Modele de Trafic pentru Reţele de Date

Radu GRIGORE
conducător ştiinţific: prof. dr. ing. Grazziela NICULESCU

23 iunie 2003

Prefaţă

Acest subiect mi se pare interesant in primul rand pentru ca este interdisciplinar. Este vorba despre studiul traficului in retele LAN si WAN; asadar este nevoie de cunoasterea cel putin a principiilor de functionare ale unor tehnologii noi. Baza matematica o constituie teoria proceselor stocastice, iar pentru unele modele aceasta se imbina putin si cu teoria fractalilor. Intr-adevar procesele stocastice autosimilare au fost studiate mai intai de B. Mandelbrot, care a facut foarte multe pentru aplicarea in diverse domenii a fractalilor.

Aplicarea modelelor autosimilare traficului din retelele de telecomunicatii a inceput practic cand o echipa de cercetatori de la laboratoarele Bellcore si universitatea din Boston si-au publicat rezultatele [2]. Era in perioada cand se lucra la standardizarea ATM-ului si una dintre probleme era dimensionarea corecta a memorilor tampon. Iata cum prezinta David Sincoskie, unul dintre inventatorii puntilor Ethernet ce invata singure, aceasta problema:

> O data ce traficul periodic a fost bine inteles am inceput sa investigam alte forme de trafic. Traficul aleator (celulele sunt destinate iesirilor cu probabilitati egale) era de asemenea usor de suportat cu memorii tampon de dimensiuni relativ mici. Nu poti atinge pierderi zero, dar probabilitatea de pierderi scade exponential cand se adauga memorie comutatorului. S-au folosit modele Markov pentru a produce trafic intermitent, dar a devenit repede evident ca nu avem nici un model bun pentru sursele de trafic de date. Desi puteam spune ca traficul intermitent cere mult mai

> multa memorie decat traficul periodic sau aleator si ca necesarul de memorie creste cu lungimea rafalelor, totusi nu aveam nici un model bun pentru timpul intre sosiri sau lugimea rafalelor care sa aiba legatura cu realitatea. De fapt nu exista nici un model bun pentru traficul de date. Ne impotmolisem.
>
> I-am rugat pe doi dintre cercetatorii mei, Dan Wilson si Will Leland, sa se uite la problema modelarii traficului de date. Un al treilea cercetator, Mark Garrett, a fost incurajat sa examineze traficul video. Abordarea pe care am sugerat-o a fost sa examineze trafic video real. Mark a ales sa digitizeze un intreg film, creand o populara baza de date cu statistici de trafic. Pe Dan l-am incurajat sa captureze niste trafic din retelele Ethernet pe care le foloseam la serviciu si sa analizeze rezultatele. Banuiam ca problema va fi dificila si va dura ceva. Nu am fost dezamagit. Dupa aproximativ cinci ani, in 1993, Wilson, Leland, Willinger si Taqqu vor descoperi si publica un articol de referinta despre natura autosimilara a traficului de date. Desi munca lor a avansat mult modelarea traficului de date, rezultatele au fost prea tarzii pentru a ajuta proiectarea comutatoarelor ATM de la Bellcore, lucru care se va intoarce impotriva noastra cativa ani mai tarziu.

Desi suntem la zece ani dupa publicarea acelui articol de referinta totusi noile modele nu sunt inca suficient exploatate. Problema dimensionarii memoriilor tampon a fost rezolvata cat de cat satisfacator. Este adevarat, nu exista relatii analitice care sa lege probabilitatea de pierdere de dimensiunea memoriei, dar s-au dezvoltat metode de generare de trafic care are caracteristici asemanatoare cu traficul real si care pot fi folosite la dimensionarea memoriilor cu ajutorul simularilor. Metoda este oricum mult mai convenabila decat utilizarea in simulari ale unor capturi de trafic[1].

[1] De unde stiu ca nu au aparut deja pierderi inainte sa captez eu traficul? Unde gasesc o retea reala cu trafic mai intens? etc.

S-a propus insa sa se utilizeze aceste modele si pentru controlul congestiei. In acest sens pasii facuti sunt inca timizi. Realizarea acestui lucru ar presupune, probabil, implementarea urmatoarelor mecanisme in protocoalele de telecomunicatii (mai exact TCP si IP):

1. Estimarea parametrilor modelului pentru traficul observat de statie, comutator, etc...

2. Utilizarea parametrilor estimati pentru ajustarea dinamica a unor parametrii functionali (cum ar fi rata de emisie, utilizarea memoriei tampon pentru diverse porturi, etc.)

In aceasta lucrare am investigat prima dintre aceste probleme. Rezultatul este un program care poate fi folosit ca instrument de lucru in aceasta directie. Iata deci o alta dimensiune a interdisciplinaritatii: programarea.

Listă de figuri

Cuprins

Capitolul 1

Introducere

In lucrarea de fata este prezentat un program ce poate fi utilizat pentru estimarea in timp real a parametrului Hurst al traficului observat de placa de retea a unui calculator personal. Denumirea lui este RTH (Real Time Hurst). Parametrul Hurst descrie autosimilaritatea unui proces stocastic. Modelarea traficului cu ajutorul proceselor stocastice autosimilare isi are inceputul in lucrarea [2] si este motivata de dorinta de a explica intermitenta traficului real observata la multe scari de timp. Ca urmare parametrul Hurst ofera si o cuantificare a notiunii de intermitenta.

Pana nu demult toate studiile de trafic privind modele autosimilare se faceau prin captarea pe o durata mare si analiza ulterioara. Un avantaj al analizei in timp real (sau on-line) este acela ca rezultatul poate fi utilizat in ajustarea comportamentului protocoalelor de telecomunicatii. Un alt avantaj este ca ajuta un cercetator sa observe rapid ce efect au diverse aplicatii, topologii de retea, etc. asupra traficului generat in retea. Un dezavantaj este ca estimarea in timp real fie consuma mult din resursele de calcul fie este mai inexacta ca urmare a metodei de estimare folosita. O problema ridicata de estimarea in timp real este adaptarea metodelor clasice de estimare.

In [22] si [23] autorii prezinta dispozitive hardware care estimeaza in timp real parametrul Hurst intr-o retea Ethernet, respectiv intr-o retea ATM. Pentru acestea consumul resurselor de calcul nu este o problema deoarece sunt dedicate estimarii. Fiind hardware contruit special pentru monitorizarea traficului ele au si caracteristici mai bune decat o placa de retea obisnuita in ceea ce priveste procentul de cadre capturate, rezolutia temporala, etc. Totusi, nefiind inca larg raspandite ele nu contribuie mult la studierea si mai ales la aplicarea modelelor autosimilare[1]. Un utilitar care functioneaza pe un calculator personal obisnuit ar trebui sa fie mult mai util celor care se ocupa de implementarea protocoalelor cum ar fi TCP si care vor sa includa si rezultatele unor cercetari noi. Principala arie de aplicabilitate in implementarea protocoalelor va fi probabil imbunatatirea controlului congestiei (vezi [1] pentru o prezentare a acestei probleme din punctul de vedere clasic).

Ca o ilustrare a utilizarii RTH in aceasta lucrare:

1. se prezinta parametrul Hurst masurat pentru traficul generat in reteaua locala a unei institutii

2. se face o comparatie din punctul de vedere al timpului de calcul intre metoda periodogramei si metoda varianta-agregare

In prima parte a lucrarii sunt prezentate din punct de vedere teoretic diverse modele de trafic. In capitolul 2 sunt prezentate modele clasice pentru trafic: modele de trafic de reinnoire, modele Markov, modele de tip fluid si modele autoregresive. In capitolul 3 sunt prezentate procesele stocastice autosimilare si modul in care sunt aplicate pentru modelarea traficului. In capitolul 4 sunt prezentate cateva metode clasice de estimare a parametrului Hurst.

In a doua parte a lucrarii este prezentat programul RTH si rezultatele obtinute cu ajutorul acestuia. In capitolul 5 este prezentata arhitectura programului RTH, diverse decizii luate la proiectare si bibliotecile folosite. In capitolul 6 sunt prezentate modificarile ce au fost aduse metodelor de estimare pentru a putea fi aplicate in timp real si cateva masuratori ale parametrului

[1]expresia "model autosimilar" este de fapt varianta scurta pentru "model de trafic bazat pe procese stocastice autosimilare"

Hurst in retele de institutie. In capitolul 7 se regasesc concluziile acestei lucrari.

Capitolul 2

Modelarea traficului

Prezentarea din acest capitol a modelelor de trafic bazate pe procese stocastice urmeaza linia generala din [7], dar contine in plus comentarii si detalii. Traficul este o notiune abstracta care identifica obiectul procesarilor dintr-o retea de telecomunicatii. De exemplu pentru o retea telefonica traficul reprezinta convorbirile, intr-o retea de date traficul reprezinta datele transferate: fisiere, email-uri, filme, pagini web, etc. Cateva definitii mai exacte vor fi date in sectiunea 2.1.

Traficul este un fenomen probabilist. Sa luam exemplul unei retele telefonice. Pentru a modela determinist traficul ar trebui sa avem un mijloc prin care sa determinam ce apeluri va efectua fiecare client al retelei. Sigur, asa ceva se poate face de exemplu daca e vorba de o retea privata de telefonie asupra careia se impun reguli stricte de utilizare. O astfel de retea ar putea fi folosita de exemplu de o companie petroliera. Aceasta retea ar avea cate un telefon langa fiecare put petrolier si unul la centru. Regula stricta de folosire pentru fiecare telefon de la un put petrolier ar fi "la ora h se suna centrul de la acest telefon pentru a anunta cantitatea de petrol extrasa in ultimele 24 de ore; in rest telefonul nu se foloseste". Exemplul poate fi considerat exagerat dar totusi nu este departe de realitate. Ei bine, intr-o astfel de retea traficul ar putea fi privit ca fiind determinist si, facand astfel, se pot construi modele mult mai apropiate de realitate decat modelele stocastice. Dar astfel de modele ar avea o arie de aplicabilitate foarte redusa pentru ca numarul de retele in care se pot impune astfel de restrictii dure este extrem de mic. In plus o astfel de restrictie are si un efect psihologic neplacut: “cum? n-am voie sa folosesc telefonul atunci cand am nevoie?”

Si iata am ajuns la o justificare intuitiva a folosirii teoriei probabilitatilor (sau stocastice): dumneavoastra stiti in general cand veti avea nevoie sa dati un telefon? Ei bine, daca nici macar dumneavoastra nu stiti atunci cum ar putea sa stie un model matematic al comportarii dumneavoastra? In principiu nu este imposibil dar puteti vedea cat de impractica ar fi o astfel de abordare.

Cum traficul are o evolutie in timp si este probabilist modelul matematic cel mai potrivit este procesul stocastic. In aceasta lucrare nu se vor prezenta procesele stocastice ca atare ci doar o clasa speciala, aceea a proceselor stocastice autosimilare (vezi capitolul 3). Inainte de a trece la definitiile matematice ale traficului sa incercam sa dam o imagine informala mai exacta asupra traficului. Orice retea de telecomunicatii poate fi privita ca o retea complicata de conducte prin care circula niste jetoane: unitatile de trafic. Conductele se intersecteaza uneori, iar acolo unde o fac exista mecanisme care dirijeaza jetoanele ce intra in intersectie. Terminatiile conductelor sunt niste cutiute care primesc si emit jetoane. Unele sunt ca niste fabrici de jetoane; altele doar mananca jetoane; iar altele primesc jetoane le prelucreaza (le rup, le lipesc, le ataseaza alte jetoane, etc.) si apoi le retrimit pe o alta conducta.

Reprezentarea aceasta este una abstracta. Jetonul poate fi un octet de date, un pachet de date sau o secunda de convorbire telefonica. Conductele pot fi cabluri telefonice, retele Ethernet sau retele ATM. Intersectiile dintre conducte pot fi comutatoare, rutere sau centrale telefonice.

2.1 Definiţii ale traficului

Emiterea unitatilor de trafic este modelata cu ajutorul unui proces punct.

Definiţia 1 (proces punct). *Se numeste proces proces punct un proces stocastic pentru care fiecare realizare particulara este o secventa crescatoare de numere reale* $T_0 = 0$, T_1, $T_2, \ldots, T_n, \ldots$

Doua reprezentari echivalente sunt procesele de numarare si procesele corespunzatoare timpului dintre sosirea a doua unitati de trafic consecutive.

Definiţia 2 (proces de numarare). *Un proces de numarare* $\{Y(t)\}_{t \geq 0}$ *este un proces aleator continuu (in timp) cu valori intregi ne-negative, unde* $Y(t) = \max\{n : T_n < t\}$ *este numarul unitatilor de trafic sosite in intervalul* $(0, t]$.

Definiţia 3 (procesul intervalelor dintre sosiri). *Procesul intervalului dintre sosiri este un proces aleator discret* $\{A_n\}$, *unde* $A_n = T_n - T_{n-1}$ *este durata scursa intre sosirile unitatilor* $n - 1$ *si* n.

Echivalenta acestor descrieri se poate observa din identitatea:

$$\{Y(t) = n\} = \{T_n \leq t < T_{n+1}\} = \Big\{ \sum_{k=1}^{n} A_k \leq t < \sum_{k=1}^{n+1} A_k \Big\} \tag{2.1}$$

Procesul de numarare este adesea denumit trafic. Derivata in timp a traficului se numeste intensitate a traficului. Daca valorile T_n sunt intregi atunci se spune despre procesele definite mai sus ca sunt in timp discret.

Uneori unitatile de trafic nu sunt identice fiecare fiind caracterizata de o incarcare pe care o aduce retelei.

Definiţia 4 (incarcare). *Incarcarea este un proces aleator discret (in timp)* $\{W_n\} = 1$.

Un exemplu tipic de incarcare este timpul de servire.

Definiţia 5 (rata traficului). *Rata traficului este* $\lambda_n = 1/E[A_n]$.

Dupa cum se vede din definitiile anterioare traficul se masoara in s^{-1}. O rata a traficului de $1 \cdot s^{-1}$ corespunde unui trafic in care durata medie dintre sosirile a doua unitati de trafic este de $1 \cdot s$. Alte notatii utilizate sunt: $\sigma_n^2 = \mathrm{Var}[A_n]$ si $c_n = \lambda_n \cdot \sigma_n$. Functia de repartitie a probabilitatilor se noteaza $F_n(t)$. De cele mai multe ori se lucreaza cu trafice pentru care $\{A_n\}$ este un proces stationar. In acest caz se omite indexul n si se folosec notatiile σ^2, c si $F(t)$.

2.2 Modele de reînnoire

In modelele de reinnoire variabilele aleatore A_n (esantionele procesului ce reprezinta traficul) sunt independente si au o distributie identica, dar oarecare. Ca urmare a independentei dintre variabilele A_n modelele de reinnoire nu pot captura fenomene precum intermitenta traficului deoarece astfel de fenomene presupun corelatii temporale.

In acest caz functia de autocorelatie este:

$$\rho(k) = \begin{cases} \sigma^2 + 1/\lambda^2 & \text{daca } k = 0 \\ 1/\lambda^2 & \text{in rest} \end{cases} \tag{2.2}$$

O clasa speciala de procese de reinnoire este clasa proceselor de reinnoire care prin superpozitie au ca rezultat tot un proces de reinnoire. Din aceasta clasa speciala fac parte procesele Poisson si procesele Bernoulli care vor fi prezentate putin mai tarziu.

Intuitiv operatia de superpozitie a proceselor ce reprezinta traficele are ca echivalent multiplexarea. Este totusi necesara o definitie mai exacta. Fie procesele intervalelor dintre sosiri $A^{(1)}$, $A^{(2)}$ si $A^{(3)}$. Fiecare realizare particulara a lui $A^{(i)}$ este complet determinata de setul $\{T_1^{(i)}, T_2^{(i)}, \ldots\}$ (vezi ecuatia 2.3). Se spune ca traficul $A^{(3)}$ este superpozitia traficelor $A^{(1)}$ si $A^{(2)}$ daca si numai daca pentru fiecare realizare particulara avem:

$$\{T_1^{(1)}, T_1^{(1)}, \ldots\} \cup \{T_1^{(2)}, T_1^{(2)}, \ldots\} = \{T_1^{(3)}, T_1^{(3)}, \ldots\} \tag{2.3}$$

2.2.1 Procese Poisson

Procesele Poisson sunt dintre cele mai vechi modele de trafic folosite. Avantajul lor este ca sunt relativ usor tractabile analitic si de aceea pot fi folosite pentru a face calcule rapide de exemplu pentru dimensionarea memoriilor tampon ale elementelor unei retele.

Definiţia 6 (proces Poisson). *Un proces aleator Poisson este un proces pentru care probabilitatea aparitiei unei sosiri in orice interval elementar dt este egala cu* $\lambda \cdot dt$*.*

Trebuie observat ca probabilitatea de aparitie a unei sosiri nu depinde de alte sosiri asa incat ne asteptam la o functie de autocorelatie de tip impuls. Aceasta definitie caracterizeaza sosirile unui proces Poisson ("procesul punct"). Urmatoarele teoreme caracterizeaza timpul dintre sosiri (procesul $\{A_n\}_{n\in\mathbf{N}}$) si numarul sosirilor (procesul $\{Y(t)\}_{n\in\mathbf{N}}$).

Teorema 1. *Variabilele aleatoare* $\{A_n\}$ *corespunzatoare unui proces Poisson sunt independente si au functia de distributie*

$$f_A(t) = \lambda \cdot e^{-\lambda t} \tag{2.4}$$

Demonstraţie. Distributia de probabilitate $f_A(t)$ ne spune ca probabilitatea de a avea o sosire dupa un interval t este $f_A(t)dt$. Pe de alta parte un interval t poate fi vazut ca fiind obtinut prin concatenarea a t/dt intervale elementare de lungime dt. Din definitie stim ca probabilitatea de a avea o sosire intr-un interval elementar este o constanta egala cu λdt. Asadar probabilitatea ca "in intervalul elementar 1 sa nu fie nici o sosire si in intervalul elementar 2 sa nu fie nici o sosire, ..., si in intervalul elementar t/dt sa nu fie nici o sosire si in intervalul $t/dt + 1$ sa fie o sosire" este

$$\begin{aligned} f_A(t)dt &= (1-\lambda dt)^{\frac{t}{dt}} \cdot \lambda dt & (2.5) \\ &= (1-\lambda dt)^{\frac{-1}{\lambda dt}\left(-\lambda dt \frac{t}{dt}\right)} \cdot \lambda dt & (2.6) \\ &= \exp(-\lambda t) \cdot \lambda dt & (2.7) \end{aligned}$$

□

Teorema 2 (incremente independente). *Procesul de numarare corespunzator unui proces Poisson satisface*

$$P\{Y(t) = n\} = \frac{(\lambda t)^n}{n!} \cdot e^{-\lambda t} \tag{2.8}$$

iar numarul de sosiri din intervale disjuncte de timp sunt statistic independente.

Demonstraţie. Ultima afirmatie decurge din faptul ca sosirile sunt independente, conform definitiei.

