ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: INFORMATYKA z. 34 Nr kol. 1381

Michał PASTUSZKA
Instytut Informatyki Teoretycznej i Stosowanej, PAN

MODELOWANIE STANÓW NIEUSTALONYCH W SIECI KOMPUTEROWEJ ZA POMOCĄ APROKSYMACJI DYFUZYJNEJ — ASPEKTY PROGRAMOWE I NUMERYCZNE

Streszczenie. Artykuł przedstawia bibliotekę klas napisaną w języku C++, umożliwiającą symulację sieci komputerowych. Do symulacji poszczególnych stanowisk wykorzystano analityczną metodę aproksymacji dyfuzyjnej. Oprogramowanie umożliwia analizę stanów przejściowych poszczególnych stanowisk. Oprogramowanie to będzie zintegrowane z modułami obliczeniowymi wykorzystujące inne metody analityczne oraz symulację, a sposób opisu modelu jest spójny z zasadami ustalonymi dla całości [12].

THE TRANSIENT STATES ANALISYS BY DIFFUSION APPROXIMATION IN THE COMPUTERS NETWORK – PROGRAMMATIC AND NUMERICAL ASPECTS

Summary. This paper presents a library of classes written in C++ which helps to simulate computer networks. The transient state of each station is computed separetly by diffusion approximation. This library will be connected with other modules which use other modelling methods.

1. Wstęp

Strumienie informacji przesyłane w szerokopasmowych sieciach mają niejednorodny charakter. Część z nich ma natężenie stałe (np. przesył głosu). Inne charakteryzują się bardzo zmiennym natężeniem przepływu informacji (np. obraz video, obliczanie rozproszone, multimedialne bazy danych), często przekraczającym wielokrotnie wartość średnią. Rozmiary buforów lub algorytmy sterowania ruchem w sieci powinny być tak dobrane,

aby przy określonych statystycznych własnościach strumieni prawdopodobieństwo strat pakietów spełniało przyjęte wcześniej warunki jakości usług telekomunikacyjnych. Ponieważ często analiza pojedynczego stanowiska nie wystarcza, aby przeprowadzić prawidłową ocenę systemu komputerowego, zachodzi konieczność przebadania zachowania całej sieci lub jej dużego fragmentu. W tym celu została stworzona biblioteka klas umożliwiająca symulację sieci dowolnie połączonych między sobą stanowisk.
Ocena stanów nieustalonych (przejściowych) za pomocą modeli symulacyjnych jest często zbyt długotrwała lub wręcz niemożliwa. Wynika stąd konieczność stosowania mniej kosztownych, choć często przybliżonych, modeli analitycznych.

W artykule opisano metodę analityczno-numeryczną wykorzystującą aproksymację dyfuzyjną. Następnie zostanie przedstawiony sposób obliczania stanu całej sieci. Opis najważniejszych klas pakietu oraz sposób ich użycia znajdą się w następnym rozdziale. W ostatniej części zostaną przedstawione dwa przykłady numeryczne.

2. Aproksymacja dyfuzyjna

Metoda aproksymacji dyfuzyjnej była już wielokrotnie wykorzystywana w modelowaniu i ocenie efektywności modeli stanowisk komputerowych, np. [2],[3],[4],[10]. Metoda ta pozwala analizować stanowiska obsługi o dowolnych strumieniach wejściowych i dowolnych czasach obsługi, czyli stanowiska, dla których dokładna analiza nie istnieje.
Podstawą tej przybliżonej metody jest zastąpienie dyskretnego procesu $N(t)$, odpowiadającego liczbie zadań w stanowisku obsługi, ciągłym procesem dyfuzji $X(t)$ o podobnych właściwościach.

$$\frac{\partial f(x,t;x_0)}{\partial t} = \frac{\alpha}{2}\frac{\partial^2 f(x,t;x_0)}{\partial x^2} - \beta\frac{\partial f(x,t;x_0)}{\partial x} \tag{1}$$

