ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1999
Seria: INFORMATYKA z. 36 Nr kol. 1414

Piotr PECKA
Instytut Informatyki Teoretycznej i Stosowanej, PAN

OBIEKTOWY SYSTEM DO MODELOWANIA PRZEŁĄCZNIKÓW SIECIOWYCH METODĄ ŁAŃCUCHÓW MARKOWA

Streszczenie. W artykule opisany jest system do modelowania sieci kolejek oparty na metodzie łańcuchów Markowa. System ten to biblioteka klas w języku C++ zawierająca obiekty opisujące pracę przełączników sieciowych (Push-out, Sliding window, Treshold itp.). Program modelujący daną sieć z wykorzystaniem tej biblioteki jest bardzo efektywny czasowo i łatwo konfigurowalny. Użytkownik ma możliwość rozszerzenia biblioteki o nowe elementy sieci. Narzędzie to dobrze nadaje sie do analizy stanów nieustalonych.

AN OBJECT ORIENTED LIBRARY FOR MODELING NETWORK SWITCHES WITH THE USE OF MARKOV CHAINS

Summary. The paper presents a class library which is written in C++ and allows to construct and solve queuing models based on continuous time Markov chains. The C++ application based on this library is time efficient and very flexible. The library contains the objects which describe the work of some high speed networks (e.g. ATM) queueing or control mechanisms (e.g push-out and threshold space-priority queues, sliding window mechanism). The user can enlarge the library by adding new network elements. The software is especially well suited to analyze transient states and to evaluate various control algorithms that prevent traffic congestion in communications networks.

1. Wstęp

Jednym z etapów wykorzystania łańcuchów Markowa do modelowania stanów nieustalonych dla zadanego modelu opisującego pracę węzła sieci komputerowej jest generacja

tzw. macierzy tranzycji. Macierz ta (po transponowaniu) definiuje układ równań różniczkowych liniowych, którego rozwiązanie pozwala znaleźć prawdopodobieństwa stanów opisujących rozkład pakietów w kolejkach danej sieci. Na podstawie tych prawdopodobieństw można policzyć wszystkie pozostałe parametry sieci, takie jak średni czas czekania w kolejce na obsługę, średnia liczba zgłoszeń w kolejce. Metodę łańcuchów Markowa zalicza się do metod analitycznych. Zaletą tej metody jest dokładność uzyskanych wyników. Poważne ograniczenie to rozmiar wygenerownej macierzy, który rośnie dramatycznie w funkcji liczby pakietów krążących w sieci.

Programy dostępne na rynku, takie jak QNAP [1] firmy Simulog czy XMARCA [2], nie dają zadowalających wyników. Maksymalna macierz, jaką dało się wygenerować tymi narzędziami,miała rząd 100 tys., a czas generacji wynosił około 2 godzin na stacji roboczej Ultra Sparc 164 MHz. Programy napisane z wykorzystaniem prezentowanej biblioteki robi to kilkanaście razy szybciej przy znacznie mniejszej zajętości pamięci operacyjnej. Dzięki temu można generować macierze rzędu kilku milionów w rozsądnym czasie.

2. Opis biblioteki klas

Proces modelowania sieci za pomocą prezentowanej metody składa się z następujących etapów:

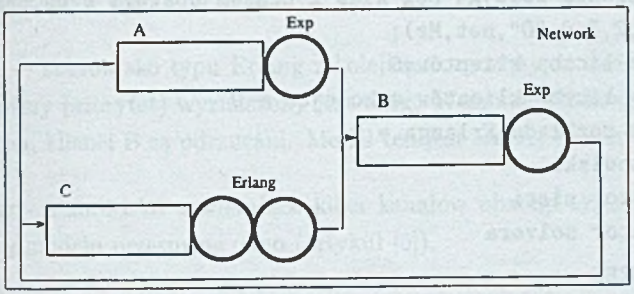
- opis sieci za pomocą obiektów dostępnych w bibliotece,
- generacja macierzy tranzycji,
- rozwiązanie układu równań różniczkowych liniowych (metoda aproksymacji Padego [3]),
- obliczenie parametrów sieci na podstawie prawdopodobieństw brzegowych.