Pentru a demonstra ecuatia 2.8 considerati iarasi ca intervalul t ca fiind format din t/dt intervale elementare dt. Probabilitatea ca in n dintre acestea sa avem cate o sosire si in restul de $t/dt - n$ sa nu avem sosiri este:

$$p = (1 - \lambda t)^{\frac{t}{dt} - n} \cdot (\lambda dt)^n \tag{2.9}$$

In plus, cele n sosiri pot veni la momente diferite. Numarul de combinatii posibile de momente este:

$$N = \binom{\frac{t}{dt}}{n} \tag{2.10}$$

Probabilitatea totala va fi

$$P = pN \tag{2.11}$$

Prelucram acum pe rand expresiile pentru p si pentru N:

$$\begin{aligned} p &= (1 - \lambda t)^{\frac{-1}{\lambda t}\left[-\lambda t\left(\frac{t}{dt} - n\right)\right]} \cdot \lambda^n dt^n && (2.12) \\ &= \exp(-\lambda t + n\lambda dt) \cdot \lambda^n dt^n && (2.13) \end{aligned}$$

Observam ca $-\lambda t + n\lambda dt \simeq -\lambda t$ asa incat:

$$p = \exp(-\lambda t) \cdot \lambda^n dt^n \tag{2.14}$$

Probabilitatea aceasta este *foarte* mica. Ar trebui ca numarul de variante posibile N sa fie foarte mare pentru a obtine o probabilitate finita.

$$\begin{aligned}
N &= \frac{\Gamma\left(\frac{t}{dt}+1\right)}{\Gamma\left(\frac{t}{dt}-n+1\right)\Gamma(n+1)} && (2.15)\\
&= \frac{\left(\frac{t}{dt}-n+1\right)\left(\frac{t}{dt}-n+2\right)\cdots\left(\frac{t}{dt}-n+n\right)}{n!} && (2.16)\\
&\simeq \frac{\left(\frac{t}{dt}\right)^n}{n!} && (2.17)\\
&= \frac{t^n}{n!}\frac{1}{dt^n} && (2.18)
\end{aligned}$$

Si intr-adevar:

$$P = pN = \frac{(\lambda t)^n}{n!} \cdot \exp(-\lambda t) \tag{2.19}$$

□

In figurile 2.1 si 2.2 este ilustrat comportamentul functiei din ecuatia 2.8.

Teorema 3 (superpozitia proceselor Poisson). *Superpozitia a doua procese Poisson este tot un proces Poisson.*

Demonstraţie. Fie un proces Poisson caracterizat de probabilitatea elementara $\lambda_1 dt$ si un alt proces Poisson caracterizat de probabilitatea $\lambda_2 dt$. Prin superpozitia celor doua se obtine un proces care, pentru fiecare interval elementar, e caracterizat de o probabilitate de sosire egala cu $(\lambda_1 + \lambda_2)dt$. Ca urmare este la randul sau un proces Poisson. □

Teorema 4 (Palm). *Superpozitia unui numar foarte mare de trafice oarecare cu rate de acelasi ordin de marime tinde catre un proces Poisson.*

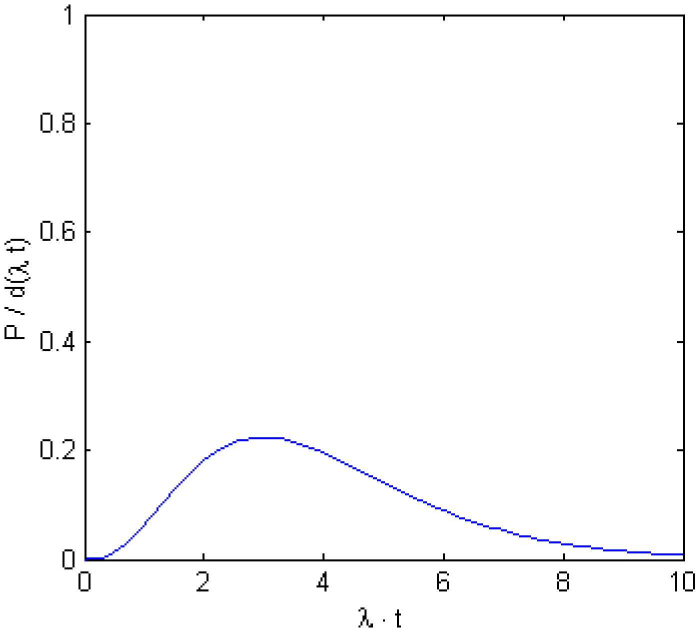

Figura 2.1: Variatia probabilitatii cu produsul (timp-intensitate trafic) pentru un numar fixat de sosiri $n = 2$ la un proces poisson.

2.2.2 Procese Bernoulli

Procesele Bernoulli sunt analogul in timp discret al proceselor Poisson. Probabilitatea de a avea o sosire la momentul T_i este p oricare ar fi i. Ca urmare numarul de sosiri de la T_0 pana la T_k este:

$$P\{Y_k = n\} = \binom{k}{n} p^n (1-p)^{k-n} \tag{2.20}$$

Comportamentul functiei din ecuatia 2.20 este ilustrat in figurile 2.3, 2.4 si 2.5.

Timpul dintre sosiri are o distributie geometrica cu parametru p:

$$P\{A_k = n\} = p(1-p)^n \tag{2.21}$$

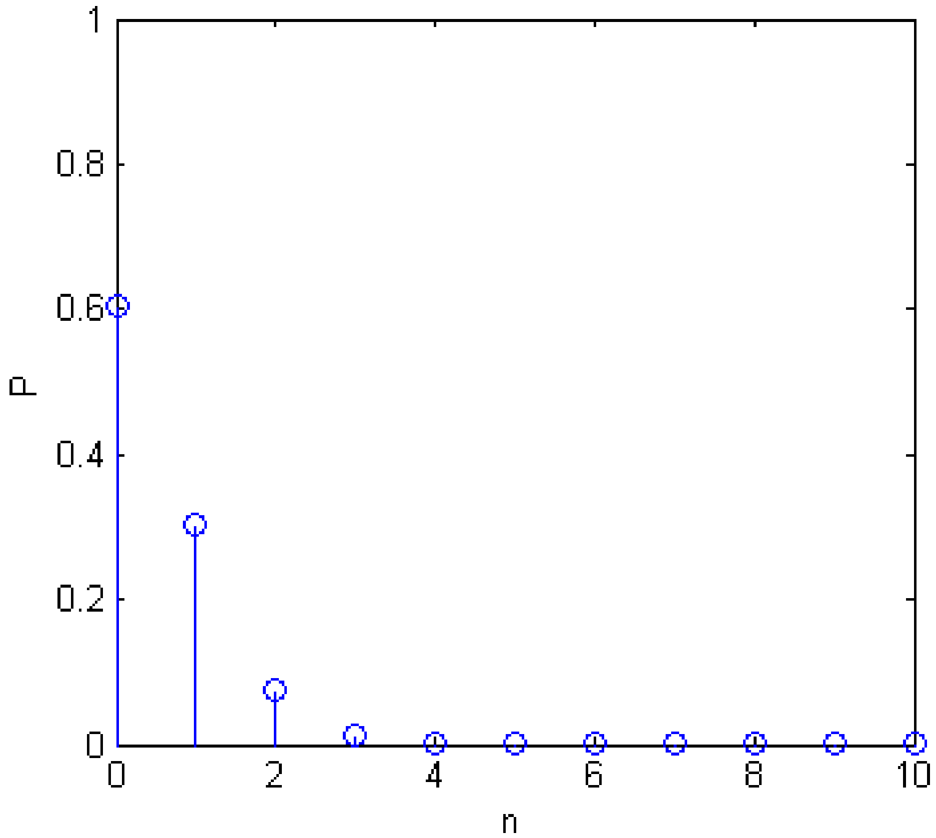

Figura 2.2: Variatia probabilitatii cu numarul de sosiri pentru un produs (timp-intensitate trafic) fixat $\lambda t = 0.5$ la un Proces Poisson.

Teorema 5 (superpozitia proceselor Bernoulli). *Superpozitia a doua procese Bernoulli este tot un proces Bernoulli.*

2.3 Modele Markov

In continuare sunt descrise doar cele mai simple dintre modelele Markov si anume procesele Poisson modulate Markov.

In modelele Markov sosirile sunt modelate cu ajutorul unui automat probabilist ce poate fi reprezentat printr-un graf ca in figura 2.6. Modelul din

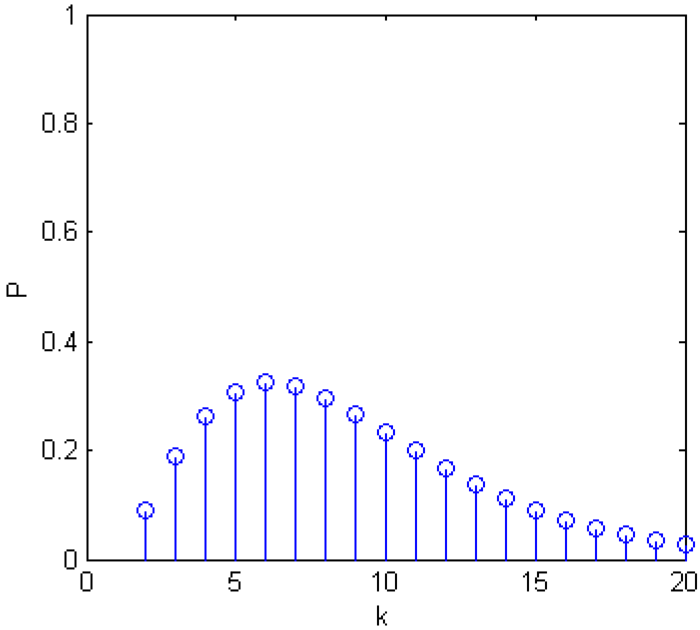

Figura 2.3: Probabilitatea de a avea $n = 2$ sosiri pentru $p = 0.3$ in functie de timpul total de asteptare k la un proces Bernoulli.

figura are 5 stari numerotate de la 0 la 4. Tranzitia de la starea i la starea j este etichetata cu probabilitatea $p_{i,j}$ exprimata in procente. Lipsa unei tranzitii de la i la j este echivalenta cu $p_{i,j} = 0$. Este respectata proprietatea:

$$\sum_{j=0}^{N-1} p_{i,j} = 1, \forall i \tag{2.22}$$

Pentru o realizare particulara comportamentul acestui model este:

- modelul sta in starea i un timp t_i care este o realizare particulara a unei variabile aleatore distribuita exponential (vezi ecuatia 2.4) de parametru λ_i ce depinde numai de i
- la expirarea timpului t_i modelul trece in starea j cu probabilitatea $p_{i,j}$

Fiecare tranzitie corespunde unei sosiri. Avantajul acestor modele fata de cele de reinnoire este ca, prin schimbarea de la un moment la altul

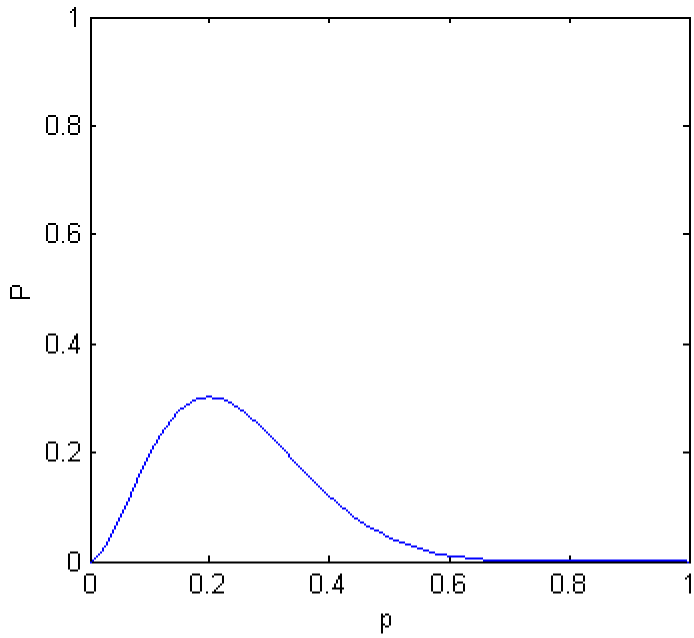

Figura 2.4: Probabilitatea de a avea $n = 2$ sosiri in $k = 10$ intervale de timp in functie de intensitatea traficului exprimata de probabilitatea p la un proces Bernoulli.

a parametrului λ_i, introduc o dependenta temporala care corespunde unei functii de autocorelatie mai complexa decat cea de la modelele de reinnoire (ecuatia 2.2). Astfel exista posibilitatea unei potriviri mai bune pe datele experimentale.

O alta reprezentare (mai compacta dar mai putin intuitiva) a aceluiasi proces Markov se poate da specificandu-se matricea probabilitatilor de tranzitie si vectorul parametrilor λ_i corespunzatori starilor. De exemplu pentru graful din figura 2.6 matricea de tranzitie este:

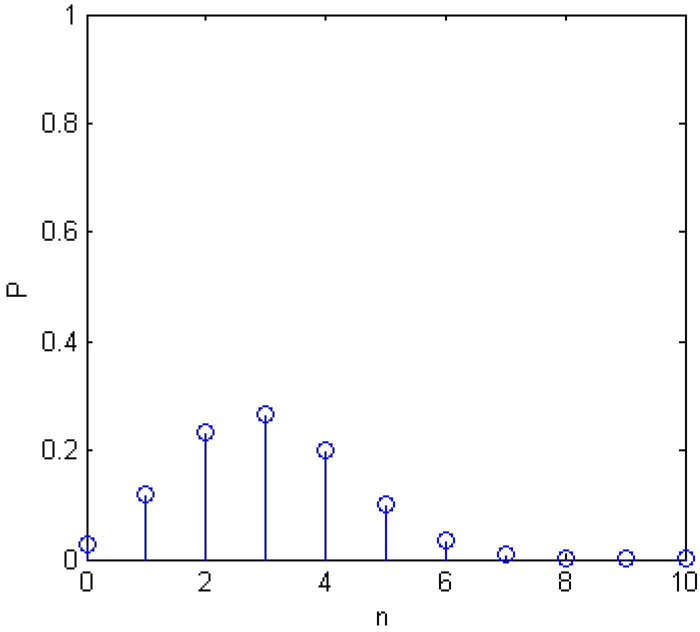

Figura 2.5: Probabilitatea ca in $k = 10$ intervale de timp la o intensitate $p = 0.3$ sa apara n sosiri la un proces Bernoulli.

$$M = \begin{bmatrix} 0 & 0.7 & 0 & 0.3 & 0 \\ 0.5 & 0 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0.8 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.23)$$

2.4 Traficul ca fluid

Este util sa se modeleze traficul ca fluid in situatia in care unitatiile de trafic sunt numeroase in comparatie cu scara de timp folosita pentru analiza. In aceste modele sursele sunt de tipul ON/OFF: fie emit cu o rata constanta λ fie nu emit deloc. Duratele ON si OFF sunt distribuite exponential si independente.

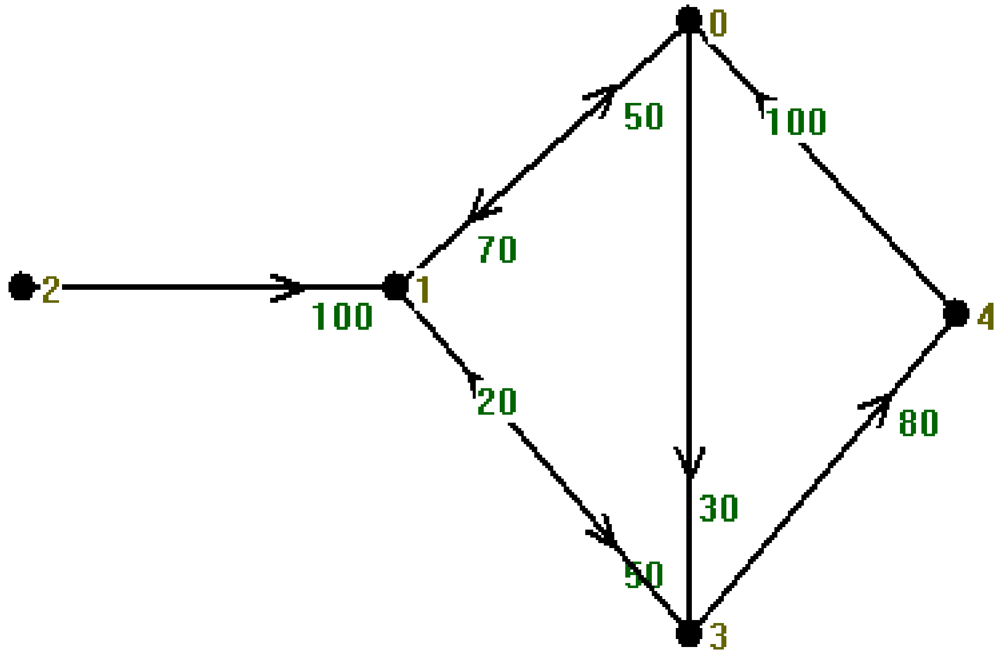

Figura 2.6: Un exemplu de proces Markov.

Ca urmare numararea unitatilor de trafic este inlocuita de masurarea volumului de trafic. Analiza matematica se face folosind procese Poisson modulate Markov sau alte modele Markov.

Aceste metode sunt potrivite mai ales in situatii in care unitatiile de trafic sunt mici in comparatie cu volumul total de trafic (de exemplu la ATM). Principalul avantaj este ca prin ignorarea caracterului discret al traficului se castiga viteza de calcul.

2.5 Modele de tip autoregresiv

Modelele de tip autoregresiv introduc o dependenta explicita liniara a traficului la un moment dat de traficul in momentele anterioare. Ele sunt foarte potrivite pentru traficul video comprimat, la care se transmit diferente fata de cadrul anterior.

2.5.1 Procese liniar autoregresive (AR)

Modelele autoregresive AR(p) sunt definite de ecuatia:

$$X_n = \sum_{r=1}^{p} a_r X_{n-r} + \epsilon_n, n > 0 \tag{2.24}$$

unde $(X_{-p+1}, \ldots, X_0)$ este un vector de variabile aleatoare dat, a_r, $1 \leq r \leq p$, sunt constante reale si ϵ_n sunt variabile aleatoare de medie zero si necorelate (zgomot alb). In mod obisnuit variabilele ϵ_n au valori mici comparativ cu X_n.

Ecuatia 2.24 este o ecuatie cu diferente finite si ca urmare se poate folosi transformata Z pentru a da o alta exprimare. Nu uitati totusi ca acum transformata Z se aplica unui proces stocastic si nu unui sir. Pentru ecuatia 2.24 imaginea in domeniul Z este:

$$X(z) = X(z) \sum_{r=1}^{p} a_r z^{-r} + \epsilon(z) \tag{2.25}$$

$$X(z) = \frac{\epsilon(z)}{1 - \sum_{r=1}^{p} a_r z^{-r}} \tag{2.26}$$

Din ecuatia 2.26 se vede ca un semnal din clasa AR(p) se obtine la iesirea unui filtru digital cu p poli si nici un zero atunci cand la intrare se aplica zgomot alb.

2.5.2 Procese de mediere glisantă (MA)

Clasa[1] MA(q) este definita de ecuatia:

$$X_n = \sum_{r=0}^{q} b_r \epsilon_{n-r}, n > 0 \tag{2.27}$$

[1]MA = moving average

unde b_r,$0 \le r \le q$, sunt constante reale iar ϵ_n sunt variabile aleatoare de medie zero si necorelate.

In domeniul Z ecuatia 2.27 se scrie:

$$X(z) = \epsilon(z) \sum_{r=0}^{q} b_r z^{-r} \tag{2.28}$$

Un proces MA se obtine la iesirea unui filtru a carui functie de transfer are numai zerouri (deci are raspuns finit la impuls) si la a carui intrare se aplica zgomot alb.

