
                return_val = "1999-03-03 08:00:01";
                break;
            case 14:
                return_val = "1999-03-04 13:00:01";
                break;
            case 15:
                return_val = "1999-03-05 08:00:01";
                break;
            case 21:
                return_val = "1999-03-08 08:00:00";
                break;
            case 22:
                return_val = "1999-03-09 08:00:01";
                break;
            case 2007:
                //return_val = "2007-08-04 23:49:36";
                return_val = "2007-08-04 23:57:54";
                break;
        }
        return return_val;
    }
}

```

```

public String pbLaikas(int diena){
    String return_val = null;
    switch (diena) {
        case 11:
            return_val = "1999-03-02 06:00:16";
            break;
        case 12:
            return_val = "1999-03-03 05:59:57";
            break;
        case 13:
            return_val = "1999-03-04 05:59:58";
            break;
        case 14:
            return_val = "1999-03-05 10:59:52";
            break;
        case 15:
            return_val = "1999-03-06 05:59:58";
            break;
        case 21:
            return_val = "1999-03-09 05:59:54";
            break;
        case 22:
            return_val = "1999-03-10 03:03:21";
            break;
        case 2007:
            //return_val = "2007-08-05 00:14:36";
            return_val = "2007-08-05 00:20:39";
            break;
    }
    return return_val;
}
}

```