W artykule zostanie opisany sposób opisu sieci za pomocą obiektów dostępnych w bibliotece, opis zasad rozbudowy biblioteki o nowe elementy oraz przykład świadczący o przydatności opisywanego narzędzia. Podstawy teoretyczne zastosowania łańcuchów Markowa do modelowania sieci kolejkowych zostały opisane w artykule [8].

2.1. Zasada budowy sieci

Użytkownik biblioteki opisuje swój model w języku C++ (w funkcji `main`). Program jest kompilowany i łączony z biblioteką zawierającą definicje obiektów sieci, po czym może

być uruchomiany na stacjach roboczych pod systemem Solaris 2.x. Poniżej przedstawiono przykład opisu modelu z rys. 1:



Rys. 1. Prosta sieć złożona trzech stanowisk obsługi
Fig. 1. A simple queueing network with exponent and Erlang service times

```
// DEFINICJA OBIEKTÓW SIECI
#include "mark_solver.h"
// net - obiekt reprezentujący sieć (klasa Network)
Network net("4 proste stanowiska obsługi",1,CLOSED);
// nazwa sieci
// liczba klas klientów
// CLOSED - sieć jest zamknięta
// obiekty reprezentujące metodę wyszukiwania stanów:
// wyszukiwanie oparte na metodzie funkcji odwrotnej [8]
// RevFunSTSearcher St((unsigned char)4,(unsigned char)10);
// wyszukiwanie oparte na metodzie funkcji haszującej,
// słownik tworzony jest w pamięci operacyjnej [4]
// HashSTSearcher St;
// wyszukiwanie oparte na metodzie funkcji haszującej, słownik
// tworzony jest w pamięci dyskowej (najczęściej stosowany)
DbmSTSearcher St;
// wybór metody Markowa (metoda alternatywna - symulacja):
MarkSolver Ms(net,&St,NO);
// identyfikator sieci
// id. wyszukiwacza stanów
// drukowanie macierzy w postaci ascii
// Exp - prosta kolejka bez klas z wykładniczym czasem obsługi
Exp A(6,6,1.0,"J",net,Ms), B(6,0,2.0,"G",net,Ms);
// maksymalna liczba klientów=6
// początkowa liczba klientów w kolejce = 6 (dla stacji A)
// nazwa stanowiska
```

```
// identyfikator sieci
// identyfikator solvera
// Erlang - prosta kolejka bez klas z czasem obsługi typu Erlang
Erlang C(6,0,2,3.0,"O",net,Ms);
// maksymalna liczba klientów=6
// początkowa liczba klientów w kolejce = 0
// liczba faz rozkładu Erlanga = 2
// nazwa stanowiska
// identyfikator sieci
// identyfikator solvera
// OPIS POŁĄCZEŃ
   main()
{
// Połączenia elementów sieci:
// A jest podłączone do B z prawdopodobieństwem 1,  klasa klientów
// jest nie zdefiniowana
A.SetTransition(CL_Undefined,B,1.0);
// B jest podłączone do A z prawdopodobieństwem 0.2,
// do C z prawdopodobieństwem 0.8, klasa klientów
// jest nie zdefinowana
B.SetTransition(CL_Undefined,A,0.2);
B.SetTransition(CL_Undefined,C,0.8);
C.SetTransition(CL_Undefined,B,1.0);
// rozpoczęcie procesu generacji macierzy
Ms.Run();
}
```

W bibliotece istnieją klasy reprezentujące następujące elementy sieci:

- Exp — kolejka rozpoznająca jedną klasę klientów ze stanowiskiem z wykładniczym czasem obsługi.
- Erlang — kolejka rozpoznająca jedną klasę klientów ze stanowiskiem z czasem obsługi typu Erlang.
- ExpSource — źródło generujące klientów (średni czas generacji klientów jest wykładniczy).
- ErlSource - źródło generujące klientów (czas generacji klientów jest typu Erlang).
- Sink - obiekt gubiący klientów.