2.5.3 Procese ARMA

Clasa ARMA(p, q) este definita de ecuatia:

$$X_n = \sum_{r=1}^{p} a_r X_{n-r} + \sum_{r=0}^{q} b_r \epsilon_{n-r} \tag{2.29}$$

In general daca se foloseste pentru modelare un proces ARMA(p, q) in loc de un proces simplu fie AR(p), fie MA(q) se pot lua valori mai mici pentru p si q la aceeasi exactitate a potrivirii pe datele experimentale. Altfel spus procesele ARMA ofera o flexibilitate mai mare pentru un acelasi ordin.

In domeniul Z ecuatia de definitie 2.29 devine:

$$X(z) = X(z) \sum_{r=1}^{p} a_r z^{-r} + \epsilon(z) \sum_{r=0}^{q} b_r z^{-r} \tag{2.30}$$

$$X(z) = \frac{\epsilon(z) \sum_{r=0}^{q} b_r z^{-r}}{1 - \sum_{r=1}^{p} a_r z^{-r}} \tag{2.31}$$

Ca urmare un proces ARMA se obtine la iesirea unui filtru liniar ce are atat zerouri cat si poli si la a carui intrare se aplica zgomot alb.

2.5.4 Procese ARIMA

Procesele ARIMA(p, d, q) sunt procese X_n pentru care diferenta de ordinul d satisface ecuatia 2.29. Aceasta ultima afirmatie se scrie in domeniul Z, tinand cont de equatia 2.31 astfel:

$$(1 - z^{-1})^d X(z) = \frac{\epsilon(z) \sum_{r=0}^{q} b_r z^{-r}}{1 - \sum_{r=1}^{p} a_r z^{-r}} \tag{2.32}$$

$$X(z) = \frac{\epsilon(z) \sum_{r=0}^{q} b_r z^{-r}}{\left(1 - \sum_{r=1}^{p} a_r z^{-r}\right)(1 - z^{-1})^d} \tag{2.33}$$

(2.34)

Un avantaj al modelelor ARMA si ARIMA este ca se cunosc metode statistice bune de estimare a parametrilor. Un dezavantaj este ca pentru orice valori ale parametrilor autocorelatia are o descrestere asimptotic geometrica catre infinit, adica $\rho(n) \sim r^n$ pentru un $0 < r < 1$ atunci cand $n \to 1$. In figurile 2.7 si 2.8 este ilustrat comportamentul autocorelatiei unui proces ARMA. Acesta a fost generat prin aplicarea de zgomot alb la intrarea unui filtru cu functia de transfer:

$$X(z) = \frac{1 - z^{-1} + z^{-2}}{1 - z^{-1} + 0.16 z^{-2}} \tag{2.35}$$

Functia de autocorelatie a zgomotului si a semnalului de la iesirea filtrului sunt aratate in figura 2.7 iar in figura 2.8 este ilustrat comportamentul asimptotic al functiei de autocorelatie a procesului ARMA. Pe ordonata este reprezentat $\log|\rho_n|$ cu albastru si cu rosu este desenat comportamentul pentru $\rho_n = cr^n$.

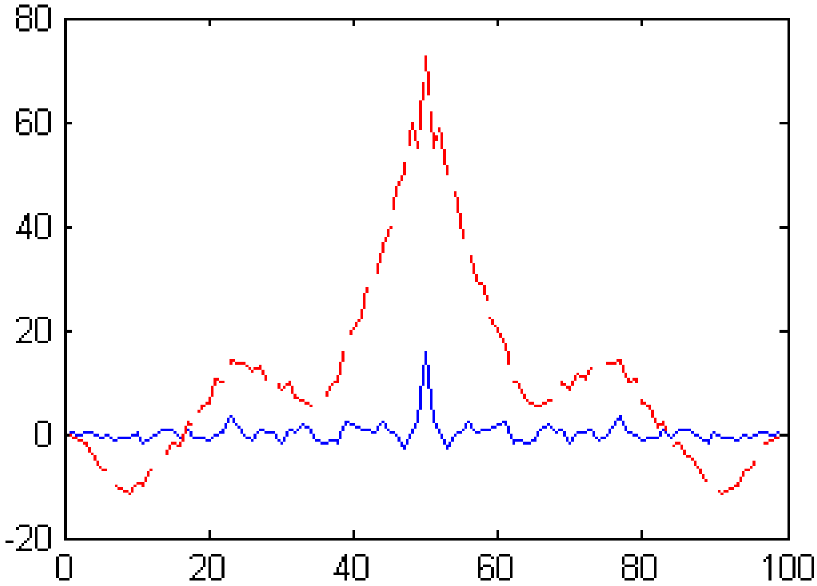

Figura 2.7: Functia de autocorelatie a unui zgomot alb (linie albastra) si a semnalului de la iesirea filtrului (linie rosie)

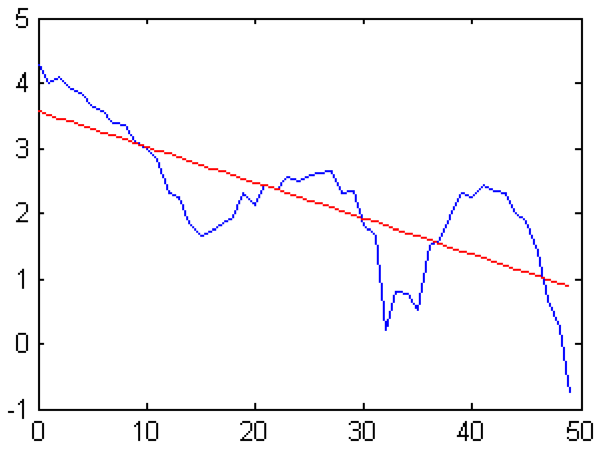

Figura 2.8: Comportarea asimptotica a functiei de autocorelatie a unui proces ARMA

Capitolul 3

Procese stocastice autosimilare

Procesele stocastice autosimilare sunt modele matematice utilizate in multe domenii: hidrologie, economie si finante, fizica[1]. Prezentarea din acest capitol urmeaza linia generala din [20].

O realizare particulara a unui proces stocastic real este o functie reala. Fiecare realizare particulara are asociata o probabilitate de aparitie, care poate fi infinitezimala. Un proces stocastic este o multime de realizari particulare ale caror probabilitati insumate (sau integrate) fac 1. Prin esantionarea unui proces stocastic ("la un moment de timp") se obtine o variabila aleatoare. O realizare particulara a unei variabile aleatoare este un numar.

3.1 Definiţie

Procesele stocastice autosimilare reprezinta un tip aparte de procese stocastice.

Definiţia 7 (proces stocastic autosimilar). *Un proces stocastic* $\{Y(t), t \geq 0\}$ *se numeste* ***autosimilar*** *daca pentru orice* $a > 0$ *exista* $b > 0$ *astfel incat*

$$Y(at) \doteq bY(t) \tag{3.1}$$

[1] vezi notiunea de renormalizare din fizica statistica si din fizica energiilor inalte

Notatia $\doteq$ semnifica egalitatea tuturor distributiilor de probabilitate finite. Semnul $=$ intre doua variabile aleatoare sau intre doua procese stocastice inseamna ca pentru orice eveniment realizarile particulare ale celor doua entitati sunt egale.

Definiţia 8 (continuitate stocastica). *Un proces stocastic* $\{Y(t), t \geq 0\}$ *este continuu stocastic in* t *daca pentru orice* $\epsilon > 0$ *avem ca*

$$\lim_{h \to 0} P\{|Y(t+h) - Y(t)|\} = 0 \tag{3.2}$$

Definiţia 9 (proces stocastic trivial). *Un proces stocastic se numeste trivial daca are o singura realizare particulara.*

Teorema 6 (unicitatea exponentului Hurst). *Daca* $\{Y(t), t \geq 0\}$ *nu este trivial, este continuu stocastic in* $t = 0$ *si este autosimilar, atunci exista si este unic exponentul* $H \geq 0$ *astfel incat* $b = a^H$*. In plus* $H > 0$ *daca si numai daca* $Y(0) = 0$*.*

S-a observat ca traficul cumulat dintr-o retea de telecomunicatii poate fi bine modelat de procese autosimilare. In general se considera ca intensitatea traficului corespunde unui proces stocastic stationar. Se spune despre un proces stocastic autosimilar ca are incremente stationare daca procesul stocastic $\{Y(h+t) - Y(t)\}$ este stationar pentru orice valoare a parametrului h. Pentru modelarea intensitatii traficului se foloseste procesul stocastic:

$$X_n = Y(n+1) - Y(n), n \in \mathbf{N} \tag{3.3}$$

3.2 Autocorelaţia

3.2.1 Utilizarea autocorelatiei pentru identificare

In general pentru a estima daca o realizare particulara provine de la un proces stocastic de tipul A sau de la un proces stocastic de tipul B trebuie gasita o estimare care se aplica ambelor tipuri dar care pentru A trebuie sa duca la un rezultat de un anumit tip, iar pentru B trebuie sa duca la un rezultat de alt tip. Ca o ilustrare sa vedem o metoda prin care se pot deosebi realizari particulare ale incrementelor unui proces autosimilar de realizari particulare ale unui proces ARMA. Pe baza unei realizari particulare se estimeaza functia de autocorelatie. Aceasta estimare pleaca de la presupunerea ca realizarea particulara apartine unui proces ergodic, asa incat este potrivita pentru ambele tipuri de procese. Comportamentul asimptotic la infinit al functiei de autocorelatie pentru un proces ARMA este $\rho_n = r^n$, $0 < r < 1$. Pentru incrementele unui proces autosimilar insa comportamentul este altul si in acest fel se poate face distinctia.

Sa vedem care este comportamentul functiei de autocorelatie

$$\rho_n = \mathrm{E}[X_0 X_n], n \in \mathbf{N} \tag{3.4}$$

Teorema 7 (autocorelatia unui proces autosimilar). *Daca* $\{Y(t)\}$ *este un proces netrivial, autosimilar cu* $H > 0$*, are incremente stationare si* $E[Y^2(1)] < \infty$ *atunci:*

$$E[Y(t)Y(s)] = \frac{t^{2H} + s^{2H} - |t-s|^{2H}}{2} E[Y^2(1)] \tag{3.5}$$

Demonstraţie.

$$\begin{aligned}
2\mathrm{E}[Y(t)Y(s)] &= \mathrm{E}[2Y(t)Y(s)] && (3.6)\\
&= \mathrm{E}[Y^2(t) + Y^2(s) - Y^2(t) - Y^2(t) + 2Y(t)Y(s)] && (3.7)\\
&= \mathrm{E}[Y^2(t)] + \mathrm{E}[Y^2(s)] - \mathrm{E}[(Y(t) - Y(s))^2] && (3.8)\\
&= \mathrm{E}[Y^2(t)] + \mathrm{E}[Y^2(s)] - \mathrm{E}[(Y(|t-s|) - Y(0))^2] && (3.9)\\
&= \mathrm{E}[t^{2H}Y^2(1)] + \mathrm{E}[s^{2H}Y^2(1)] - \mathrm{E}[|t-s|^{2H}Y^2(1)] && (3.10)\\
&= \{t^{2H} + s^{2H} - |t-s|^{2H}\}\mathrm{E}[Y^2(1)] && (3.11)
\end{aligned}$$

NOTA: Conform teoremei 6 avem ca $Y(0) = 0$ □

Putem in sfarsit cum ar trebui sa se comporte autocorelatia intensitatii traficului daca traficul cumulat este un proces aleator.

Teorema 8. *Autocorelatia incrementelor unui proces autosimilar are comportamentul:*

$$\rho_n \begin{cases} \sim H(2H-1)n^{2H-2}E[Y^2(1)], n \to \infty, \ \ daca \ H \neq 0.5 \\ = 0, n \geq 1, \ \ daca \ H = 0.5 \end{cases} \tag{3.12}$$

Demonstrație. Pentru $n = 0$ functia de autocorelatie este:

$$\rho_0 = \mathrm{E}[X_0^2] = \mathrm{E}[Y^2(1)] = \mathrm{const} \tag{3.13}$$

Pentru $n \geq 1$ functia de autocorelatie este:

$$\rho_n = \mathrm{E}[X_0 X_n] = \mathrm{E}[Y(1)\{Y(n+1) - Y(n)\}] \tag{3.14}$$

$$= \mathrm{E}[Y(1)Y(n+1)] - \mathrm{E}[Y(1)Y(n)] \tag{3.15}$$

$$= \frac{1}{2}\big\{[1 + (n+1)^{2H} - n^{2H}] - [1 + n^{2H} - (n-1)^{2H}]\big\}\mathrm{E}[Y^2(1)] \tag{3.16}$$

$$= \frac{1}{2}\big\{(n+1)^{2H} - 2n^{2H} + (n-1)^{2H}\big\}\mathrm{E}[Y^2(1)] \tag{3.17}$$

Daca $H = 0.5$ atunci $\rho_n = 0$, pentru $n \geq 1$.

Pentru o pseudodemonstratie a comportamentului asimptotic introducem functia $f(x) = \frac{1}{2}x^{2H}$. Observam ca

$$f''(x) = \lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h} \tag{3.18}$$

$$\rho_n = \frac{f(x+h) - 2f(x) + f(x-h)}{h}\Big|_{x=n, h=1} \cdot \mathrm{E}[Y^2(1)] \tag{3.19}$$

Dar derivata a doua a functiei $f(x)$ este usor de calculat:

$$f''(x) = H(2H-1)x^{2H-2} \tag{3.20}$$

De aici rezulta afirmatia din teorema. $\square$

Se poate asadar observa comportamentul diferit al functiei de autocorelatie pentru $n \to \infty$: pentru procese ARMA este r^n cu $0 < r < 1$, iar pentru incrementele unui proces autosimilar este n^β cu $-2 < \beta < 0$. Se spune ca in cazul proceselor ARMA autocorelatia scade geometric, pe cand in cazul proceselor autosimilare scade exponential. Altfel spus la n mare graficul $(\rho_n; n)$ al unui proces ARMA (sau ARIMA) se potriveste pe o dreapta daca ordonata este la scara logaritmica, iar graficul $(\rho_n; n)$ al incrementelor unui proces autosimilar se potrivesc pe o dreapta daca atat ordonata cat si abscisa sunt reprezentate la scara logaritmica. In figura 3.1 este reprezentata functia de autocorelatie data de ecuatia 3.17 pentru doua valori diferite ale parametrului Hurst. Important este comportamentul asimptotic la n mare.

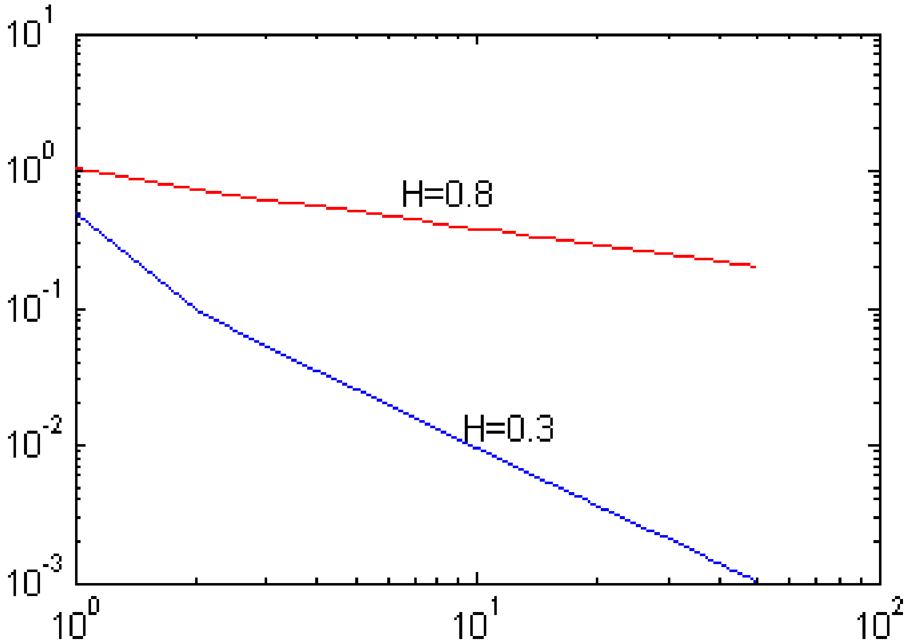

Figura 3.1: Functia de autocorelatie pentru incrementele unui proces stocastic autosimilar.

3.2.2 Legatura cu dependenţa pe perioade mari

Procesele cu dependenta pe perioade mari (LRD[2]) sunt adeseori confundate cu procesele autosimilare insa ele constituie o clasa diferita.

[2]Long Range Dependance

Definiţia 10 (LRD). *Procesul aleator* $\{X_n\}$ *se numeste LRD daca autocorelatia sa nu este absolut sumabila:*

$$\sum_n |\rho_n| = \infty \tag{3.21}$$

Legatura dintre procesele autosimilare si cele LRD este data de urmatoarea teorema.

Teorema 9. *Daca* $\{Y(t)\}_{t\geq 0}$ *este un proces autosimilar netrivial cu incremente stationare si* $X_n = Y(n+1) - Y(n)$*, iar* $\rho_n = E[X_0 X_n]$ *atunci:*

1. $0 < H < 0.5 \Rightarrow \sum_{n=0}^{\infty} |\rho_n| < \infty$

2. $H = 0.5 \Rightarrow \{X_n\}$ *este necorelat*

3. $0.5 < H < 1 \Rightarrow \sum_{n=0}^{\infty} |\rho_n| = \infty$

3.3 Densitatea spectrală de putere

Plecand de la ecuatia 3.17 se poate calcula (cel putin numeric) densitatea spectrala de putere a procesului $X(t)$ pentru diferite valori ale lui H. In figurile 3.2, 3.3, 3.4 si 3.5 sunt reprezentate graficele obtinute in acest fel pentru $H = 0.3$ (SRD) si $H = 0.8$ (LRD).

Se poate observa ca la frecvente mici graficul log-log seamana foarte mult cu o dreapta. De fapt se poate arata ca, asimptotic, densitatea spectrala de putere se comporta astfel:

$$f(\lambda) \sim |1 - e^{\imath\lambda}|^{1-2H} \tag{3.22}$$

$$\sim \lambda^{1-2H} \tag{3.23}$$

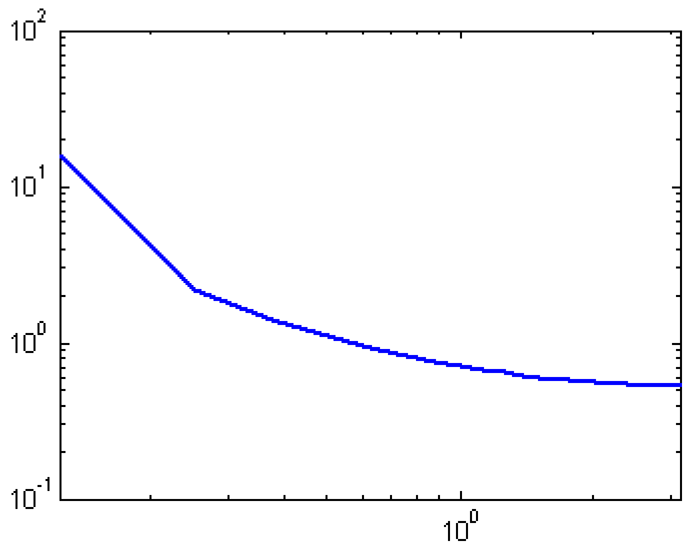

Figura 3.2: Partea reala a densitatii spectrale de putere pentru $H = 0.8$

3.4 Renormalizarea

Notiunea de renormalizare este mai ales utilizata in fizica. Ea este insa relevanta din punctul de vedere al metodei varianta-agregare de estimare a parametrului H. Pentru o secventa $\{X_n\}_{n\geq 0}$ de variabile aleatoare se defineste operatia de renormalizare astfel:

$$T(N,H)X = \left\{\big(T(N,H)X\big)_n\right\} \tag{3.24}$$

$$\big(T(N,H)X\big)_n = \frac{1}{N^H}\sum_{k=nN}^{(n+1)N-1} X_k \tag{3.25}$$

Secventa $\{T(N,H), N \geq 1\}$ formeaza un semigrup multiplicativ numit grupul de renormalizare de index H. Aceasta deoarece $T(N,H)T(M,H) = T(NM,H)$. Incrementele unui proces stocastic autosimilar cu parametrul Hurst H sunt puncte fixe pentru transformarile din acest grup. Acest lucru ne spune cum va varia estimarea variantei in functie de gradul de agregare.