Rozwiązując równanie dyfuzji (1) z odpowiednio dobranymi współczynnikami i warunkami brzegowymi uzyskujemy funkcję gęstości procesu dyfuzji, która przybliża rozkład prawdopodobieństwa długości kolejki, a ten daje nam informacje o stopniu wykorzystania stanowiska, o czasie czekania na obsługę, o prawdopodobieństwie odrzucenia zadania z powodu przepełnienia kolejki itp. Jest to aproksymacja drugiego stopnia, tzn. metoda nie wymaga informacji o postaci rozkładu czasu obsługi i rozkładu czasu między zgłoszeniami w strumieniu wejściowym, które mogą być dowolne, lecz tylko o dwóch pierwszych momentach tych rozkładów – określają one parametry rozpatrywanego procesu dyfuzji. Warunek brzegowy ogranicza proces do niezerowej półosi (liczba zadań w systemie ni-

gdy nie jest ujemna), drugi warunek brzegowy występuje, jeżeli modelowana kolejka ma ograniczoną pojemność i limituje proces dyfuzji maksymalną długością kolejki. Warunki są sformułowane w postaci tzw. procesów elementanych: gdy proces dojdzie do zera, pozostaje w nim przez okres odpowiadający czasowi, przez który stanowisko jest puste i następnie przeskakuje do wartości 1 (po okresie bezczynności do stanowiska przychodzi pierwsze zadanie). Podobnie bariera ustawiona w punkcie odpowiadającym maksymalnej długości kolejki przetrzymuje proces dyfuzji przez czas odpowiadający dokończeniu aktualnie wykonywanego zadania, a następnie proces dyfuzji zmniejsza skokowo swą wartość o 1 (z systemu odeszło zakończone zadanie).

Metoda pozwala więc analizować stanowiska obsługi o dowolnych strumieniach wejściowych i dowolnych czasach obsługi, czyli stanowiska, dla których dokładna analiza nie istnieje. Co więcej, pozwala rozpatrywać pracę sieci takich stanowisk, wprowadzać klasy klientów o różnych czasach obsługi i różnej drodze pomiędzy stanowiskami w sieci, umożliwia opis pracy równoległych stanowisk obsługi (wówczas parametry procesu dyfuzji zależą od wartości tego procesu), a także wprowadzać różne regulaminy kolejki szeregującej zgłoszenia czekające na wykonanie, co pozwala modelować różne algorytmy zarządzania ruchem pakietów w sieciach komputerowych.

Najważniejszą zaletą aproksymacji dyfuzyjnej jest możliwość uzyskiwania rozwiązania w stanie nieustalonym – a taki właśnie stan panuje w sieci komputerowej, której obciążenie wciąż się zmienia – natężenie strumienia informacji wysyłanej przez użytkownika np. w przypadku przesyłania ruchomych obrazów może zmieniać się wielusetkrotnie. Najbardziej interesujące są wyniki modeli w momentach szczególnie nasilonego ruchu, kiedy maleje niezawodność przesyłu i rosną opóźnienia przesyłanych wiadomości.
Uzyskanie rozwiązania łączy się jednak z dość złożonymi obliczeniami. Metoda rozwiązywania równań dyfuzji z warunkami brzegowymi w postaci powrotów elementarnych podaje rozwiązanie, tj. zmienną w czasie funkcję gęstości procesu dyfuzji w postaci transformat Laplace'a, których oryginałów trzeba szukać numerycznie. Poza tym metoda zakłada stałe w czasie wartości parametrów dyfuzji, co, gdy nie jest spełnione, trzeba obchodzić szukając rozwiązania w krótkich przedziałach czasu, a rozwiązanie na końcu takiego przedziału stanowi warunek początkowy dla przedziału następnego.

3. Obliczanie stanu sieci stanowisk

Aproksymacja otwartej sieci systemów G/G/1/N polega na dekompozycji całego modelu sieci i została przedstawiona w [5]. Najpierw oblicza się parametry wejściowe dla poszczególnych stanowisk, a następnie każde stanowisko opisuje się niezależnie za pomocą przedstawionego powyżej modelu.

Parametrami wejściowymi pojedynczego stanowiska, które należy obliczyć to: λ (strumień wejściowy do stanowiska) oraz C_A^2 (współczynnik zmienności rozkładów odstępów czasów między klientami przychodzącymi do stanowiska).
W celu obliczenia parametru λ_i dla stanowiska i korzystamy z równania ruchu

$$\lambda_i = \lambda_{0i} + \sum_{j=1}^{M} \mu_j \rho_i r_{ji}, \tag{2}$$

gdzie:

λ_{0i} - intensywność strumienia dochodzącego do stanowiska i spoza sieci,

r_{ji} - prawdopodobieństwo przejścia pomiędzy stanowiskiem j oraz stanowiskiem i.