• ExpSwitch — obiekt typu węzeł, współpracujący z obiektami typu źródło. Zmienia swój stan z on na off i odwrotnie, zgodnie z rozkładem wykładniczym. W stanie on przepuszcza klientów, w stanie off nie.

• ErlTreshHold — stanowisko typu Erlang z kolejką, rozpoznające dwie klasy klientów. Dla klasy B (niższy priorytet) wyznaczony jest próg. W momencie kiedy wartość progu jest przekroczona, klienci B są odrzucani. Model ten jest szerzej opisany w [6].

• MultiErlServer - stanowisko zawierające kilka kanałów obsługi typu Erlang. Obiekt wykorzystano w modelu przesuwne okno (artykuł [5]).

• PushOutErl - stanowisko typu Erlang z kolejką priorytetową rozpoznającą dwie klasy pakietów różnej długości z regulaminem typu push-out (pakiety o wyższym priorytecie mogą wyrzucić z kolejki odpowiednią liczbę pakietów o niższym priorytecie, artykuł [7]).

• DoubleQueueErl - dwie kolejki (dla 2 klas klientów) obsługiwane przez jedno stanowisko obsługi typu Erlang. Stanowisko pobiera naprzemiennie dwóch klientów z pierwszej kolejki i jednego z drugiej.

• ErlTwoClasses - stanowisko typu Erlang z kolejką typu FIFO dla 2 klas klientów.

• Queue3Preempt - stanowisko typu Erlang obsługuje 3 kolejki priorytetowe z wywłaszczaniem (stanowisko obsługuje pierwszą, gdy tylko coś się w niej pojawi, drugą pod warunkiem że pierwsza jest pusta, trzecią, gdy pierwsza i druga są puste. Obsługa klienta z kolejki o niższym priorytecie jest zawieszana, gdy w kolejce o wyższym priorytecie pojawi się nowe zgłoszenie.

• MMPPSource - jest to źródło typu on-off generujące strumień klientów wg rozkładu wykładniczego.

Po skompilowaniu funkcji `main` i połączeniu z biblioteką otrzymujemy plik binarny, który po uruchomieniu wygeneruje macierz tranzycji dla zadanego modelu. Do obliczenia prawdopodobieństw stanów na podstawie macierzy tranzycji służy dodatkowy program stanowiący część systemu.

3. Rozszerzenie biblioteki klas o nowe obiekty

Klasy Exp, Erlang, MultiErlServer reprezentują elementy sieci. Każda z tych klas dziedziczy (dziedziczenie wielobazowe) z dwóch klas:

- MarkItem: poprzez funkcje (funkcje wirtualne) zdefiniowane w tej klasie, solver (obiekt, którego metody zawierają główny kod generacji macierzy tranzycji) wykonuje operacje na obiektach sieci (obiekty klasy Exp, Erlang itd.). Najważniejsze funkcje zdefiniowane w tej klasie to SndTransition() i RcvTransition().

- NetItem: dzięki funkcjom zdefiniowanym w tej klasie obiekty sieci można ze sobą łączyć (funkcja SetTransition()). Niektóre z tych funkcji służą do sprawdzenia poprawności połączeń w sieci.

Opisane dziedziczenie umożliwia łatwe rozszerzenie biblioteki o nowe elementy sieci - po zdefiniowaniu nowej klasy reprezentującej element sieci, dziedziczy się z klasy NetItem oraz MarkItem (przy definiowaniu klasy dla metody np. symulacji będzie to klasa SimuItem).

Programista, chcąc rozszerzyć bibliotekę o klasę reprezentującą nowy element sieci, powinien w części deklaracyjnej tej klasy zadeklarować podwektor stanu (klasa VSItem) odpowiedniego rozmiaru. Następnym krokiem jest zdefiniowanie dwóch funkcji:

- SndTransition() — ta funkcja zawiera odpowiedni kod związany z wysłaniem klienta do innego stanowiska obsługi, do którego obiekt wywołujący ją, jest podłączony.

- RcvTransition() — tę funkcję wywołuje stacja wysyłająca klienta. Funkcja zwraca kopię zmienionego (na skutek tranzycji) podwektora stanu.