Sa luam intai cazul zgomotului alb pentru a vedea mai bine diferenta.

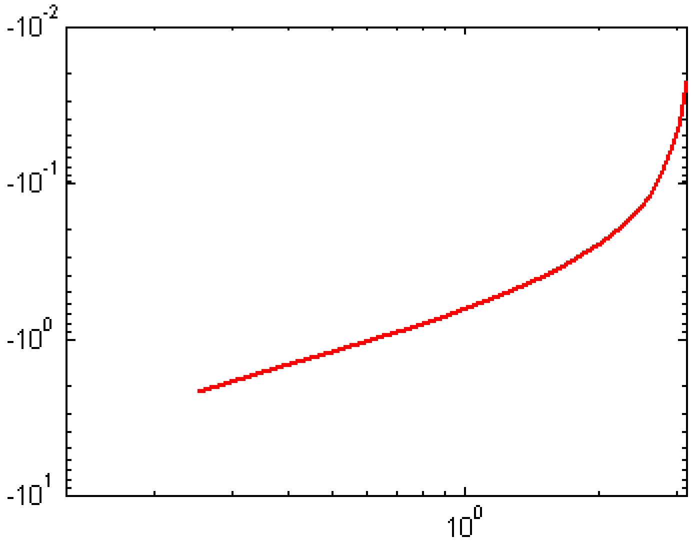

Figura 3.3: Partea imaginara a densitatii spectrale de putere pentru $H = 0.8$

Asadar consideram ca variabilele $X_1, X_2, \ldots$ sunt identic distribuite si independente. In acest caz avem ca:

$$\mathrm{Var}\Big[\frac{1}{N}(X_1 + X_2 + \ldots + X_N)\Big] = \frac{1}{N^2}\big(\mathrm{Var}[X_1] + \ldots + \mathrm{Var}[X_N]\big) \tag{3.26}$$

$$= \frac{1}{N}\sigma^2 \tag{3.27}$$

Asadar ne asteptam ca graficul varianta-agregare in acest caz sa aiba forma $\sim N^{-1}$.

In cazul proceselor autosimilare insa ne asteptam la o alta forma. Faptul ca incrementele unui proces autosimilare sunt puncte fixe pentru $T(N, H)$ ne spune ca $(T(N, H)X)_j$ are aceeasi distributie de probabilitate ca si X_j:

$$(T(N, H)X)_j \doteq X_j \tag{3.28}$$

In cazul general procesul cu gradul de agregare m este definit astfel:

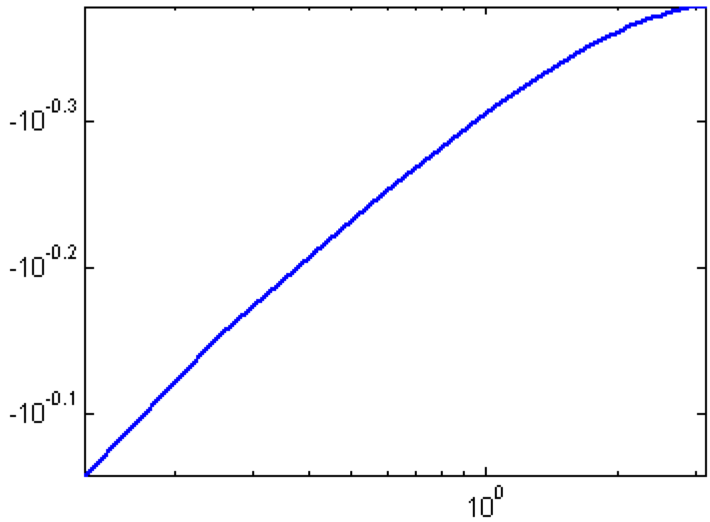

Figura 3.4: Partea reala a densitatii spectrale de putere pentru $H = 0.3$

$$X_j^{(m)} = \frac{1}{m} \sum_{k=jm}^{(j+1)m-1} X_j \tag{3.29}$$

Asadar stim ca:

$$(T(N,H)X)_j = m^{1-H} X_j^{(m)} \doteq X_j \tag{3.30}$$

Daca distributiile sunt aceleasi atunci si variantele sunt aceleasi si deci:

$$\mathrm{Var}[X_j^{(m)}] = m^{-2(1-H)} \cdot \mathrm{Var}[X_j] \tag{3.31}$$

Aceasta comportare a variantei cu gradul m de agregare poate fi folosita la estimarea parametrului Hurst si pentru a decide daca o realizare particulara este a unui proces autosimilar sau nu.

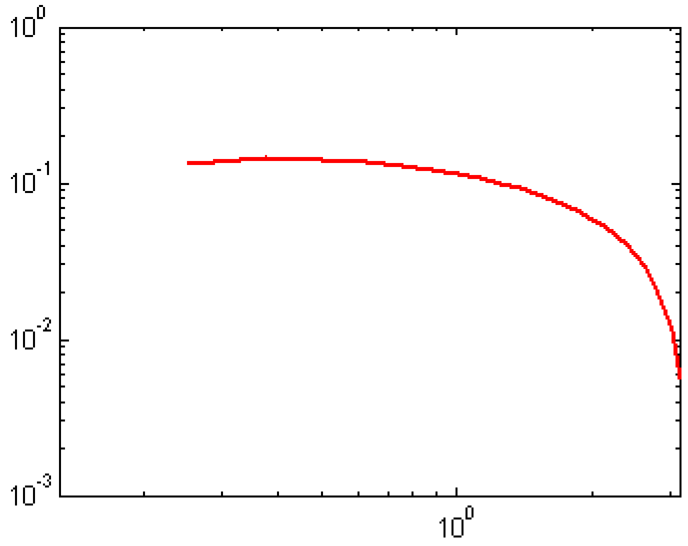

Figura 3.5: Partea imaginara a densitatii spectrale de putere pentru $H = 0.3$

Capitolul 4

Estimarea parametrului Hurst

In acest capitol sunt prezentate diverse tehnici statistice de estimare a parametrului Hurst care au, in general, doua etape:

1. Estimarea prin metode clasice a unei functii ce descrie procesul stocastic. Exemple de astfel de functii sunt autocorelatia, densitatea spectrala de putere si varianta ca functie de gradul de agregare.

2. Prin alegerea parametrului Hurst se potriveste forma tipica a functiei respective pentru procese autosimilare pe rezultatul estimarii anterioare.

In continuare vor fi prezentate, urmand linia generala din [31], metodele:

- Metoda R/S
- Analiza varianta-timp
- Metoda Higuchi
- Metoda corelogramei
- Metoda periodogramei
- Estimatorul Whittle

- Metoda Veitch-Abry

Dintre aceste metode implementate in RTH sunt: analiza varianta-timp si metoda periodogramei. Codul MATLAB utilizat se gaseste in anexa A.

4.1 Metoda R/S

Aceasta metoda a fost propusa chiar de catre Hurst intr-o lucrare de hidrologie intitulata Capacitatea pe termen lung a rezervoarelor din 1951. Fie $\{Y_i\}$ un proces de numarare in timp discret care corespunde traficului. Diferenta de ordinul 1 a acestui proces corespunde intensitatii traficului:

$$X_i = Y_i - Y_{i-1} \tag{4.1}$$

Din cele n esantioane ale intensitatii traficului estimam deviatia standard astfel:

$$\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{4.2}$$

$$\hat{\gamma}_n(k) = \frac{1}{n}\sum_{i=1}^{n-k}(X_i - \hat{\mu}_n)\cdot(X_{i+k} - \hat{\mu}_n) \tag{4.3}$$

$$\hat{\sigma}_n^2 = \hat{\gamma}_n(0) \tag{4.4}$$

Apoi se calculeaza statistica RS:

$$\hat{\mathrm{RS}}(n) = \frac{1}{\hat{\sigma}_n}\Big[\max_{0\le j\le n}\{Y_j - \frac{j}{n}Y_n\} - \min_{0\le j\le n}\{Y_j - \frac{j}{n}Y_n\}\Big] \tag{4.5}$$

Sa vedem de ce aceasta statistica da o masura a neregularitatii datelor masurate. Termenul $Y_j - j/n \cdot Yn$ reprezinta diferenta dintre traficul pana la momentul j si traficul pe care l-am fi asteptat pentru o intensitate constata. Statistica RS este proportionala cu diferenta dintre valoarea maxima a acestui termen (corespunzatoare momentului cand traficul este mult peste asteptari) si valoarea minima a acestui termen (corespunzatoare momentului cand traficul este mult sub asteptari). Constanta de proportionalitate $1/\hat{\sigma}_n$ este cu atat mai mare cu cat intensitatea traficului are o varianta mai mica.

Ei bine, daca procesul de numarare este autosimilar comportamentul asimptotic la infinit ($n \to \infty$) al acestei statistici este:

$$\hat{\mathrm{RS}}(n) \sim cn^H \tag{4.6}$$

Asadar graficul $(\log n; \log \hat{\mathrm{RS}}(n))$ este o dreapta cu panta H. Ca urmare pentru estimarea parametrului Hurst se poate utiliza regresia liniara.

4.2 Analiza varianţă-timp

Daca un proces aleator este privit la o scara de timp mai mare el pare in general mai putin variat. De fapt daca are o autocorelatie de tip impuls varianta are asimptotic comportamentul $\mathrm{Var}[X^{(m)}] \sim \mathrm{Var}[X] \cdot m^{-2(1-H)}$, unde $X^{(m)}$ este:

$$X_k^{(m)} = \sum_{i=mk}^{m(k+1)-1} X_i \tag{4.7}$$

Se poate asadar utiliza regresia liniara pe graficul $(\log \mathrm{Var}[X^{(m)}]; \log m)$ pentru a determina parametrul Hurst.

4.3 Metoda Higuchi

Aceasta metoda presupune calcularea "lungimii normalizate":

$$L(m) = \frac{n-1}{m^3} \sum_{i=1}^{m} \lfloor \frac{n-i}{m} \rfloor^{-1} \sum_{k=1}^{\lfloor \frac{n-i}{m} \rfloor} |Y_{i+km} - Y_{i+(k-1)m}| \tag{4.8}$$

Pentru procese autosimilare aceasta variaza conform relatiei $L(m) \sim cm^{H-2}$. Se poate asadar utiliza regresia liniara pe graficul $(\log L(m); \log m)$ pentru a determina parametrul Hurst.

4.4 Metoda corelogramei

In aceasta metoda intai este estimata autocorelatia intensitatii traficului conform formulei 4.3. Apoi se foloseste faptul ca pentru trafice autosimilare comportamentul asimptotic al acesteia la infinit este $\rho(n) \sim cn^{-2(1-H)}$. Si aici se poate folosi o regresie liniara.

4.5 Metoda periodogramei

In acest caz se face intai o estimare spectrala a intensitatii traficului folosind o periodograma cu ponderare data de un nucleu Dirichlet:

$$I(\lambda) = |d(\lambda)|^2 \tag{4.9}$$

$$d(\lambda) = \frac{\sum_{t=1}^{n} h_t X_t e^{\imath t \lambda}}{\sqrt{2\pi \sum_{t=1}^{n} |h_t^2|}} \tag{4.10}$$

$$h_t = \left(1 - e^{\frac{\imath 2\pi t}{n}}\right)^p \tag{4.11}$$

unde p este denumit “ordinul nucleului”.

Pentru un proces autosimilar LRD forma densitatii spectrale de putere la frecvente joase este:

$$I(\lambda) = C|1 - exp(\imath\lambda)|^{1-2H} \tag{4.12}$$

Ca urmare se poate folosi regresia neliniara pe graficul $(\log I(\lambda); \lambda)$ pentru a determina parametrul Hurst.

Daca se doreste sa se tina cont de SRD[1] atunci se poate face o potrivire neliniara pe:

[1] Short Range Dependance

$$I(\lambda) = C|1 - exp(\imath\lambda)|^{1-2H} \exp(a\lambda + b\lambda^2) \tag{4.13}$$

Capitolul 5

Programul RTH

Programul *Real Time Hurst* afiseaza parametrul Hurst estimat prin metoda periodogramei si prin metoda varianta-agregare. In plus este afisat si timpul de calcul. Seriile pe care se face estimarea sunt obtinute prin captura de la o interfata de retea si sunt de doua feluri:

- numarul de pachete ce au trecut prin interfata in cate o perioada de esantionare
- numarul de octeti ce au trecut prin interfata in cate o perioada de esantionare

In continuare se prezinta modul de utilizare a programului, problemele aparute la implementare si solutiile adoptate, structura rezultata a programului si bibliotecile folosite.

5.1 Ghidul utilizatorului

Interfata grafica consta dintr-o singura fereastra de dialog ce are patru zone importante:

- comenzi de pornire / oprire estimare
- configurari globale

- configurari specifice unei metode de estimare
- rezultate ale estimarii

5.1.1 Pornirea şi oprirea estimării

RTH are doua moduri de lucru: configurare si estimare. La pornire el este in starea de configurare. Atata timp cat este in starea de estimare sunt afisate rezultate dar nu se pot modifica configurarile.

Figura 5.1: Butoanele de pornire si oprire a estimarii

Trecerea dintr-o stare in cealalta se face cu ajutorul butoanelor *Start* si *Stop*, dupa cum se observa si in figura 5.1. Atunci cand se da comanda de pornire se face validarea configurarilor si utilizatorul este informat daca ceva este in neregula.

5.1.2 Configurări globale

Zona configurarilor globale este ilustrata in figura 5.2.

Prima configurare importanta este alegerea interfetei de retea pe care se face captura. In acest exemplu ea este o placa de retea "AMD PCNET".

Perioada de esantionare trebuie specificata in *ms*. La alegerea acesteia trebuie sa tineti cont de viteza interfetei si de calitatea ei, care influenteaza precizia masurarilor temporale.

Parametrul Hurst este recalculat dupa fiecare achizitionare a unui grup de esantioane. Marimea acestui grup este configurabila si in acest exemplu este 100. Datele din figura 5.2 arata ca actualizarea parametrului Hurst se va face la fiecare $100 \cdot 10$ms, adica o data pe secunda. Totusi trebuie spus

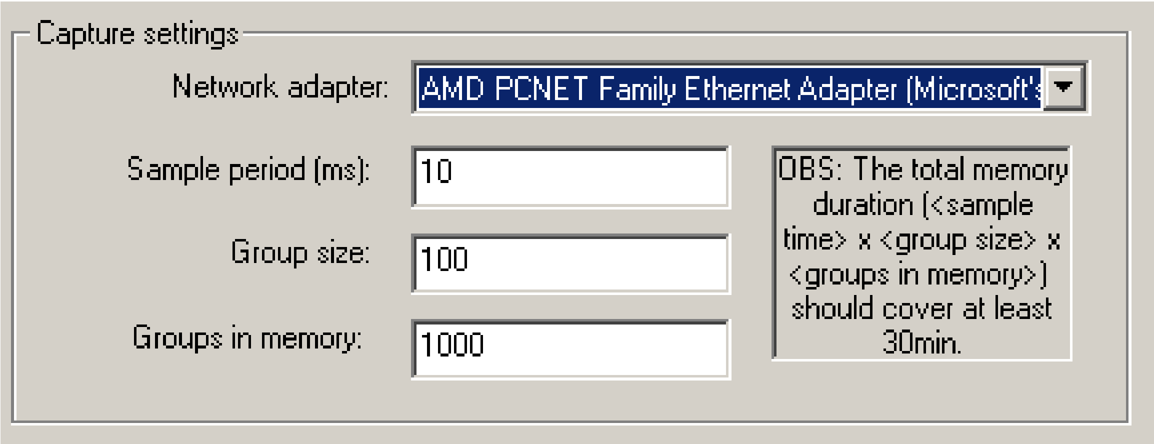

Figura 5.2: Configurari globale

ca desi recalcularea efectiva se face la un interval care depinde de perioda de esantionare si de marimea grupului, totusi actualizarea afisarii se face la un interval fix de 0.5s.

Toate metodele de estimare a parametrului Hurst trebuie sa tina cont de un numar mare de esantioane pentru a da rezultate inteligibile (datorita variantei estimatorului). De aceea la calcularea parametrului Hurst se tine cont de mai multe grupuri. Aceasta este cea de-a patra configurare.

5.1.3 Configurări specifice metodei de estimare

Varianţă-agregare

Figura 5.3 ilustreaza configurarile necesare pentru metoda de estimare varianta-agregare. Acestea reprezinta limita minima si limita maxima de agregare care vor fi luate in considerare atunci cand se face regresia liniara.

Pentru detalii vezi sectiunile 4.2 si 6.1.1.

Periodograma

Ca si la metoda varianta-agregare se specifica limitele folosite pentru regresie. In general frecventa minima trebuie sa fie cat mai apropiata de zero.

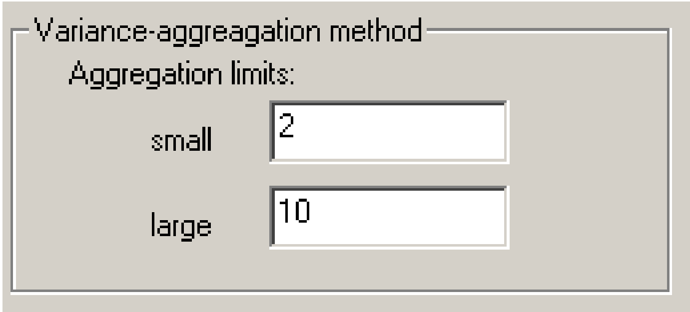

Figura 5.3: Configurari specifice metodei de estimare varianta-agregare

Frecventa maxima poate fi micsorata daca traficul are importante caracteristice de dependenta pe perioade scurte care influenteaza spectrul mai ales la frecvente mari.

Un ordin al nucleului Dirichlet egal cu 0 inseamna ca se calculeaza spectrul fara nici o fereastra. Daca ordinul nucleului Dirichlet creste atunci se acorda o pondere mai mica esantioanelor de la marginea memoriei[1] (vezi ecuatia 4.11).