Do obliczenia współczynnika zmienności C_{Ai}^2 korzystamy z układu dwu równań

$$C_{Di}^2 = C_{Ai}^2(1-\rho_i) + \rho_i^2 C_{Bi}^2 + \rho_i(1-\rho_i) \tag{3}$$

$$C_{Aj}^2 = \frac{1}{\lambda_j}\sum_{i=1}^{M} r_{ij}\mu_i\rho_i[(C_{Di}^2 - 1)r_{ij} + 1] + \frac{C_{0j}^2\lambda_{0j}}{\lambda_j}, \tag{4}$$

gdzie:

C_{0j}^2 - współczynnik zmienności strumienia dochodzącego do stanowiska i spoza sieci.

Po skonstruowaniu powyższych trzech równań i rozwiązaniu tego układu równań otrzymujemy szukane wartości, które stanowią parametry wejściowe dla poszczególnych stanowisk. Następnym etapem jest posłużenie się odpowiednio dobranymi do potrzeb symulacji modelami stanowisk. Zastosowanie tych modeli do każdej ze stacji z osobna umożliwia obliczenie stanu całej sieci.

4. Implementacja metody aproksymacji dyfuzyjnej

Opisana metoda aproksymacji dyfuzyjnej oraz sposób rozwiązywania sieci stanowisk zostały zaimplementowane jako zbiór klas w ramach pakietu oprogramowania. Na podstawie przeprowadzonej analizy zostało wyodrębnionych kilka klas reprezentujących odrębne zagadnienia.

Metoda aproksymacji dyfuzyjnej została zawarta w klasie **GG1N**. Klasa **ExperimentsDevice** została stworzona, aby ułatwić użytkownikowi przeprowadzanie serii eksperymentów na pojedynczym stanowisku. Wreszcie klasa **diffusionTSolver** umożliwia dokonanie symulacji całej sieci stanowisk obsługi.

4.1. Klasa diffusionTSolver

Klasa **diffusionTSolver** należy do rodziny klas (**Solver**), które służą do analizowania sieci komputerowych używając różnych modeli kolejkowych, utworzonych za pomocą pakietu. Klasa ta wykorzystuje opisaną wcześniej metodę aproksymacji dyfuzyjnej. Wszystkie obliczenia związane z wykorzystaniem samej metody aproksymacji dyfuzyjnej wykonywane są przez klasy **ExpDev_GG1N** oraz **GG1N**, które zostaną opisane poniżej. Najbardziej istotne metody tej klasy, z punktu widzenia użytkownika, to:

setCountParam ustawia dla wskazanej stacji μ oraz C_b^2,

setTime ustawia nowy czas, dla którego mają być wykonane obliczenia stanu sieci,

setdT ustawia kwant czasu, co który bedą wykonywane kolejne obliczenia stanu sieci,

lock uruchamia powtórne obliczenie wartości λ i C_A^2 (dane wejściowe dla stanowiska), co jest równoznaczne z wymianą informacji między stacjami o stanie, w jakim znajdują się one w danej chwili,

runExperiments bezpośrednie uruchomienie obliczeń (wykonywane są po kolei obliczenia dla wszystkich stacji),

setInitState ustawia dla wskazanej stacji jej stan początkowy, tzn. ilość klientów znajdujących się w kolejce w trakcie rozpoczęcia obliczeń.

4.2. Klasa GG1N

Klasa **GG1N** ma za zadanie obliczenie pojedynczego stanowiska obsługi, które jest modelem typu G/G/1/N. Wynikiem działania metod tej klasy jest otrzymanie gęstości prawdopodobieństwa stanu kolejki w dowolnym czasie. Klasa ta dziedziczy z 2 klas: **DiffusionServer** oraz **ClassLT**. Pierwsza z nich jest klasą wirtualną, która przechowuje najważniejsze parametry stanowiska, np. λ, μ, C_a^2, C_b^2 lub t, oraz potrafi dokonywać operacji na nich. Druga klasa umożliwia korzystanie z transformaty Laplace'a, którą zastosowano w trakcie obliczeń numerycznych.