Poniżej przedstawiono część definicyjną klasy Exp:

```
// CZĘŚĆ DEFINICYJNA
// definicja konstruktora
Exp::Exp(short int mc,short int nc,float m,char *name,
Network &net, MarkSolver &solv):
mi(m),max_client(mc),nb_client(nc),MarkNetSwitch(solv,name)
// uwaga:  klasa Exp dziedziczy z klasy MarkItem i NetItem
// pośrednio przez klasę MarkNetSwitch !
// w nowszych kompilatorach klasa pośrednia MarkNetSwitch jest zbędna
{
char c[1];
c[0]=nc; // inicjalizacja liczbą klientów
myvsi= new VSItem(1,c); // tworzenie podwektora stanu długości 1
CreateMyNode() ; // tworzenie węzła sieci
net.AddNetItem(*this) ; // dodanie siebie do sieci
mynode=GetMyNode(); // pobranie adresu przydzielonego węzła
vs_flag=TRUE; // obiekty NIE posiadające swojego podwektora
// stanu (np.  Sink, ExpSource) mają tę flagę ustawioną na FALSE
```

```
};

void Exp::SndTransition()
{
// ustawienie węzła na pierwszy element sieci:
mynode->Reset(CL_Undefined);
NetItem* ni; // wskaźnik do elementu sieci
VSToken * vst; // wskaźnik do obiektu VSToken
double p;
if((myvsi->vsb[0])<=0)
// kolejka jest pusta,  nie można wykonać tranzycji
return;
// w tej pętli obiekt będzie wymuszał tranzycje na elementach
// sieci, do których jest podłączony:
do
{
//pobranie kolejnego adresu elementu sieci:
mynode->GetTrans(CL_Undefined, ni, p);
// w ni znajduje się adres obiektu, do którego zostanie wysłany
// klient z prawdopodobieństwem p;
// sprawdzenie, czy wyjście ze stanowiska jest sprzężone z kolejką
// w tym przypadku wykonuje się instrukcję pustą:
if(ni==this)
;
else
// wymuszenie tranzycji na elemencie sieci
//(wysyła się do niego jednego klienta )
if((vst=((MarkItem *)(MarkNetSwitch*)ni)->
RcvTransition(1,CL_Undefined))!=NULL)
{
// obiekt przyjął tranzycję,  w vst znajduje się adres
// obiektu VSToken z wszystkimi potrzebnymi informacjami:
// identyfikator obiektu, adres zmienionego na skutek tranzycji
// podwektora stanu
// przydzielenie obiektu VSToken:
// ( inicjalizacja podwektorem stanu )
VSItem *nextvsi= new VSItem(myvsi);
// zmniejszenie wartości podwektora stanu
```

```
// o jeden, co jest związane z odejściem
// klienta ze stanowiska:
nextvsi->vsb[0]--;
// przydzielenie obiektu VSToken
// z ustawieniem własnego identyfikatora
VSToken *snd=new VSToken(myident);
// przyłączenie obiektu VSToken
// zwróconego przez obiekt wymuszany:
snd->nextvst[0]=vst;
snd->vsi=nextvsi; // przyłączenie własnego podwektora stanu
snd->t_val=p*mi; // wpisanie wartości tranzycji
mysolver->InsertToken(snd); // wysłanie VSTokena do kolejki solvera
}
} while(mynode->Next(CL_Undefined));
// kontynuacja,jeżeli jest dostępny następny
// element sieci
};

// Metoda wywoływana w momencie wymuszenia tranzycji
// przez inny obiekt:
VSToken * Exp::RcvTransition(int nbu,Classes cl )
{
// tworzenie obiektu VSItem zainicjalizowanego
// kopią własnego podwektora stanu:
VSItem *vsi=new VSItem(myvsi);
// sprawdzenie, czy liczba klientów, która przybyła do stanowiska,nie
// przekroczyła długości kolejki:
if((vsi->vsb[0]+nbu)<=max_client)
// jeżeli nie przekroczyła, to zwiększa się wartość
// podwektora stanu o liczbę klientów, która nadeszła:
vsi->vsb[0]+=nbu;
else
{
// jeżeli przekroczyła, to usuwa się przydzielony
// obiekt VSToken i wysyła pusty wskaźnik,  informujący,
// że tranzycja nie może się odbyć
delete vsi;
return(NULL);
```

```
};
// metoda zwraca VSToken ze swoją zmienioną kopią podwektora stanu i
// identyfikatorem swojego obiektu:
VSToken *rcv = new VSToken(myident,0,0,vsi);
return(rcv);
};
```