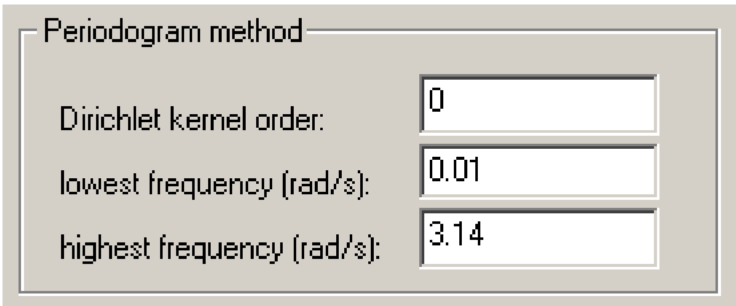

Figura 5.4: Configurari specifice metodei de estimare cu periodograma

5.1.4 Rezultatele estimarii

RTH afiseaza valoarea estimata a parametrului Hurst. Teoretic aceasta trebuie sa fie intre 0 si 1. O valoare egala cu 0.5 indica faptul ca eantioanele

[1]adica cele mai vechi si cele mai noi

nu sunt corelate. O valoare mai *mica* de 0.5 arata ca intre esantioane exista o dependenta pe periade scurte (mai exact, functia de autocorelatie este absolut sumabila). O valoare mai *mare* de 0.5 indica existenta unor dependente pe perioade lungi (mai exact, functia de autocorelatie nu este absolut sumabila).

Se observa in figura 5.5 ca este prezentat si timpul de calcul pentru estimarea. Unitatea de masura este secunda.

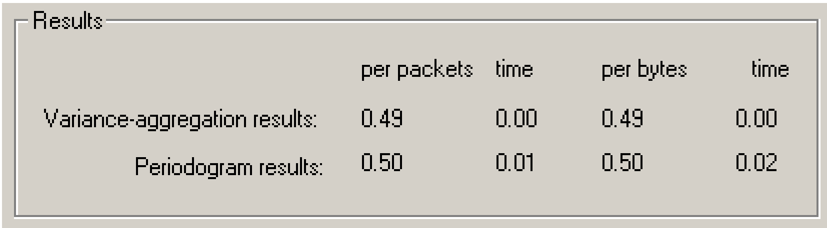

Figura 5.5: Rezultatele RTH

5.1.5 Fişiere de înregistrare (log)

RTH isi inregistreaza evolutia in patru fisiere:

- *errors.log* - Erori aparute in timpul rularii
- *packets.log* - Esantioanele cu numarul de pachete
- *bytes.log* - Esantioanele cu numarul de octeti
- *hurst.log* - Rezultatele estimarilor

Fisierul *errors.log* poate fi folosit pentru a observa daca s-au pierdut esantioane. Acest lucru se intampla cand se acumuleaza un grup de esantioane dar inca nu s-a terminat estimarea pentru grupul anterior.

Fisierele *packets.log* si *bytes.log* pot fi utilizate pentru a face estimari offline. Acestea pot fi mai exacte si pot verifica rezultatele din *hurst.log*.

Fisierul *hust.log* a fost utilizat pentru a obtine graficele din sectiunea 6.2.

5.2 Probleme de implementare

Problemele de implementare cele mai interesante sunt

- Ce insemna o estimare in timp real?
- Cum se poate realiza simultan captura si estimarea? Este vreo legatura intre timpul de calcul si captura?
- Cum putem avea grija in acelasi timp de interactiunea cu utilizatorul?
- Cum pot fi aplicate metodele clasice de estimare pentru o captura in timp real?

In continuarea acestei sectiuni sunt prezentate raspunsurile adoptate in scrierea RTH.

In general estimarea unui parametru al unui proces stocastic presupune captura unei realizari particulare a acestuia si apoi efectuarea unor calcule pentru gasirea parametrului dorit. In general o realizare particulara este infinita in timp. De aceea in practica sunt captate realizari particulare suficient de lungi. Cu cat inregistrarea este mai scurta cu atat estimarea este mai proasta.

In figura 5.6 sunt date doua modalitati de grupare a esantioanelor in date ce vor fi date la intrarea estimatorilor. Fiecare estimator va considera un astfel de grup ca fiind o realizare particulara. Se observa ca in primul caz rezultatele se obtin cu o frecventa mai scazuta, iar grupurile au dimensiuni mai mici asa incat erorile vor fi mai mari. In cel de-al doilea caz actualizarile estimarilor s-ar putea face mai des daca s-ar gasi o solutie de manipulare a cantitatii mai mari de date de intrare. In orice caz estimarea va fi mai buna. Aceasta din urma este metoda aleasa.

Din cele de mai sus rezulta ca este necesar sa se desfasoare in paralel doua activitati: captura esantioanelor si estimarea. De aceea sunt necesare doua fire de executie. Este evident ca timpul de calcul a unei estimari trebuie sa fie mai mic decat timpul dintre doua comenzi de pornire a actualizarii.

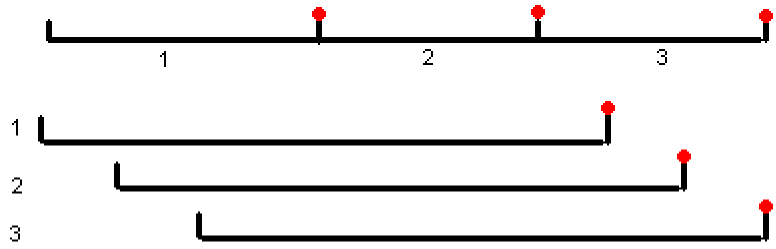

Figura 5.6: Doua posibilitati de grupare a esantioanelor in “realizari particulare”

Comanda de pornire a actualizarii este data in RTH dupa fiecare acumulare de M esantioane. Estimarea se face pe $N \cdot M$ esantioane si trebuie sa dureze cel mult $M \cdot T_e$, unde T_e este perioada de esantionare.

Nu trebuie uitat ca pe durata capturii sau pe durata estimarii interfata grafica nu trebuie sa se blocheze: trebuie sa afiseze continuu rezultatele si trebuie sa astepte ascultatoare comanda de oprire. Asadar sunt necesare de fapt trei fire de executie.

Avand in vedere ca timpul de rulare al estimatorului este limitat de $N \cdot T_e$, care *nu depinde de* M, ar fi de dorit ca nici complexitatea algoritmului de estimare sa nu depinda de M, desi intrarea are dimensiune $M \cdot N$. Acest lucru se poate obtine prin modificarea algoritmilor de estimare plecand de la observatia ca in cazul acesta intrarile a doua rulari succesive difera doar prin doua grupuri de M esantioane (vezi figura 5.6). Detaliile solutiilor alese se gasesc in sectiunea 6.1.

5.3 Structura programului

5.3.1 Fire

RTH are trei fire de executie:

- interfata cu utilizatorul

- `Sampler`

- `Estimator`

In continuare sunt prezentate pe scurt functiile acestora.

Interfata cu utilizatorul permite alegerea a diversi parametrii de catre utilizator (pentru detalii vezi 5.1). Tot interfata cu utilizatorul este cea care interogheaza periodic obiectul ce stocheaza rezultatele si le aafiseaza.

Firul `Sampler` se ocupa cu captura esantianelor. La pornire citeste configurarile stabilite de utilizator si pana este oprit captureaza esantioane si le stocheaza intr-un obiect din care vor fi citite de `Estimator`.

Firul `Estimator` citeste esantioanele de fiecare data cand se strang in numar de M si actualizeaza valoarea estimata a parametrului Hurst.

5.3.2 Obiecte de comunicare

Pentru comunicarea sigura intre fire exista obiectele[2]:

- `Settings`

- `Results`

- `RawData`

Fiecare dintre aceste obiecte pune la dispozitie metode de acces la date care sunt sigure din punctul de vedere al executiei concurente.

Obiectul `Settings` contine toate informatiile de configurare alese de utilizator prin interfata grafica. Obiectul `Results` contine toate rezultatele prezentate de interfata grafica utilizatorului. Obiectul `RawData` este utilizat pentru cominicarea intre `Sampler` si `Estimator` si contine esantioanele propriu-zise. Implementarea lui `RawData` este cu o memorie tampon dubla cu comutare. Functionarea sa conceptuala este ilustrata in figura 5.7.

[2]sunt de fapt clase Singleton

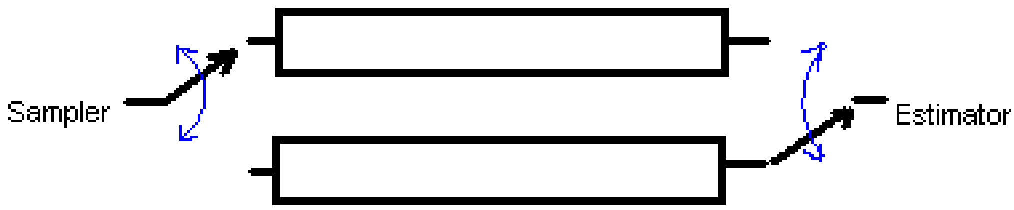

Figura 5.7: Functionarea obiectului RawData

5.3.3 Înregistrarea activităţii

Inregistrarea activitatii se face prin intermediul obiectului `Logger`. Acesta are grija sa puna informatia in fisierul potrivit si sa adauge etichete temporale ce pot fi utilizate pentru o analiza ulterioara.

5.4 Biblioteci folosite

In continuare sunt prezentate cateva dintre bibliotecile utilizate la dezvoltarea programului. Acestea faciliteaza captura cadrelor, calculul unor functii matematice si executia multifilara. Dependenta de sistemul de operare este data de utilizarea bibliotecii MFC, care poate fi inlocuita de exemplu cu wxWindows.

5.4.1 PCAP

Biblioteca PCAP ofera acces direct la subnivelul MAC permitand astfel realizarea unor functii mult mai complexe decat socketurile. Functia principala a acestei librarii este aceea de captura de cadre, filtrate dupa diverse criterii. Dar ea permite si emiterea de cadre si obtinerea de statistici. Statisticile oferite sunt exact cele necesare pentru RTH: numarul de pachete si numarul de octeti din ultima perioada de esantionare.

Utilizarea acesteia presupune trei etape:

- initializarea
- captura efectiva
- eliberarea resurselor

Initializarea se face de catre `Sampler` fiecare data cand se da comanda de pornire. Pentru initializare intai se obtine numele adaptorului de retea[3]:

```
char errbuf[PCAP_ERRBUF_SIZE];
pcap_if_t* adapters;
pcap_findalldevs(&adapters, errbuf);
char* first_adapter_name  = adapters->name;
char* second_adapter_name = adapters->next->name;
```

Apoi, avand numele adaptorului se poate initializa captura si seta modul de lucru statistic:

```
pcap_t* capturer;
int     sample_time  = 10; // in miliseconds
capturer = pcap_open_live(adapter_name, 0xffff, 1, sample_time, errbuf);
pcap_setmode(capturer, MODE_STAT);
```

Captura efectiva se face cu comanda:

```
pcap_loop(capturer, 1, SampleDispatcher, NULL);
```

Aici `SampleDispathcer` este o functie ce este apelata dupa ce s-a receptionat un esantion.

Eliberarea resurselor se face astfel:

[3]codul de aici are doar rolul de a ilustra utilizarea bibliotecilor

```
pcap_close(capturer);
pcap_freealldevs(adapters);
```

Un avantaj al acestei biblioteci este ca e disponibila atat sub UNIX cat si sub Windows si utilizarea ei este libera.

5.4.2 MATLAB

Pentru estimarile implementate este nevoie de unele prelucrari matematice ce sunt greu de scris intr-un limbaj general cum este C/C++:

- calcularea periodogramei
- regresie liniara
- regresie neliniare

In MATLAB toate acestea se fac prin apelul unei singure functii: `periodogram`, `polyfit`, respectiv `nlinfit`. Si alte prelucrari de matrici se fac mult mai usor in MATLAB. De aceea prelucrarile intensiv matematice se fac in functii MATLAB apelate din C++. Codul functiilor MATLAB se gaseste in anexa A.

In principiu tot ce trebuie facut este sa se compileze in cod obiect functiile MATLAB si sa se lege impreuna cu niste biblioteci dinamice ale MATLAB la programul ce le utilizeaza. In practica se observa diverse incompatibilitati cu bibliotecile standard, necesitatea unor setari obscure, etc. care fac ca apelul functiilor MATLAB sa nu fie tocmai trivial.

In plus o asemenea solutie va introduce o intarziere datorata legarii dinamice cu bibliotecile MATLABului ce au dimensiuni impresionante. Si dimensiunea programului creste de substantial din acest motiv. Totusi, usurinta cu care se realizeaza din program prelucrari matematice datorata bazei mare de functii puse la dispozitie face sa merite efortul integrarii.

Bibliotecile MATLAB sunt disponibile atat pentru sistemul de operare UNIX cat si pentru Windows.

5.4.3 MFC

Biblioteca MFC[4] accelereaza mult dezvoltarea de aplicatii C++ pentru Windows in MSVC6[5]. Ea a fost utilizata atat pentru interfata grafica cat si pentru programarea multifilara. In continuare este prezentata numai utilizarea pentru programarea multifilara.

Sunt doua probleme importante: firele de executie si comunicarea dintre ele. Firele de executie trebuie create, pornite si oprite. Comunicarea trebuie sa foloseasca niste obiecte de sincronizare puse la dispozitie de catre sistemul de operare[6] (si chiar de procesor[7]).

MFC usureaza crearea firelor de executie prin punerea la dispozitie a clasei `CWinThread`. Programatorul trebuie sa mosteneasca aceasta clasa si sa rescrie functia membra `Run`:

```
class MyThread : public CWinThread
{
public:
   virtual int Run();
};
```

La apelul functiei `CreateThread` incepe executia unui fir:

```
MyThread thread;
thread.CreateThread();
```

Executia firului se termina odata cu terminarea executiei functiei `Run`. Atentie, nu trebuie confundat obiectul `thread` cu firul de executie (din punctul de vedere al sistemului de operare). Asadar nici durata de viata a obiectului `thread` nu este egala cu timpul de rulare al firului; ea este intotdeauna mai mare.

---

[4] Microsoft Foundation Classes

[5] Microsoft Visual C/C++, versiunea 6

[6] fapt demonstrat de Dijkstra

[7] vezi instructiunea "test" pe procesoarele x86

Cealalta problema este comunicarea firelor de executie. Acest lucru se realizeaza cu ajutorul obiectelor ce au asociat un `mutex`[8] care este “incuiat” pe perioada cat este accesat de un anumit fir. O clasa minimala care realizeaza acest lucru este:

```
class ThreadSafeClass
{
public:
   ThreadSafeClass() {mutex = new CMutex();}
   ~ThreadSafeClass() {delete mutex;}

   void SetMyData(MyDataType data)
   {
      CSingleLock lock(mutex);
      lock.Lock();
      myData = data;
   }

   MyDataType GetMyData()
   {
      CSingleLock lock(mutex);
      lock.Lock();
      return myData;
   }

private:
   CMutex* mutex;
   MyDataType myData;
};
```

Fiind un produs Microsoft biblioteca MFC nu este disponibila decat pentru sistemul de operare Windows. In plus este foarte strans legata de mediul

[8] MUTual EXclusion

de dezvoltare MSVC6 astfel incat utilizarea in alte medii de dezvoltare este greoaie[9].

[9]din cate stiu doar Borland C++ mai suporta MFC

Capitolul 6

Rezultate

Principalele rezultate obtinute sunt:

1. Modificarea algoritmilor de estimare pentru reducerea complexitatii in cazul in care datele de intrare ale unor apeluri succesive se suprapun (vezi figura 5.6).

2. Realizarea programului RTH ce poate fi utilizat in studiul traficului din retelele de calculatoare.

3. Realizarea de inregistrari de trafic, impreuna cu estimarile pentru parametrul Hurst realizate de RTH.

6.1 Modificari aduse metodelor de estimare

In RTH actualizarea estimarii parametrului Hurst se face la fiecare M esantioane. Pentru ca estimarea sa fie mai exacta se tine cont insa de ultimele N grupe de cate M esantioane. Ca urmare, desi intrarea are teoretic dimensiunea $M \cdot N$, algoritmul de estimare ar trebui sa aiba o complexitate ce nu depinde de N. Acest lucru se poate realiza datorita legaturii dintre "intrarile" a doua rulari succesive.

Pe scurt, algoritmii de estimare vor avea o memorie de $M \cdot N$ esantioane. La apelare primesc numai ultimul grup de M esantioane (astfel se micsoreaza dimensiunea intrarii). Pentru a putea avea o complexitate ce nu depinde de N acestia nu se pot uita decat la o portiune mica din propria memorie: practic doar la cel mai vechi grup de M esantioane. Pentru a se putea realiza acest lucru fiecare algoritm are nevoie sa memoreze si cateva rezultate partiale.

In continuare se prezinta modificarile aduse metodei periodogramei si metodei varianta-agregare, impreuna cu o mica analiza a acestora.

6.1.1 Modificări ale analizei varianţă-timp

Aceasta metoda a fost descrisa pe scurt in sectiunea 4.2. Rezultatul se obtine prin potrivirea graficului varianta-agregare pe o curba de tipul $y = cx^{-2(1-H)}$. Aceasta potrivire are o complexitate ce depinde de numarul de grade de agregare luate in considerare si deci nu depinde de numarul de esantioane.

Operatia care depinde de numarul de esantioane este obtinerea curbei. In principiu pentru calculul unui punct al curbei este nevoie agregarea de grad m a unui vector de dimensiune MN si estimarea variantei vectorului agregat. Aceste doua operatii au complexitate $O(MN)$:

$$P \equiv \lfloor NM/m \rfloor \tag{6.1}$$

$$\mu = \frac{1}{MN} \sum_{n=0}^{MN-1} X_n \tag{6.2}$$

$$X_k^{(m)} = \sum_{j=km}^{(k+1)m-1} X_j, 0 \leq k \leq P \tag{6.3}$$

$$\sigma^2(m) = \frac{1}{P-1} \sum_{n=0}^{P-1} (X_n^{(m)} - \mu)^2 \tag{6.4}$$

Insa ne putem descurca mai bine daca tinem minte vechea dispersie, precum si grupul de esantioane cel mai vechi. Voi ilustra ideea doar pentru gradul de agregare $m = 1$. Generalizarea este destul de simpla dar ar complica notatia si ar incurca intelegerea.

Presupunem asadar ca avem $N + 1$ grupuri de M esantioane pe care le indexam pentru a ne putea referi la ele: 0, 1, ..., $N - 1$, N. Problema este urmatoarea: stim varianta σ^2_{old} pentru vectorul format din grupurile 0, 1, ..., $N-1$ si vrem sa gasim un algoritm cu o complexitate ce nu depinde de N pentru a gasi varianta σ^2_{new} vectorului format din grupurile 1, 2, ..., N.

Facem urmatoarele notatii:

$$\mu_k = \frac{1}{M} \sum_{j=kM}^{(k+1)M-1} X_j \tag{6.5}$$

$$\mu_{\text{new}} = \frac{1}{MN} \sum_{j=M}^{M(N+1)-1} X_j \tag{6.6}$$

$$\mu_{\text{old}} = \frac{1}{MN} \sum_{j=0}^{MN-1} X_j \tag{6.7}$$

$$S_k(\mu) = \sum_{j=kM}^{(k+1)M-1} (X_j - \mu)^2 \tag{6.8}$$

$$\sigma^2_{\text{old}} = \frac{1}{MN-1} \sum_{j=0}^{MN-1} (X_j - \mu_{\text{old}})^2 \tag{6.9}$$

$$\sigma^2_{\text{new}} = \frac{1}{MN-1} \sum_{j=M}^{M(N+1)-1} (X_j - \mu_{\text{new}})^2 \tag{6.10}$$

$$\alpha = \frac{1}{N}(\mu_N - \mu_0) \tag{6.11}$$

Cateva dintre aceste notatii se pot exprima usor in cuvinte. De exemplu μ_k este media grupului k; μ_{old} si σ^2_{old} sunt media si respectiv varianta vectorului format din grupurile 0, 1, ..., $N-1$; μ_{new} si σ^2_{new} sunt media si respectiv varianta vectorului format din grupurile 1, 2, ..., N.