Wykorzystując mechanizm dziedziczenia z klas **DiffusionServer** i **ClassLT**, istnieje możliwość tworzenia kolejnych klas, które, stosując metodę aproksymacji dyfuzyjnej, będą mogły modelować także inne stanowiska niż G/G/1/N.

4.3. Klasa ExperimentsDevice

Klasa **ExperimentsDevice** została skonstruowana tak, aby użytkownikowi ułatwić korzystanie z obiektów klasy **GG1N**, a w szczególności zapewnia możliwość przeprowadzania i sterowania procesem obliczeń. Najistotniejsze metody to:

setParamExp ustawia parametry pracy stanowiska; między innymi: μ, λ, N, C_a^2, C_b^2,

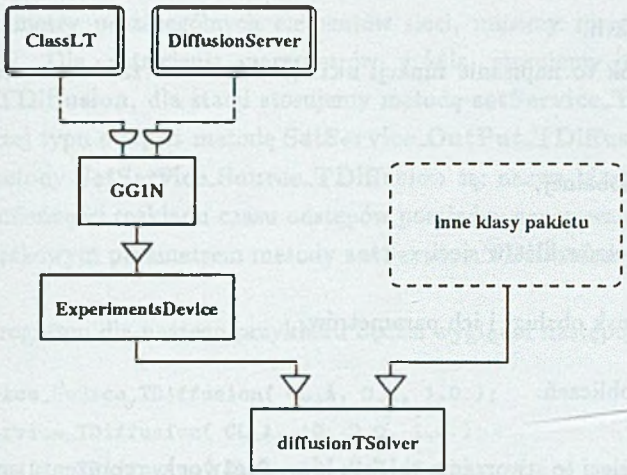
runExp wykonanuje pojedyncze obliczenie,

runSeriesExps wykonanuje serię symulacji; ilość kroków ustalona zostaje wcześniej przez metodę setParamSeriesExps,

setInitState ustawia dla wskazanej stacji jej stan początkowy, tzn. ilość klientów znajdujących się w kolejce na początku obliczeń.

Klasa ta umożliwia przeprowadzanie symulacji tylko dla pojedynczej stacji. Dlatego, jeśli użytkownik jest zainteresowany wyłącznie symulacją sieci stanowisk, a nie pojedynczego stanowiska, nie musi używać klas **GG1N** ani **ExperimentsDevice**, a jedynie klasę **diffusionTSolver**.

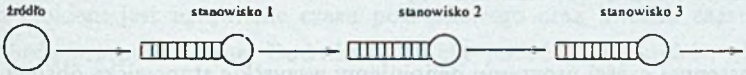
Na rysunku 1 został przedstawiony fragment struktury obiektów pakietu, związany z wykorzystaniem aproksymacji dyfuzyjnej.



Rys. 1. Fragment struktury obiektów
Fig. 1. Fragment of the objects structure

5. Sposób użycia pakietu przez użytkownika

W celu przeprowadzenia symulacji sieci komputerowej należy napisać program w języku C++ z użyciem opisanych wcześniej klas, skompilować go, a następnie uruchomić. Etapy postępowania przy pisaniu programu są podobne jak przy korzystaniu z innych metod rozwiązywania sieci (np. MVAP, Markow, symulacja itp.) zawartych w pakiecie. Schemat postępowania zostanie zilustrowany przykładem (rys.2).



Rys. 2. Przykład sieci stanowisk
Fig. 2. Example of the network

Sieć składa się z trzech stacji oraz źródła generującego klientów. Chcemy zaobserwować, jak w czasie zmienia się stan kolejek w poszczególnych stacjach, w sytuacji zmieniającego sie natężenia przychodzących klientów.

Parametry źródła (`zrodlo`): $\lambda = 0.1$ dla $t \in [0, 10]$, $\lambda = 4.0$ dla $t \in [10, 20]$.

Parametry wszystkich stanowisk (`stacja1`, `stacja2` i `stacja3`) są takie same i wynoszą: $N = 10$, $\mu = 2$, $C_a^2 = 1$.

Pierwszy krok polega na udostępnieniu programowi niezbędnych bibliotek, które znajdują

się w pliku **amok.h**.