Struktura funkcji SndTransition i RcvTransition dla pozostałych klas reprezentujących elementy sieci jest podobna. W obiekcie klasy VSToken zapisuje się zmienione na skutek tranzycji podwektory stanów. Obiekt ten jest wysyłany do kolejki solvera, gdzie określany jest numer kolumny i element macierzy tranzycji.

4. Model przełącznika sieciowego

Na rys. 3 pokazano model przełącznika sieciowego dla transmisji multimedialnej opisanego dokładnie w artykule [9]. Jako model kolejkowy składa się on z 8 źródeł on-off generujących 4 klasy klientów o różnych priorytetach (A - najwyższy, D - najniższy). Stanowisko z rozkładem typu Erlang (aproksymacja rozkładu ze stałym czasem obsługi) pobiera klientów z bufora A; z bufora B, C, jeżeli bufor A jest pusty; z bufora D, jeżeli pozostałe są puste. Poniżej pokazano źródło w C++ dla tego modelu:

```
#include "mark_solver.h"
DbmSTSearcher St;
MarkSolver Ms(net,&St,4,OPEN);
Network net("4 Prempt  ",1,YES);
MMPPSource source1a(1,CL_A,0.1,0.21,0.611,2,"jinx1",net,Ms);
MMPPSource source1b(1,CL_B,0.1,0.21,0.511,2,"jinx2",net,Ms);
MMPPSource source1c(1,CL_C,0.1,0.21,0.411,2,"jinx3",net,Ms);
MMPPSource source1d(1,CL_D,0.1,0.21,0.411,2,"jinx4",net,Ms);
Queue4Preempt ns(1,3,2,2,(2.0/2.5),(2.0/1.67),(12.0/1.11),"dixie",net,Ms);
Sink sink("pixie",net,Ms);

main()
{
  source1a.SetTransition(CL_Undefined,ns,1.0);
  source1b.SetTransition(CL_Undefined,ns,1.0);
  source1c.SetTransition(CL_Undefined,ns,1.0);
  source1d.SetTransition(CL_Undefined,ns,1.0);
```

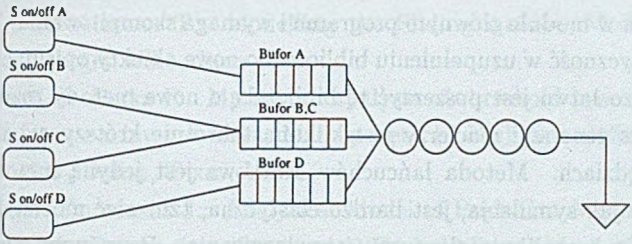
```
ns.SetTransition(CL_Undefined,sink,1.0);
Ms.Run();
}
```