In anexa C se gaseste demonstratia urmatoarelor relatii:

$$\mu_{\text{new}} = \mu_{\text{old}} + \alpha \tag{6.12}$$

$$\sigma^2_{\text{new}} = \sigma^2_{\text{old}} + \frac{\left[S_N(\mu_{\text{new}}) - S_0(\mu_{\text{old}}) - M\alpha\big((N-1)\alpha + 2\mu_0 - 2\mu_{\text{old}}\big)\right]}{MN-1} \tag{6.13}$$

Se poate vedea ca aceste doua relatii permit actualizarea valorii lui μ si a lui σ in $O(M)$, complexitate ce nu depinde de N.

6.1.2 Modificări ale metodei periodogramei

Calculul densitatii spectrale de putere de catre functia `periodogram` din MATLAB se face calculand o transformata Fourier in P puncte. Daca dimensiunea vectorului de intrare este mai mica decat P atunci se adauga elemente nule. Daca dimensiunea vectorului de intrare este mai mare decat P atunci se face o pliere in domeniul timp. Demonstratia faptului ca o "pliere" in domeniul timp are ca efect o esantionare in frecventa este data in anexa C.

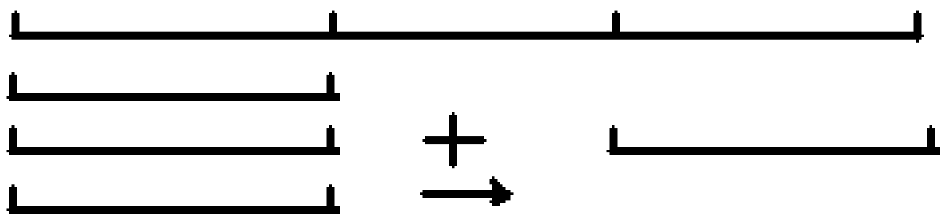

Figura 6.1: Ilustrarea operatiei de "pliere"

Daca P este constant atunci operatia de pliere a unui vector de lunigme MN are complexitate $O(MN)$. Ca urmare aceasta operatie nu se poate lasa pe seama functiei `periodogram`.

Ideea este sa se mentina un vector V de lungime P deja pliat. Cand vine urmatorul grup de M esantioane atunci acestea sunt adaugate la V in pozitia corespunzatoare, iar cele mai vechi M esantioane sunt scazute din V.

In figura 6.2 este ilustrat efectul acestei modificari. Datele analizate reprezinta realizari particulare ale unor variabile independente distribuite uniform in intervalul $[-1, 1]$. Aplicarea metodei periodogramei pe intreg setul de date cu parametrii[1]:

[1] p este ordinul nucleului Dirichlet, iar $f_{\min}$ si $f_{\max}$ sunt limitele datelor luate in con-

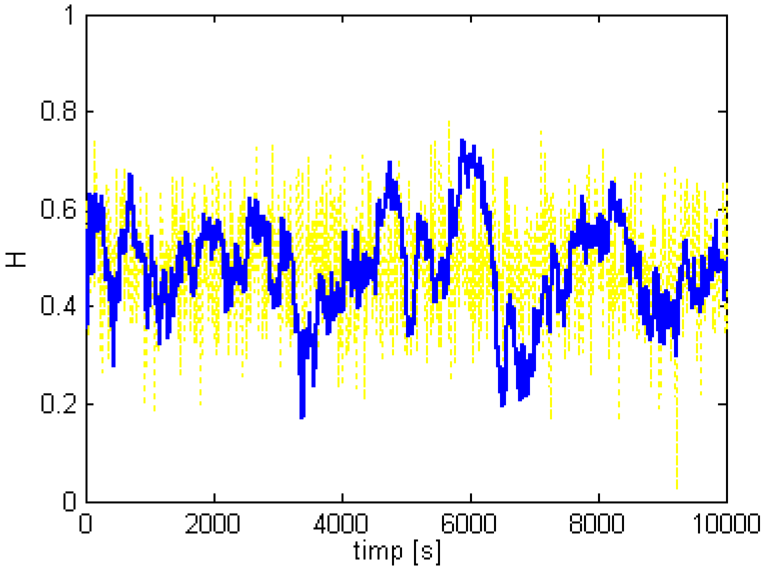

Figura 6.2: Estimarea parametrului Hurst pe grupuri de 1024 esantioane si pe seturi de 100 de grupuri de 1024 de esantioane

$$p = 1 \tag{6.14}$$
$$f_{\min} = 0.01\text{rad/s} \tag{6.15}$$
$$f_{\max} = 3.14\text{rad/s} \tag{6.16}$$
$$\tag{6.17}$$

duce la rezultatul

$$H = 0.5249 \tag{6.18}$$

Teoretic, parametrul Hurst pentru aceasta situatie ar trebui sa fie 0.5, asa incat rezultatul este destul de bun. Rezultatele estimarii sunt insa mai proaste cand se folosesc pentru estimare doar ultimele 1024s (linie albastra)

siderare la regresie

si mult mai proaste cand se folosesc doar ultimele 10s (galben punctat). Metoda prezentata mai sus face ca pentru o actualizare a afisajului la 10s sa se obtina *cu aceeasi complexitate de calcul* rezultate mai bune decat cele reprezentate galben punctat.

6.2 Înregistrări ale parametrului Hurst

Inregistrarile prezentate in sectiunea aceasta si in urmatoarea au fost facute intr-o retea locala cu trafic redus a unei institutii[2]. Voi prezenta in continuare doua seturi de date care au fost inregistrate astfel:

1. in ziua 17 iunie 2003 incepand de la ora 08:46 si pana la ora 10:00

2. in ziua 17 iunie 2003 incepand de la ora 14:36 si pana la ora 17:32

Pentru fiecare dintre aceste seturi parametrii folositi la captura au fost:

- perioada de esantionare 10ms
- perioada de actualizare a parametrului Hurst 1s
- lungimea memoriei 1000s

Parametrii specifici metodei variantei au fost:

- grad minim de agregare 2
- grad maxim de agregare 10

Parametrii specifici metodei periodogramei au fost:

- ordinul nucleului Dirichlet 0 (fara ferestruire)
- frecventa minima 0.01rad/s
- frecventa maxima 3.14rad/s

[2]Serviciul de Telecomunicatii Speciale

Pentru fiecare set de inregistrari se prezinta seria cu numarul de cadre pe perioada de esantionare, seria cu numarul de octeti pe perioada de esantionare si estimarile parametrului hurst pentru cele doua serii. In plus se mai da rezultatul aplicarii metodei periodogramei pe intreg setul de date.

6.2.1 Primul set de inregistrari

In figurile 6.3 si 6.4 se observa ca traficul este neregulat si se deosebeste evident de un zgomot alb. Estimarea parametrului Hurst este o metoda de a cuantifica aceasta afirmatie: cu cat parametrul Hurst se indeparteaza mai mult de valoarea 0.5 cu atat o realizare particulara ca cea din figurile amintite va semana mai putin cu zgomot alb.

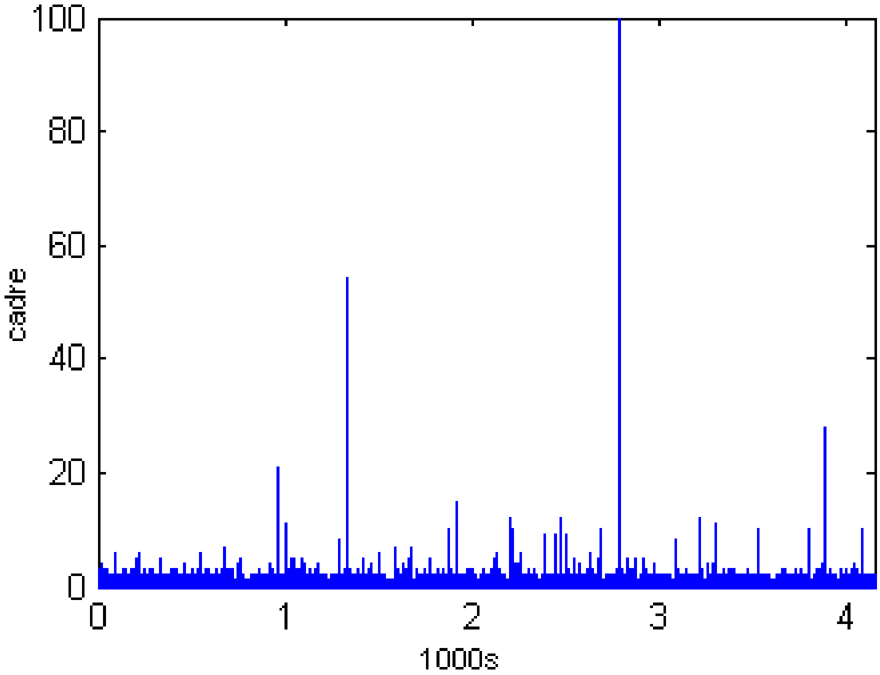

Figura 6.3: SET1: Numarul de pachete pe perioada de esantionare

Parametrul Hurst calculat prin metoda periodogramei pe tot setul de date (off-line) este 0.5492. Valoarea este atat de mica pentru ca traficul este scazut

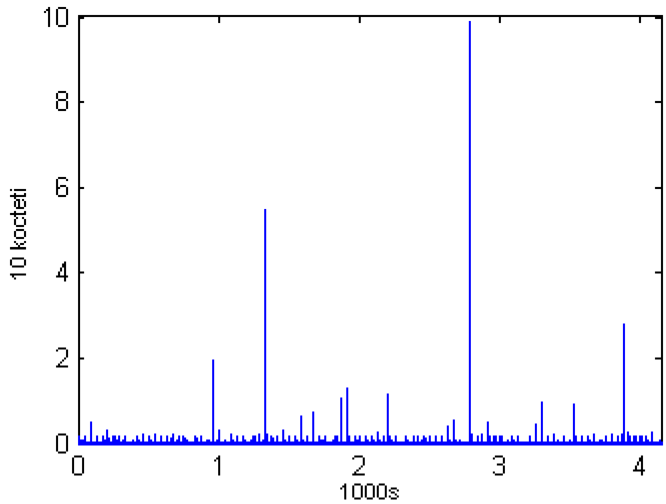

Figura 6.4: SET1: Numarul de octeti pe perioada de esantionare

si provine de la putine surse. Se observa ca metoda variantei da rezultate mult mai stabile decat metoda periodogramei.

6.2.2 Al doilea set de inregistrari

In figurile 6.7 si 6.8 se observa de asemenea ca traficul se deosebeste de zgomotul alb. Pentru a ilustra denumirea de autosimilaritate am inclus figura 6.9 care reprezinta o portiune din figura 6.8 marita cu lupa.

Parametrul Hurst calculat prin metoda periodogramei pe tot setul de date (off-line) este 0.5491. Aceasta valoare este foarte apropiata de cea calculata pentru primul set de date asa incat avem motive puternice sa banuim ca ea caracterizeaza intr-adevar traficul din reteaua locala studiata.

Si de data aceasta se observa ca cei dou estimatori au comportamente cat de cat asemanatoare. De exemplu se observa ca pe perioada marita cu lupa ambele metode dau valori mari pentru parametrul Hurst. Totusi rezultatele date de metoda periodogramei par destul de instabile.

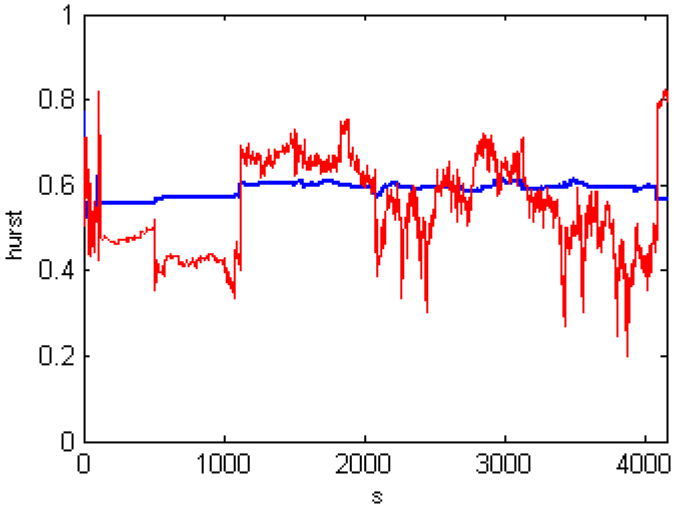

Figura 6.5: SET1: Valoarea estimata in timp real a parametrului Hurst pentru numarul de cadre (albastru = metoda variantei; rosu = metoda periodogramei)

6.3 Comparatie intre timpii de calcul

Pentru aceasta comparatie voi folosi doar datele din al doilea set de masuratori. Datele din primul set sunt asemanatoare si duc la aceleasi concluzii.

Din figurile 6.12 si 6.13 se observa ca metoda variantei este mult mai rapida din punctul de vedere al timpului de calcul. In tabelele 6.1 si 6.2 este dat gradul de utilizare mediu al procesorului[3] pentru a face estimarile.

%	cadre	octeti
metoda variantei	0.05	0.05
metoda periodogramei	1.41	2.32

Tabela 6.1: SET1: Gradul de utilizare al procesorului pentru estimarile parametrului Hurst.

[3] AMD 1300MHz

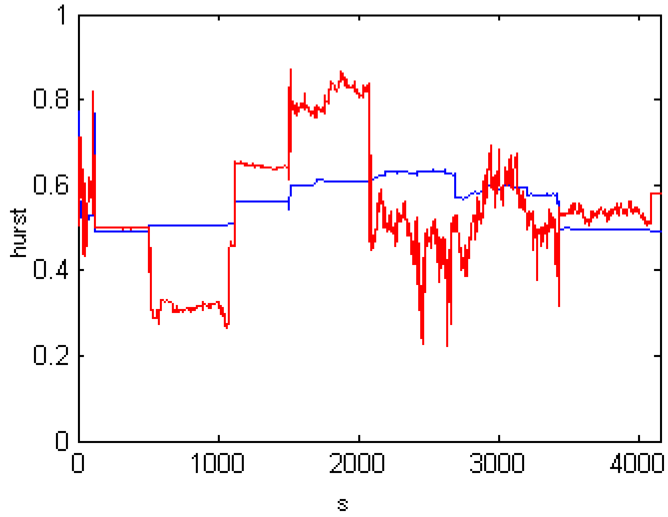

Figura 6.6: SET1: Valoarea estimata in timp real a parametrului Hurst pentru numarul de octeti (albastru = metoda variantei; rosu = metoda periodogramei)

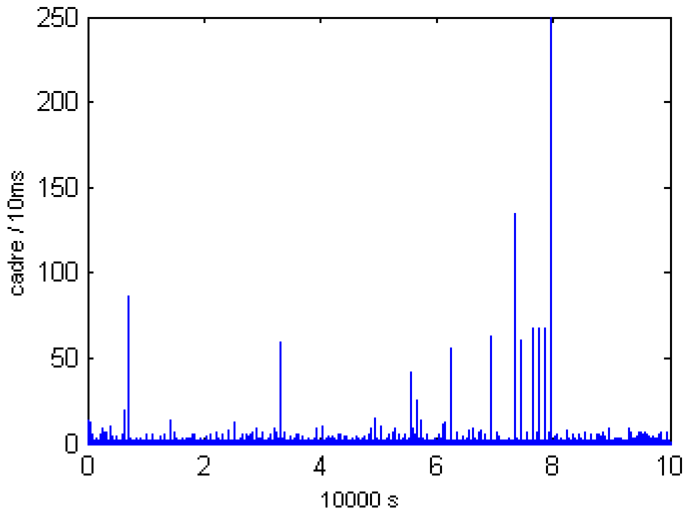

Figura 6.7: SET2: Numarul de pachete pe perioada de esantionare

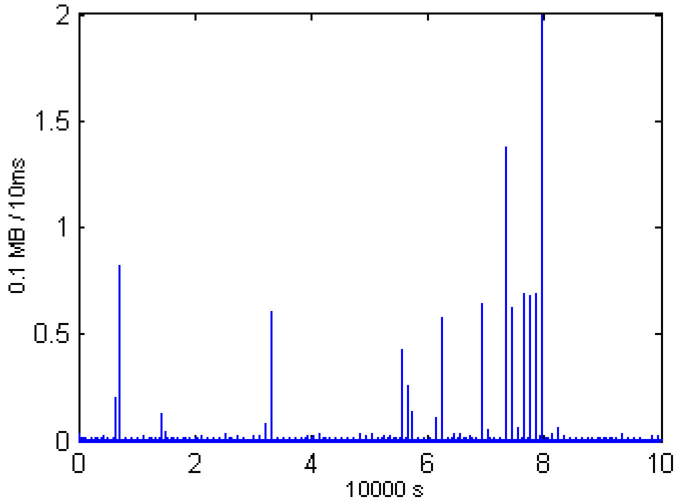

Figura 6.8: SET2: Numarul de octeti pe perioada de esantionare

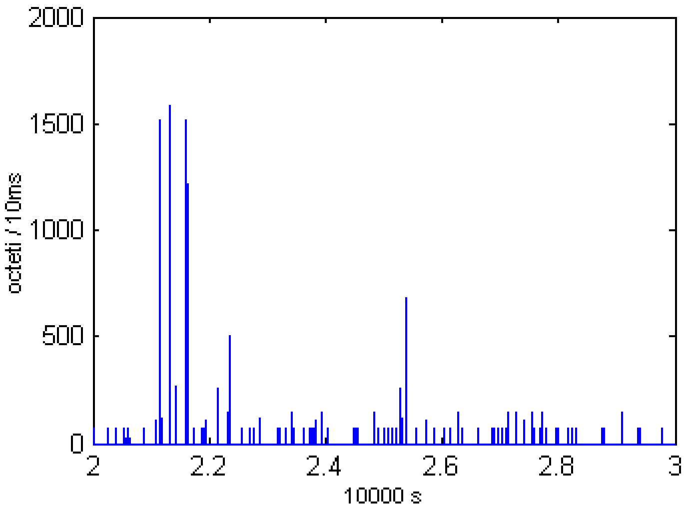

Figura 6.9: SET2: Numarul de octeti pe perioada de esantionare privit cu o lupa

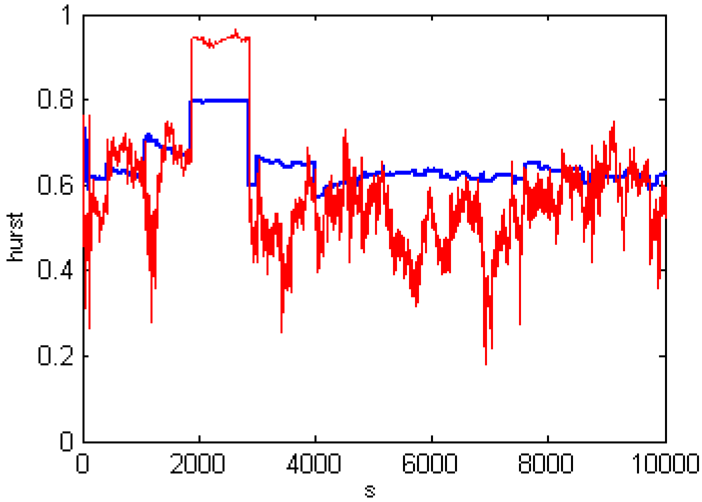

Figura 6.10: SET2: Valoarea estimata in timp real a parametrului Hurst pentru numarul de cadre (albastru = metoda variantei; rosu = metoda periodogramei)