Następny krok to napisanie funkcji `main()`. Musi ona zawierać cztery główne elementy:

1. definicja sieci globalnej,
2. definicja parametrów klas w sieci,
3. definicje stanowisk obsługi i ich parametrów,
4. opis przebiegu obliczeń.

ad 1) Definicja sieci to stworzenie obiektu klasy **Network**, reprezentującego całość sieci stanowisk:

```
Network net("Nazwa_sieci",1);
```

Pierwszy parametr to *nazwa sieci*, drugi to *liczba klas klientów* w sieci.

ad 2) Definicja parametrów klas sprowadza się do nazwania wszystkich klas klientów, które występują w sieci. Ponieważ nasz model uwzględnia tylko jedną klasę, więc w naszym przykładzie napiszemy:

```
net.SetClassPar( CL_A, 5, "Klasa 1" );
```

ad 3) W następnej części programu definiujemy wszystkie stanowiska obsługi oraz źródła, które występują w naszej sieci, jednocześnie dołączając je do wcześniej zdefiniowanej sieci (`net`).

```
SimpleServer source("zrodlo", net);
SimpleServer station1("1",    net);
SimpleServer station2("2",    net);
SimpleServer station3("3",    net);
```

Każde zadanie, które opuszcza sieć, musi być kierowane do stacji typu **output**.

```
SimpleServer out("out", net);
```

Ustawiając parametry poszczególnych elementów sieci, musimy rozróżnić i uwzględnić ich rodzaje. Dla ustawienia parametrów źródła stosujemy metodę **SetService_Source_TDiffusion**, dla stacji stosujemy metodę **setService_TDiffusion** i dla stacji pomocniczej typu **output** metodę **SetService_OutPut_TDiffusion**.
Parametrami metody **SetService_Source_TDiffusion** są: nazwa klasy, częstotliwość i współczynnik zmienności rozkładu czasu odstępów pomiędzy generowanymi klientami. Czwartym, dodatkowym parametrem metody **setService_TDiffusion** jest długość bufora.
Ten fragment programu dla naszego przykładu będzie wyglądał następująco:

```
source.SetService_Source_TDiffusion( CL_A, 0.1, 1.0 );
station1.setService_TDiffusion( CL_A, 10, 2.0, 1.0 );
station2.SetService_TDiffusion( CL_A, 10, 2.0, 1.0 );
station3.SetService_TDiffusion( CL_A, 10, 2, 1.0 );
out.SetService_OutPut_TDiffusion( CL_A );
```

Metoda **SetTransition** określa, do jakich stacji i z jakimi prawdopodobieństwami mają przechodzić klienci z poszczególnych klas (w naszym przypadku dotyczy to jednej klasy):

```
source.SetTransition( CL_A, station1, 1.0 );
station1.SetTransition( CL_A, station2, 1.0 );
station2.SetTransition( CL_A, station3, 1.0 );
station3.SetTransition( CL_A, out, 1.0 );
```

ad 4) Przebieg procesu obliczeń steruje się za pomocą metod klasy **diffusionTSolver**. Pierwszym krokiem jest ustawienie czasu początkowego oraz kwantu czasu, co który obliczenia bedą przeprowadzane. Dodatkowo należy podać liczbę kroków obliczeń bez wymiany informacji pomiędzy stacjami.

```
sol.setTime( 0.1 );
sol.setdT( 1 );
sol.setNbExp( 1 );
```

Następnie, należy ustawić wartości stanów początkowych stacji, tzn. początkowe długości kolejek oraz parametry źródła (λ i C_a^2).

```
sol.setInitState( "1", 5 );
sol.setInitState( "2", 5 );
sol.setInitState( "3", 5 );
sol.setCountParam( "zrodlo", 0.1, 1.0 );
```

Po podaniu wszystkich potrzebnych danych możemy przejść do uruchomienia obliczeń dla $t = 1..10$.

```
sol.lock();
for( i=0; i<10; i++ )
{
   sol.runExperiments();
   sol.lock();
}
```

Po zakończeniu pierwszej części obliczeń należy zmienić parametry źródła na nowe. Dodatkowo możemy zwiększyć częstotliwość pomiarów poprzez zmniejszenie kwantu czasu, co który będą przeprowadzane obliczenia, a następnie ponownie je uruchomić.