Obiekty sourcela, sourcelb, sourcelc, sourceld (klasa MMPP) to podwójne źródła on-off (4 obiekty reprezentują 8 źródeł on-off: 2 źródła dla klasy A, 2 dla klasy B itd.) Parametry konstruktora to kolejno: liczba wysyłanych klientów, klasa, intensywność źródła, średni czas trwania stanu on, średni czas trwania stanu off, liczba strumieni on-off. Obiekt ns reprezentuje przełącznik sieciowy (klasa Queue3Preempt) składający się z 3 kolejek, z których jedna rozpoznaje 2 klasy klientów: B i C. Parametry konstruktora dla klasy Queue4Preempt: rozmiar pierwszego bufora (dla klasy A), drugiego bufora (dla klasy B i C), rozmiar 3 bufora, liczba faz Erlanga, średnie czasy obsługi dla poszczególnych kolejek. Wektor stanu V dla tego modelu zawiera 11 składowych V[La,Lb,Lc,Ld,Fa,Fbc,Fd,Sa,Sb,Sc,Sd], gdzie La to liczba klientów w buforze pierwszym, Lb i Lc to liczba klientów w buforze drugim, Ld w buforze trzecim. Fa, Fbc i Fd to fazy Erlanga dla klientów poszczególnych klas, Sa określa liczbę aktywnych źródeł dla klasy A, Sb dla klasy B itd. Ze względu na 5 obiektów reprezentujących sieć (obiekt sink nie posiada podwektora stanu) wektor V możemy podzielić na 5 podwektorów: SVa[Sa], SVb[Sb], SVc[Sc], SVd[Sd], Sns[La,Lb,Lc,Ld,Fa,Fbc,Fd]. Liczbę możliwych stanów można określić wzorem kombinatorycznym. Program zlicza nowo powstałe stany w czasie generacji. Szybkie sprawdzenie wektorów stanów w słowniku haszującym (w celu sprawdzenia, czy dany stan nie wystąpił i odczytania jego numeru, który będzie numerem wiersza macierzy tranzycji) przyczyniło się do szybkości algorytmu generacji macierzy tranzycji. Na rozmiar tej macierzy mają wpływ rozmiary buforów, liczba faz Erlanga i liczba źródeł on-off. W tabelce poniżej przedstawiono rozmiar macierzy, liczbę elementów niezerowych macierzy, czas generacji macierzy tranzycji, czas obliczenia prawdopodobieństw stanów metodą aproksymacji Padego dla różnej liczby faz stanowiska o czasie obsługi typu Erlanga.

Liczba faz	Rząd mac.	L. elem. niezer.	Czas generacji [s]	Czas oblicz prawd.[s]
1	11583	102276	42	30
3	101169	871560	236	193
5	337851	2888028	800	1130

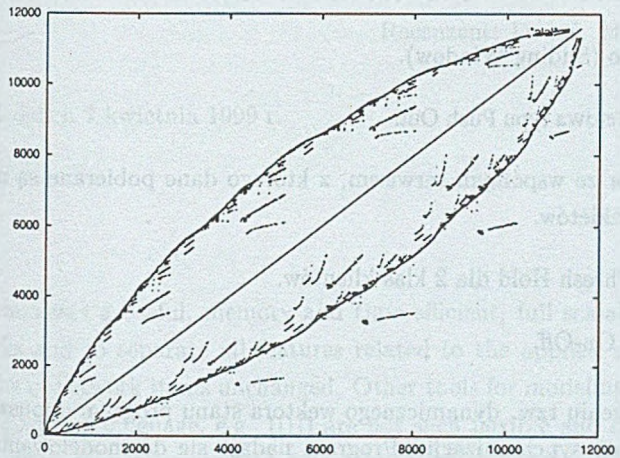
Rys. 2. Czas generacji macierzy i obliczenia wektora własnego
Fig. 2. Benchmark for matrix generation and eigenvector computation

Liczba elementów niezerowych jest kilka razy większa niż rząd macierzy. Świadczy to o tym, że macierze tranzycji należą do tzw. macierzy rzadkich i istnieją specjalne



Rys. 3. Model kolejkowy przełącznika sieciowego
Fig. 3. Model of network's switch

metody iteracyjne dla ich rozwiązania, takie jak: metoda Gaussa-Seidela, metoda Arnoldiego (stany ustalone) oraz aproksymacja Padego dla stanów nieustalonych. Rys. 4 jest reprezentacją graficzną macierzy rzadkiej modelu przełącznika: czarnym kolorem zaznaczono elementy niezerowe.