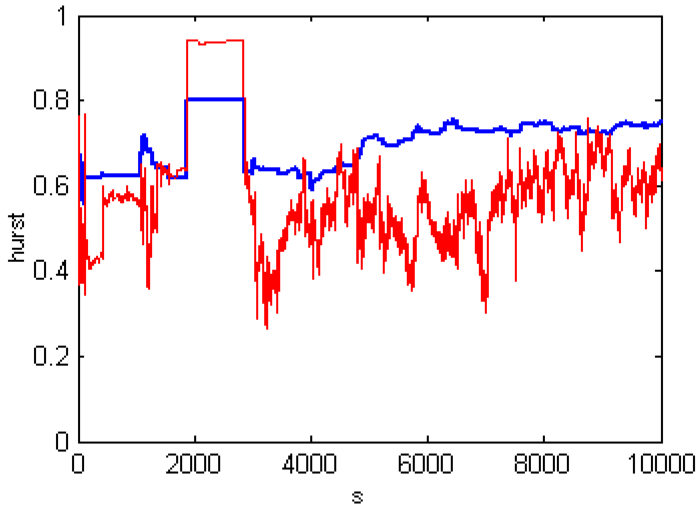

Figura 6.11: SET2: Valoarea estimata in timp real a parametrului Hurst pentru numarul de octeti (albastru = metoda variantei; rosu = metoda periodogramei)

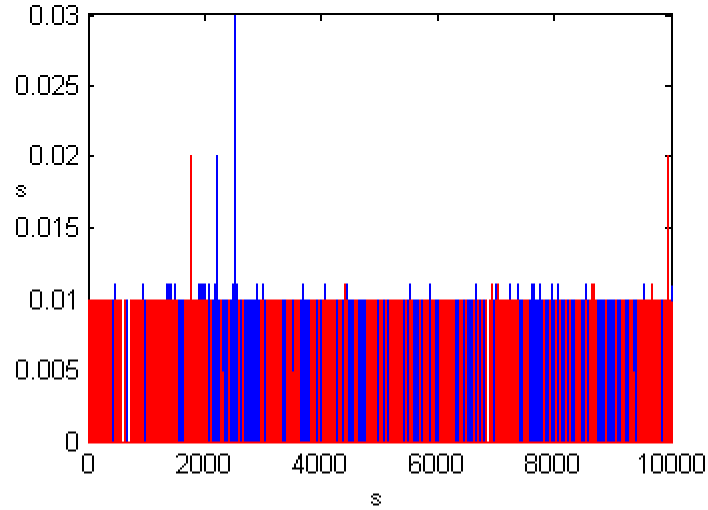

Figura 6.12: SET2: Timpul necesar pentru o actualizare a parametrului Hurst prin metoda variantei

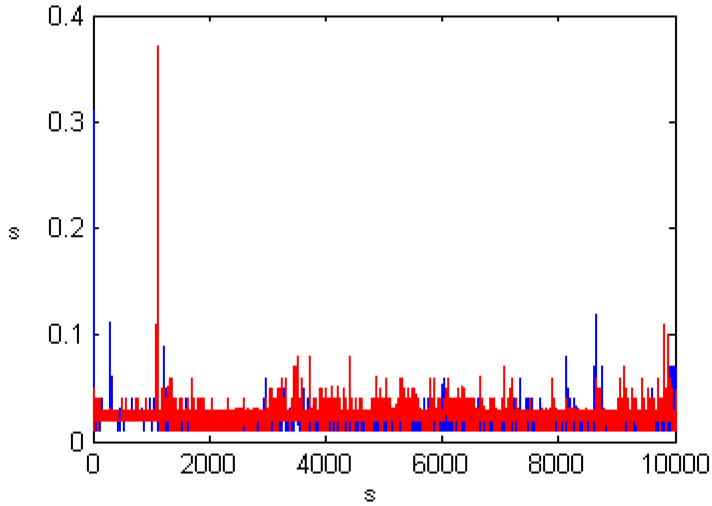

Figura 6.13: SET2: Timpul necesar pentru o actualizare a parametrului Hurst prin metoda periodogramei

%	cadre	octeti
metoda variantei	2.40	0.06
metoda periodogramei	16.50	2.34

Tabela 6.2: SET2: Gradul de utilizare al procesorului pentru estimarile parametrului Hurst.

Capitolul 7

Concluzii

Modelarea traficului cu ajutorul proceselor stocastice autosimilare este promitatoare. Pana acum s-au facut studii mai ales in ceea ce priveste dimensionarea memoriilor tampon ale elementelor de retea. Se spera insa ca noile modele sa aiba aplicabilitate si in zone conexe cum ar fi controlul congestiei. Un prim pas necesar in acest sens este realizarea in timp real si prin software a estimarii parametrului Hurst.

In [22] si in [23] sunt prezentate dispozitive hardware care estimeaza parametrul Hurst al traficului in timp real folosind metoda Veitch-Abry (neprezentata in aceasta lucrare). Totusi majoritatea mecanismelor de control al congestiei sunt implementate la nivelul sistemului de operare si deci este normal sa se incerce estimarea parametrului Hurst tot aici.

Metodele clasice de estimare a parametrului Hurst nu sunt direct aplicabile pentru cazul estimarii in timp real. In aceasta lucrare au fost prezentate modificari a doua metode clasice de estimare care permit aplicarea lor in timp real. In general metoda periodogramei este privita ca fiind mai stabila (in sensul ca estimatorul are varianta mai mica) decat metoda variantei. Totusi varianta modificata a metodei variantei s-a comportat mai bine decat varianat modificata a metodei periodogramei atat din punctul de vedere al stabilitatii cat si din punctul de vedere al timpului de calcul necesar.

Cele doua metode au fost implementate intr-un program care deocamdata ruleaza doar pe sistemul de operare Windows. Acesta foloseste o biblioteca de captura de cadre (PCAP) si o biblioteca pentru calcule matematice (MATLAB). Utilitatea programului a fost ilustrata prin folosirea lui la observarea traficului intr-o retea reala. Rezultatele au permis compararea celor doua metode de estimare. In plus s-a observat ca parametrul H are valori destul de scazute, lucru datorat probabil traficului scazut.

Ceea ce am sperat este ca acest progrm (RTH) sa fie un punct de plecare pentru o unealta de studiu in directia aplicarii noilor modele de trafic in controlul congestiei. De altfel acesta era si scopul dispozitivului prezentat in [22] si [23]. O implementare software are doua avantaje:

- este mai ieftina si deci se poate raspandi mai usor
- este mai aproape de locul unde va fi implementata probabil varianta utila: in sistemul de operare

Directii de dezvoltare ulterioara a RTH sunt:

- implementarea de alte metode de estimare (precedata binentales daca este nevoie de modificarea acestora)
- independenta de platforma (utilizarea wxWindows in locul MFC)
- trecerea mecanismelor de estimare la nivelul sistemului de operare

Anexa A

Codul MATLAB utilizat pentru estimare

A.1 Metoda varianţă-agregare

```
function H = VarianceHurstFit(var, smallAgg, largeAgg)

% function H = VarianceHurstFit(var, smallAgg, largeAgg)
%

k = smallAgg:largeAgg;
log_k = log(k);
log_var = log(var);
p = polyfit(log_k, log_var, 1);
H = 1 + p(1) / 2;
```

A.2 Metoda periodogramei

A.2.1 PeriodogramHurst.m

```
function H = PeriodogramHurst(data, p, lowfreq, highfreq)
```

```
% function H = PeriodogramHurst(data, p, lowfreq, highfreq)
%
% Estimates Hurst parameter by using the Periodogram
% method.
%    data:   the data vector
%       p:   the order of the Dirichlet kernel used for
%            estimating the spectrum
% lowfreq, highfreq: only use frequancies between these limits
%   for fitting; the frequencies are between 0 and PI

[log_phc, w] = PeriodogramHurstCurve(data, p, lowfreq, highfreq);
H = PeriodogramHurstFit(log_phc, w);
```

A.2.2 PeriodogramHurstCurve.m

```
function [log_phc, w] = PeriodogramHurstCurve(data, p, lowfreq, highfreq)

% function [phc, w] = PeriodogramHurstCurve(data, p, lowfreq, highfreq)
%
% Estimates the spectrum of data using a periodogram
% tappered by a Dirichlet kernel of order p. Returns
% only the spectrum for the interval [lowfreq; highfreq].
% Returns the log power spectrum (phc) and the
% the exact frequancies(w) where the spectrum was evaluated.
% These two vectors are the input of PeriodogramHurstFit

% Some constants. Replace with parameters?
fft_points = 1024;
step = 2*pi / fft_points;

% Compute entire periodogram
```

```
data = data - mean(data);
dataLen = length(data);
n = 1:dataLen;
taps = (1-exp(i*2*pi*n/dataLen)).^p;
[DATA, freq] = periodogram(data, taps, fft_points);

% Cut the portion between lowfreq and highfreq
idx_min = floor(lowfreq / step) + 1;
idx_max = ceil(highfreq / step) + 1;
k = idx_min:idx_max;

log_phc(k-idx_min+1) = log(real(DATA(k)));
w(k-idx_min+1)       = freq(k);
```

A.2.3 PeriodogramHurstFit.m

```
function H = PeriodogramHurstFit(log_phc, w)

% function H = PeriodogramHurstFit(log_phc, w)
%
% Receives the logarithm of the power spectrum log_phc
% and the normalized frequencies.
% Tries to (lineary) fit this to the curve:
%
% log_phc = log(C) + (1-2H) log(w)
%
% Then uses the results to refine the H estimate by
% trying to fit to the function
%
% log_phc = log(C) + (1-2H) |1-exp(iw)| + (aw + bw^2)
```

```
p = polyfit(log(w), log_phc, 1);
param0 = [p(1), p(2), 0, 0];
param = nlinfit(w, log_phc, @fitSpectrum, param0);
%y = fitSpectrum(param, w);
%plot(log(w), y, 'r', log(w), log_phc, 'g');
H = (1-param(1))/2;
```

Anexa B

Porţiuni din codul C++ al RTH

B.1 Clasa Sampler

Interfata este:

```
#if !defined(AFX_SAMPLER_H__DC3B356A_DC6F_437E_BBD9_5B1C21BC71EC__INCLUDED_)
#define AFX_SAMPLER_H__DC3B356A_DC6F_437E_BBD9_5B1C21BC71EC__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Sampler.h : header file
//

#include "pcap.h"

/////////////////////////////////////////////////////////////////////////////
// Sampler thread

class Sampler : public CWinThread
```

```
{
public:
    virtual ~Sampler();

protected:
    Sampler();

// Attributes
public:

// Data
private:
    pcap_t* capturer;

// Operations
public:
    static Sampler* GetInstance();
    void Start();
    void Stop();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(Sampler)
    public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    virtual int Run();
    //}}AFX_VIRTUAL

// Implementation
protected:
```

```
    bool sampling;
    CMutex* mutex;

    // Generated message map functions
    //{{AFX_MSG(Sampler)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the p

#endif // !defined(AFX_SAMPLER_H__DC3B356A_DC6F_437E_BBD9_5B1C21BC71EC__INCLUDED_)
```

Implementarea este:

```
// Sampler.cpp : implementation file
//

#include "stdafx.h"
#include "rth.h"
#include "Sampler.h"
#include "RawData.h"
#include "Logger.h"
#include "Settings.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
#endif

extern RawData*  rawData;
extern Logger*   logger;
extern Settings* settings;

//---------------------------------------------------------
// Callback for PCAP

void SampleDispatcher(u_char* param,
    const struct pcap_pkthdr* header, const u_char* pkt_data)
{
    PCapStatData sample;
    sample.sec     = header->ts.tv_sec;
    sample.usec    = header->ts.tv_usec;
    sample.packets = (*((int*)(pkt_data)));
    sample.bytes   = (*((int*)(pkt_data + 8)));
    rawData->WriteSample(sample);
}



/////////////////////////////////////////////////////////////////////////////
// Sampler

Sampler::Sampler()
{
    capturer = NULL;
    sampling = false;
    mutex    = new CMutex();
```

```
}

Sampler::~Sampler()
{
    delete mutex;
}

//----------------------------------------------------------
// Operations

Sampler* Sampler::GetInstance()
{
    static Sampler obj;
    return &obj;
}

void Sampler::Start()
{
    CSingleLock lock(mutex);
    lock.Lock();

    char errbuf[PCAP_ERRBUF_SIZE];
    std::string name = settings->GetAdapterName();
    int sample_time  = settings->GetSampleTime();
    capturer = pcap_open_live(name.c_str(), 0xffff, 1, sample_time, errbuf);
    if (capturer == NULL)
    {
        AfxMessageBox(errbuf);
        sampling = false;
        return;
    }
```

```
    pcap_setmode(capturer, MODE_STAT);

    sampling = true;
}


void Sampler::Stop()
{
    CSingleLock lock(mutex);
    lock.Lock();
    if (capturer != NULL)
        pcap_close(capturer);
    sampling = false;
}


BOOL Sampler::InitInstance()
{
    return TRUE;
}

int Sampler::ExitInstance()
{
    // TODO:  perform any per-thread cleanup here
    return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(Sampler, CWinThread)
    //{{AFX_MSG_MAP(Sampler)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// Sampler message handlers

int Sampler::Run()
{
    CSingleLock lock(mutex);

    lock.Lock();
    while (sampling)
    {
        lock.Unlock();
        pcap_loop(capturer, 1, SampleDispatcher, NULL);
        lock.Lock();
    }

    return 0;
}
```

B.2 Clasa Estimator

Interfata este:

```
#if !defined(AFX_ESTIMATOR_H__B2A14C9B_C9EF_42EB_9125_CF83534CDEEC__INCLUDED_)
#define AFX_ESTIMATOR_H__B2A14C9B_C9EF_42EB_9125_CF83534CDEEC__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Estimator.h : header file
//
```

```
#include "libhurst.hpp"

// Must be power of two and correspond to PeriodogramHurstCurve.m
const int PERIODOGRAM_PSD_SAMPLES = 1024;


/////////////////////////////////////////////////////////////////////////////
// Estimator thread

class Estimator : public CWinThread
{
public:
    virtual ~Estimator();

protected:
    Estimator();

// Attributes
public:

// Operations
public:
    static Estimator* GetInstance();
    void Start();
    void Stop();

// Partial results
private:
    // The last data
    Vector    packets;            // the captured data
```

```
    Vector    bytes;

    // Used in common (all samples)
    StlMatrix pckData;        // all samples
    int       pckDataIdx;    // where to write next
    StlMatrix byteData;        // all samples
    int       byteDataIdx;    // where to write next

    // Periodogram method specific
    Vector    perPckNow;        // the data to be analyzed (1024 samples)
    int       perPckNowIdx;        // where to write next
    Vector    perByteNow;        // the data to be analyzed (1024 samples)
    int       perByteNowIdx;    // where to write next

    // Variance method specific
    Vector    varPckGroupMean;        // The mean of each group
    double    varPckMean;            // The mean for all samples
    Vector    varPckSigma;            // Variance for different aggreagations (all s

    Vector    varByteGroupMean;        // The mean of each group
    double    varByteMean;            // The mean for all samples
    Vector    varByteSigma;            // Variance for different aggreagations (all

// Helpers
private:
    int  memLength;
    int  groupSize;
    int  varLowAgg;        // minimum aggregation
    int  varHighAgg;    // maximum aggregation

    void UpdateHistory();
```

```
    double Average(const Vector& v);
    Vector AggregateVector(const Vector& v, int m);
    double ComputeSigmaSum(const Vector& v, double mean);

// Workers
private:
    double UpdatePeriodogramEstimation(const Vector& newData,
        const Vector& oldData, Vector& now, int& nowIdx);
    double UpdateVarEstimation(const Vector& newData,
        const Vector& oldData, double& mean, double& allMean, Vector& sigma);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(Estimator)
    public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    virtual int Run();
    //}}AFX_VIRTUAL


// Implementation
protected:
    bool     estimating;
    CMutex*  mutex;

    // Generated message map functions
    //{{AFX_MSG(Estimator)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the p

#endif // !defined(AFX_ESTIMATOR_H__B2A14C9B_C9EF_42EB_9125_CF83534CDEEC__INCLUDED_)
```