Rys. 4. Reprezentacja graficzna macierzy rzadkiej
Fig. 4. Spectrum of transition matrix

5. Wnioski

W artykule opisano bibliotekę do rozwiązywania modeli kolejkowych metodą łańcuchów Markowa. Narzędzie to, w przeciwieństwie do innego tego typu systemów, jak XMARCA czy QNAP firmy Simulog, jest biblioteką klas napisaną w języku C++. Opis

sieci zawarty jest w module głównym programu i wymaga skompilowania. Takie podejście daje pełną elastyczność w uzupełnieniu biblioteki o nowe obiekty opisujące prace elementów sieci. Bardzo łatwo jest poszerzyć tę bibliotekę o nowe metody rozwiązywania, np. symulację. Czas generacji macierzy jest kilkunastokrotnie krótszy niż w innych wspomnianych narzędziach. Metoda łańcuchów Markowa jest jedyną metodą analityczną, która, podobnie jak symulacja, jest bardzo elastyczna, tzn. sieć można dowolnie rekonfigurować, co nie komplikuje algorytmu rozwiązywania. Poważnym ograniczeniem tej metody jest złożoność obliczeniowa dla dużych modeli. Narzędzie to zastąpi poprzednią wersję napisaną w jezyku C, która nadawała się do analizy szczególnych przypadków modeli. Zastosowany tam mechanizm funkcji odwrotnej (który poważnie ograniczał zastosowanie tego pakietu) został zastąpiony przez nową koncepcję opartą na funkcji mieszającej, co pozwoliło na modelowanie dowolnych modeli spełniających założenia Markowa. Narzędzie to okazało się bardzo pomocne przy obliczeniach modeli opisujących prace elementów sieci typu ATM. W bibliotece,oprócz prostych kolejek z serwisem typu Erlang, Exponent, znajdują sie obiekty opisujące pracę węzła ATM, takie jak:

1. Przesuwne okno (Sliding Window).
2. Kolejka priorytetowa typu Push Out.
3. Podwójny bufor ze wspólnym serwisem, z którego dane pobierane są naprzemiennie z zadaną ilością pakietów.
4. Kolejka typu Thresh Hold dla 2 klas klientów.
5. Elementy typu On-Off.

Dzięki wprowadzeniu tzw. dynamicznego wektora stanu program dopuszcza klasy klientów oraz elementy synchronizacji. Program nadaje sie do modelowania zarówno sieci zamkniętych, jak i otwartych.

LITERATURA

1. QNAP2 guide de l'utilisateur, SIMULOG, St-Quentin en Yvelines, 1992.
2. Stewart W. J.: Introduction to the Numerical Solution of Markov Chains", Princeton University Press, Princeton, New Jersey 1994.
3. Sidje R. B.: Parallel Algorithms for Large Sparse Matrix Exponentials: application to numerical transient analysis of Markov processes. PhD thesis, University of Rennes 1, July 1994.
4. SPARCompiler C++ 4.01, Tools.h++ Introduction & Reference Manual, 1994.

5. Jouaber B., Atmaca T.: Modelling the Sliding window Mechanism, ICC'98, Atlanta, Georgia, USA, June 1998.
6. Hamma S., P. Pecka P.: Markovian analysis of threshold based priority mechanism for frame-relay networks, Management in High Speed Networks, 2-5 November 1997, Dallas, Texas, USA.
7. Hamma S., Domanska J.: Analytical Model of Push-Out Mechanism for Frame Relay Switch: Comparative Study with Treshold Based Mechanism and Robustness, ICT '98, Porto Carras Greece, June 1998.
8. Niemiec M., Pecka P.: Metoda odwzorowania stanów w przestrzeń liczb naturalnych w markowowskich modelach systemów komputerowych, Archiwum Informatyki Teoretycznej i Stosowanej Tom 9 (1997) z. 1-4.
9. Tomasik A.: Wpływ modulowania źródła na funkcję autokowariancji generowanego strumienia danych, ZN Pol. Śl. s. Informatyka, z.30, Gliwice 1999.

Recenzent: Dr inż. Marcin Skowronek

Wpłynęło do Redakcji 7 kwietnia 1999 r.

Abstract

The paper discusses a useful, memory and time efficient, full scalable tool to build queuing networks and to separate all features related to the applied solution method, keeping the name of network items unchanged. Other tools for modeling, where network is described in a special language, e.g. ([1]) are not such flexible and can not be easily extended. The work is in progress and new solvers which are based on multi thread mechanism (Solaris thread library) has just been finished and will be presented in future.