Implementarea este:

```
#include "stdafx.h"
#include "rth.h"
#include "Estimator.h"
#include "RawData.h"
#include "Results.h"
#include "Settings.h"
#include "Logger.h"
#include "time.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern RawData*  rawData;
extern Results*  results;
extern Settings* settings;
extern Logger*   logger;

//---------------------------------------------------------
```

```
// Helper functions

static double timeDiff(clock_t stop, clock_t start)
{
    return (double(stop) - double(start)) / double(CLOCKS_PER_SEC);
}


//////////////////////////////////////////////////////////////////////////////
// Estimator

Estimator::Estimator()
{
    mutex      = new CMutex();
    estimating = false;
}

Estimator::~Estimator()
{
    delete   mutex;
    //delete[] data;
}

//----------------------------------------------------------
// Operations

Estimator* Estimator::GetInstance()
{
    static Estimator obj;
    return &obj;
}
```

```
void Estimator::Start()
{
    int i;
    CSingleLock lock(mutex);
    lock.Lock();

    // General initialization
    estimating = true;
    groupSize = settings->GetGroupSize();
    memLength = settings->GetMemoryLength();

    pckData.clear();
    pckData.resize(memLength);
    for (i = 0; i < memLength; ++i)
        pckData[i].resize(groupSize, 0.0);
    pckDataIdx = 0;

    byteData.clear();
    byteData.resize(memLength);
    for (i = 0; i < memLength; ++i)
        byteData[i].resize(groupSize, 0.0);
    byteDataIdx = 0;

    // Periodogram method initializations
    perPckNow.clear();
    perPckNow.resize(PERIODOGRAM_PSD_SAMPLES, 0.0);
    perPckNowIdx  = 0;

    perByteNow.clear();
    perByteNow.resize(PERIODOGRAM_PSD_SAMPLES, 0.0);
```

```
    perByteNowIdx  = 0;

    // Variance method initializations
    varLowAgg  = settings->GetVarSmallAggregation();
    varHighAgg = settings->GetVarLargeAggregation();

    varPckGroupMean.clear();
    varPckGroupMean.resize(memLength, 0.0);
    varPckMean = 0.0;
    varPckSigma.clear();
    varPckSigma.resize(varHighAgg - varLowAgg + 1, 0.0);

    varByteGroupMean.clear();
    varByteGroupMean.resize(memLength, 0.0);
    varByteMean = 0.0;
    varByteSigma.clear();
    varByteSigma.resize(varHighAgg - varLowAgg + 1, 0.0);
}


void Estimator::Stop()
{
    CSingleLock lock(mutex);
    lock.Lock();
    estimating = false;
}


BOOL Estimator::InitInstance()
{
    return TRUE;
```

```
}

int Estimator::ExitInstance()
{
    return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(Estimator, CWinThread)
    //{{AFX_MSG_MAP(Estimator)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// Estimator message handlers

int Estimator::Run()
{
    int i;
    StatDataVector data;
    double hurst;

    clock_t start, stop;
    CSingleLock lock(mutex);
    lock.Lock();
    while (estimating)
    {
        lock.Unlock();
        data = rawData->ReadSampleVector();
        packets.clear();
        bytes.clear();
```

```
for (i = 0; i < data.size(); ++i)
{
    packets.push_back(data[i].packets);
    bytes.push_back(data[i].bytes);
}
lock.Lock();

start = clock();
hurst = UpdatePeriodogramEstimation(packets,
    pckData[pckDataIdx],
    perPckNow,
    perPckNowIdx);
stop  = clock();
results->SetPeriodogramHurstPckTime(timeDiff(stop, start));
results->SetPeriodogramHurstPck(hurst);

start = clock();
hurst = UpdatePeriodogramEstimation(bytes,
    byteData[byteDataIdx],
    perByteNow,
    perByteNowIdx);
stop  = clock();
results->SetPeriodogramHurstByteTime(timeDiff(stop, start));
results->SetPeriodogramHurstByte(hurst);

start = clock();
hurst = UpdateVarEstimation(packets,
    pckData[pckDataIdx],
    varPckGroupMean[pckDataIdx],
    varPckMean, varPckSigma);
stop  = clock();
```

```
        results->SetVarHurstPckTime(timeDiff(stop, start));
        results->SetVarHurstPck(hurst);

        start = clock();
        hurst = UpdateVarEstimation(bytes,
            byteData[byteDataIdx],
            varByteGroupMean[byteDataIdx],
            varByteMean,
            varByteSigma);
        stop  = clock();
        results->SetVarHurstByteTime(timeDiff(stop, start));
        results->SetVarHurstByte(hurst);

        UpdateHistory();
        logger->packets(packets);
        logger->bytes(bytes);
        logger->hurst();
    }

    return 0;
}


//---------------------------------------------------------
// Workers

double Estimator::UpdatePeriodogramEstimation(const Vector& newData,
    const Vector& oldData, Vector& now, int& nowIdx)
{
    int i;
    int start_idx;
```

```
// Remove the effect of the oldest remembered group
start_idx = (nowIdx - groupSize * memLength) % PERIODOGRAM_PSD_SAMPLES;
if (start_idx < 0)
    start_idx += PERIODOGRAM_PSD_SAMPLES;
for (i = 0; i < groupSize; ++i)
    now[(i+start_idx) % PERIODOGRAM_PSD_SAMPLES] -= oldData[i];

// Add the effect of the new data
for (i = 0; i < groupSize; ++i)
    now[(i+nowIdx) % PERIODOGRAM_PSD_SAMPLES] += newData[i];

// Update counter
nowIdx += groupSize;
nowIdx %= PERIODOGRAM_PSD_SAMPLES;

// Compute Hurst using MATLAB
try
{
    mwArray dirichlet(settings->GetPeriodogramDirichlet());
    mwArray lowFreq(settings->GetPeriodogramLowFreq());
    mwArray highFreq(settings->GetPeriodogramHighFreq());
    mwArray dataArray(1,
        PERIODOGRAM_PSD_SAMPLES, static_cast<double*>(now.begin()));

    mwArray hurst = PeriodogramHurst(dataArray,
        dirichlet, lowFreq, highFreq);

    return hurst.ExtractScalar(1);
}
catch(...)
```

```
    {
        TRACE("Exception triggered while estimating using MATLAB!!!\n");
        return -1.;
    }
}

double Estimator::UpdateVarEstimation(const Vector& newData,
    const Vector& oldData, double& mean, double& allMean, Vector& sigma)
{
    int i;
    double oldMean = mean;
    double oldAllMean = allMean;
    mean = Average(newData);        // the mean of latest samples
    double alpha = (mean - oldMean) / memLength;  // auxiliary (allMean - oldAllMean
    allMean = oldAllMean + alpha;    // the new mean of all remembered samples
    double beta;            // auxiliary (newSigma - sigma[i])
    Vector aggregated;

    // Update sigma vector one aggregation at a time
    for (i = 0; i <= varHighAgg - varLowAgg; ++i)
    {
        double grp = double(ceil(double(groupSize) / double(i+varLowAgg)));
        beta = 0.0;
        aggregated = AggregateVector(newData, i+varLowAgg);
        beta += ComputeSigmaSum(aggregated, allMean);
        aggregated = AggregateVector(oldData, i+varLowAgg);
        beta -= ComputeSigmaSum(aggregated, oldAllMean);
        beta -= grp * alpha * ((memLength-1)*alpha + 2
            * oldMean - 2 * oldAllMean);
        beta /= double(memLength * grp - 1);
```

```
        sigma[i] += beta;
    }

    // Use MATLAB to fit sigma - aggregation curve
    try
    {
        mwArray varianceArray(1, varHighAgg-varLowAgg+1,
            static_cast<double*>(sigma.begin()));
        mwArray smallAgg(varLowAgg);
        mwArray highAgg(varHighAgg);

        mwArray hurst = VarianceHurstFit(varianceArray, smallAgg, highAgg);
        return hurst.ExtractScalar(1);
    }
    catch (...)
    {
        TRACE("Exception triggered while estimating using MATLAB!!!\n");
        return -1.;
    }
}


//-----------------------------------------------------------
// Helpers

void Estimator::UpdateHistory()
{
    pckData[pckDataIdx++]   = packets;
    byteData[byteDataIdx++] = bytes;

    pckDataIdx  %= memLength;
```

```
    byteDataIdx %= memLength;
}


double Estimator::Average(const Vector& v)
{
    int i;
    double average = 0.0;
    for (i = 0; i < v.size(); ++i)
        average += v[i];
    return average / v.size();
}


Vector Estimator::AggregateVector(const Vector& v, int m)
{
    Vector result;
    double sum;
    int i, j;
    for (i = 0; i < v.size() / m; ++i)
    {
        sum = 0.0;
        for (j = 0; j < m; ++j)
            sum += v[i*m+j];
        result.push_back(sum / m);
    }
    if (v.size() % m != 0)
    {
        sum = 0.0;
        for (i = (v.size() / m) * m; i < v.size(); ++i)
            sum += v[i];
```

```
        result.push_back(sum / (v.size() % m));
    }
    return result;
}


/** Computes \sum_{k=0}^{N-1} (v_k - mean)^2
 */
double Estimator::ComputeSigmaSum(const Vector& v, double mean)
{
    double result = 0;
    int i;
    for (i = 0; i < v.size(); ++i)
        result += (v[i] - mean) * (v[i] - mean);
    return result;
}
```

Anexa C

Demonstraţii

C.1 Metoda varianţă-agregare

Plecam de la urmatorul set de definitii:

$$\mu_k = \frac{1}{M} \sum_{j=kM}^{(k+1)M-1} X_j \tag{C.1}$$

$$\mu_{\text{new}} = \frac{1}{MN} \sum_{j=M}^{M(N+1)-1} X_j \tag{C.2}$$

$$\mu_{\text{old}} = \frac{1}{MN} \sum_{j=0}^{MN-1} X_j \tag{C.3}$$

$$\sigma^2_{\text{old}} = \frac{1}{MN-1} \sum_{j=0}^{MN-1} (X_j - \mu_{\text{old}})^2 \tag{C.4}$$

$$\sigma^2_{\text{new}} = \frac{1}{MN-1} \sum_{j=M}^{M(N+1)-1} (X_j - \mu_{\text{new}})^2 \tag{C.5}$$

$$\alpha = \frac{1}{N}(\mu_N - \mu_0) \tag{C.6}$$

$$S_k(\mu) = \sum_{j=kM}^{(k+1)M-1} (X_j - \mu)^2 \tag{C.7}$$

Demonstrarea relatiei dintre μ_{new} si μ_{old} este simpla si se bazeaza pe gruparea termenilor din suma:

$$MN\mu_{\text{old}} = \sum_{j=0}^{M-1} X_j + \sum_{j=M}^{MN-1} X_j \tag{C.8}$$

$$MN\mu_{\text{old}} = M\mu_0 + \sum_{j=M}^{MN-1} X_j \tag{C.9}$$

$$MN\mu_{\text{new}} = \sum_{j=M}^{MN-1} X_j + \sum_{j=MN}^{M(N+1)-1} X_j \tag{C.10}$$

$$MN\mu_{\text{new}} = MN\mu_{\text{old}} - M\mu_0 + M\mu_N \tag{C.11}$$

De aici rezulta:

$$\mu_{\text{new}} = \mu_{\text{old}} + \alpha \tag{C.12}$$

Demonstratia relatiei dintre σ^2_{old} si σ^2_{new} urmeaza aceeasi idee dar este un pic mai laborioasa. Intai separam sumele:

$$(MN-1)\sigma^2_{\text{new}} = \sum_{j=M}^{MN-1} (X_j - \mu_{\text{new}})^2 + \sum_{j=MN}^{M(N+1)-1} (X_j - \mu_{\text{new}})^2 \tag{C.13}$$

$$(MN-1)\sigma^2_{\text{old}} = \sum_{j=M}^{MN-1} (X_j - \mu_{\text{old}})^2 + \sum_{j=0}^{M-1} (X_j - \mu_{\text{old}})^2 \tag{C.14}$$

Daca facem notatia

$$A = \sum_{j=M}^{MN-1} \left[(X_j - \mu_{\text{new}})^2 - (X_j - \mu_{\text{old}})^2\right] \tag{C.15}$$

atunci scazand ecuatiile C.13 si C.14 se obtine ca:

$$(MN-1)(\sigma^2_{\text{new}} - \sigma^2_{\text{new}}) = A + \left[S_N(\mu_{\text{new}}) - S_0(\mu_{\text{old}})\right] \tag{C.16}$$

Mai ramane doar de calculat A. Intai prelucram doar un termen al sumei (aici am sarit cateva calcule algebrice simple care tin cont de ecuatia C.12):

$$(X_j - \mu_{\text{new}})^2 - (X_j - \mu_{\text{old}})^2 = -2\alpha X_j + \alpha(2\mu_{\text{old}} + \alpha) \tag{C.17}$$

Apoi observam ca:

$$\sum_{j=M}^{MN-1} X_j = MN\mu_{\text{old}} - M\mu_0 \tag{C.18}$$

Si putem gasi in sfarsit ca

$$A = -2\alpha(MN\mu_{\text{old}} - M\mu_0) + \alpha M(N-1)(2\mu_{\text{old}} + \alpha) \tag{C.19}$$

$$A = -2\alpha MN\mu_{\text{old}} + 2\alpha M\mu_0 + 2\alpha MN\mu_{\text{old}} + \alpha^2 MN - 2\alpha M\mu_{\text{old}} - \alpha^2 M \tag{C.20}$$

$$A = 2\alpha M\mu_0 - 2\alpha M\mu_{\text{old}} + \alpha^2 MN - \alpha^2 M \tag{C.21}$$

$$A = 2\alpha M(\mu_0 - \mu_{\text{old}}) + \alpha^2 M(N-1) \tag{C.22}$$

$$A = \alpha M[2(\mu_0 - \mu_{\text{old}}) + \alpha(N-1)] \tag{C.23}$$

Si evident

$$\sigma^2_{\text{new}} = \sigma^2_{\text{old}} + \frac{A + S_N(\mu_{\text{new}}) - S_0(\mu_{\text{old}})}{MN-1} \tag{C.24}$$

C.2 Metoda periodogramei

Demonstrarea validitatii modificarilor aduse metodei periodogramei presupune a arata ca o periodizare in timp corespunde unei esantionari in frecventa.

Fie un setul $\{x_k\}_{k=0,1,\ldots,N-1}$ si transformata Fourier discreta a lui:

$$X_n = \sum_{k=0}^{N-1} x_k w^{kn}, n = 0, 1, \ldots, N-1 \tag{C.25}$$

$$w = \exp\left(\imath \frac{2\pi}{N}\right) \tag{C.26}$$

Setul $\{x_k\}_{k=0,1,\ldots,N-1}$ este periodizat prin pliere si se obtine:

$$y_k = \sum_{p=0}^{m-1} x_{pN/m+k}, k = 0, 1, \ldots, \frac{N}{m} - 1 \tag{C.27}$$

Pentru simplitate in continuare vom considera ca m este submultiplu al lui N. Transformata Fourier a semnalului periodizat este:

$$Y_n = \sum_{k=0}^{\frac{N}{m}-1} y_k w^{mkn}, n = 0, 1, \ldots, \frac{N}{m} - 1 \tag{C.28}$$

$$= \sum_{k=0}^{\frac{N}{m}-1} \sum_{p=0}^{m-1} x_{pN/m+k} w^{mkn} \tag{C.29}$$

Observam acum ca:

$$w^{mnk} = w^{mn(pN/m+k)}, p \in \mathbf{Z} \tag{C.30}$$

Ca urmare

$$Y_n = \sum_{k=0}^{\frac{N}{m}-1} \sum_{p=0}^{m-1} x_{pN/m+k} w^{mn(pN/m+k)} \tag{C.31}$$

$$= \sum_{q=0}^{N-1} x_q w^{mnq} \tag{C.32}$$

$$= X_{mn} \tag{C.33}$$

unde am notat $q = pN/m + k$. Acesta este rezultatul dorit. AM plecat de la o periodizare in timp si am aratat ca intr-adevar aceasta inseamna o esantionare in frecventa.

Bibliografie

[1] *Congestion Avoidance and Control*, V. Jacobson, M. Karels, 1988 [jacobson88congestion.pdf]

[2] *On the Self-Similar Nature of Ethernet Traffic (extended version)*, W. E. Leland, M. S. Taqqu, W. Willinger, D. V. Wilson, 1993 [leland93selfsimilar.pdf]

[3] *Empirically-derived Analytic Models of Wide Area TCP Connections*, V. Paxson, 1994 [paxson94empiricallyderived.pdf]

[4] *Wide-Area Traffic: The Failure of Poisson Modeling*, V. Paxson, S. Floyd, 1994 [paxson94widearea.pdf]

[5] *Analysis*, Modeling and Generation of Selfsimilar VBR Video Traffic, M. Garrett, W. Willinger, 1994 [sigcomm94mwg.pdf]

[6] *Selfsimilarity through High Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level*, W. Willinger, M. Taqqu, R. Sherman, D. V. Wilson, 1995 [willinger95selfsimilarity.pdf]

[7] *Stochastic Modeling of Traffic Processes*, D. Jagerman, B. Melamed, W. Willinger, 1996 [jagerman96stochastic.pdf]

[8] *On the Relationship between File Sizes*, Transport Protocols, and Selfsimilar Network Traffic, K. Park, G. Kim, M. Crovella, 1996 [park96relationship.pdf]

[9] *The Importance of Long Range Dependance of VBR Video Traffic in ATM Traffic Engineering: Myths and Realities*, B. Ryu, A. Elwalid, 1996+?? [ryu.pdf]

[10] *Protocols Can Make Traffic Appear Selfsimilar*, J. Peha, 1997 [self.pdf]

[11] *Network Monitoring System Design*, B. Barr, S. Yoo, T. Cheatham, 1998 [p102-barr.pdf]

[12] *Data Networks as Cascades: Investigating the Multifractal Nature of Internet WAN Traffic*, A. Feldmann, A. C. Gilbert, W. Willinger, 1998 [p42-feldman.pdf]

[13] *On the Relevance of Long Range Dependance in Network Traffic*, M. Grossglauser, J.-C. Bolot, 1998+?? [relevanceLRD.pdf]

[14] *Fast*, Approximate Synthesis of Fractional Gaussian Noise for Generating Selfsimilar Network Traffic, V. Paxson, 1998 [9809030.pdf]

[15] *Selfsimilarity and Heavy Tails: Structural Modeling of Network Traffic*, W. Willinger, V. Paxson, M. Taqqu, 1998 [tails-w16-posted.pdf]

[16] *Network Traffic Modeling using a Multifractal Wavelet Model*, M. S. Crouse, R. H. Riedi, V. J. Ribeiro, R. G. Baraniuk, 1999 [Rie1999Feb5NetworkTra.pdf]

[17] *Selfsimilar network Traffic: An Overview*, K. Park, 1999 [park99selfsimilar.pdf]

[18] *Wavelet-based Queuing Analysis of Gaussian and NonGaussian Long Range dependent Traffic*, V. Ribeiro, 1999 [Rib1999May2Waveletba.pdf]

[19] *Measuring Long Range Dependance under Changing Traffic Conditions*, M. Roughan, D. Veitch, 1999 [roughan99measuring.pdf]

[20] *An Introduction to the Theory of Self-Similar Stochastic processes*, P. Embrechts, M. Maejima, 2000 [an-introduction-to-the.pdf]

[21] *Performace Evaluation of Multiple Time Scale TCP under Selfsimilar Traffic Conditions*, K. Park, T. Tuan, 2000 [park.pdf]

[22] *Real-Time Estimation of the Parameters of Long Range Dependance (extended version)*, M. Roughan, D. Veitch, P. Abry, 2000 [roughan00realtime.pdf]

[23] *Real-Time Measurement of Long Range Dependance in ATM Networks*, M. Roughan, D. Veitch, J. Yates, M. Ahsberg, H. Elgelid, M. Castro, M. Dwyer, P. Abry, 2000+?? [real-time-measurement-of.pdf]

[24] *TCPs Role in Propagation of Slefsimilarity in the internet*, A. Veres, Zs. Kenesi, S. Molnar, G. Vattay, 2000 [tcp-s-role-in.pdf]

[25] *Selfsimilar Processes with Independent Increments Associated with Lévy and Bessel Processes*, M. Jeanblanc, J. Pitman, M. Yor, 2001 [jeanblanc01selfsimilar.pdf]

[26] *Multiscale Queuing Analysis of Long Range Dependant Traffic*, V. J. Ribeiro, R. H. Riedi, M. S. Crouse, R. G. Baraniuk, 2001 [Rib2001Feb1Multiscale.pdf]

[27] *Connection-level Analysis and Modeling of Network Traffic*, S. Sarvotham, R. Riedi, R. Baraniuk, 2001 [p99-sarvotham.pdf]

[28] *A Flow-based Model for Internet BackBone Traffic*, C. Barakat, P. Thiran, G. Iannaccone, C. Diot, P. Owezarski, 2002 [p35-barakat.pdf]

[29] *Does Fractal Scaling at IP Level Depend on TCP Flow Arrival Processes?*, N. Hohn, D. Veitch, P. Abry, 2002 [p63-hohn.pdf]

[30] *Testing the Gaussian Approximation of Aggregate Traffic*, J. Kilpi, I. Norros, 2002 [p49-kilpi.pdf]

[31] *A Novel Approach to the Estimation of the Long Range Dependance Parameter*, M. El Kettani, 2002 [KettaniThesis.pdf]

[32] *Internet Traffic Engineering*, R. Mortier, 2002 [internet_traf.pdf]

NOTA: Publicatiile de mai sus au fost obtinute de pe Internet din trei surse:

1. CiteSeer [*http://citeseer.nj.nec.com/*]
2. ArXiv [*http://arXiv.org/*]
3. Biblioteca digitala ACM [*http://portal.acm.org/*]