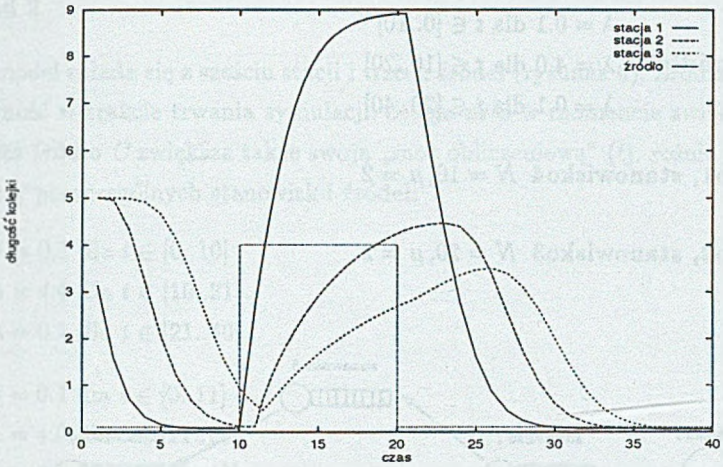
```
sol.setCountParam( "zrodlo", 4.0, 1.0 );
sol.lock();
sol.setdT( 0.5 );
for( i=0; i<20; i++ )
{
   sol.runExperiments();
   sol.lock();
}
```

Po dokonaniu obliczeń dla $t = 11..20$ ponownie zmieniamy parametry źródła i uruchamiamy program, aby uzyskać wyniki dla $t = 21..30$.

```
sol.setCountParam( "zrodlo", 0.1, 1.0 );
sol.lock();
sol.setdT( 1 );
for( i=0; i<20; i++ )
{
   sol.runExperiments();
   sol.lock();
}
```

5.1. Wyniki

W trakcie obliczeń wyniki są generowane automatycznie i zapisywane do plików. Dla każdej stacji jest tworzony osobny plik o nazwie takiej, jaką nadano stacji w konstruktorze klasy **SimpleServer**.



Rys. 3. Wyniki dla przykładu w postaci średniej liczby klientów w stanowiskach
Fig. 3. Result for the example as the mean value of clients in the stages

Dane w pliku są podzielone na rekordy, które odpowiadają poszczególnym krokom obliczeń. Każdy rekord zawiera wiersz informacyjny (czas, μ, λ, długość bufora itp.) oraz wiersze, w których zapisane są stany kolejki (liczba klientów w kolejce) oraz ich prawdopodobieństwa wystąpienia. Dane, zawarte w tym pliku, mogą być następnie dowolnie przetworzone przez użytkownika. Na przykład, może zostać obliczona średnia długość kolejki w kolejnych momentach symulacji (rys. 3).

6. Przykłady numeryczne

W tej części zostaną zaprezentowane 2 przykłady wykorzystania pakietu do rozwiązywania modeli kolejkowych sieci.

Przykład 1

Model składa się z czterech stanowisk obsługi i dwu źródeł (rys.4). Przez 10 jednostek czasu źródła generują klientów z częstotliwością $\lambda = 0.1$, a następnie zwiększają częstotliwość generowanych klientów do $\lambda = 4.0$.
Oto parametry poszczególnych stanowisk:

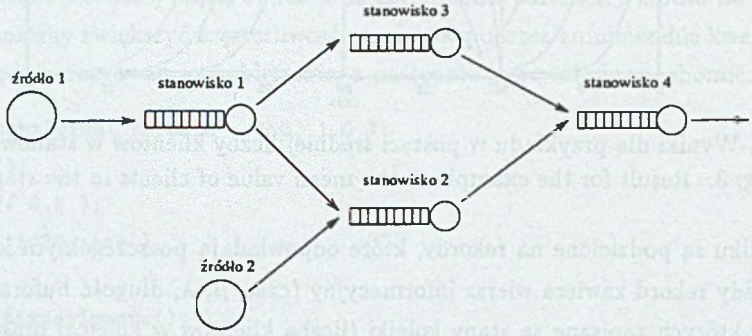
$\lambda = 0.1$ dla $t \in [0..10]$

źródło1, źródło2 $\lambda = 4.0$ dla $t \in [10..20]$

$\lambda = 0.1$ dla $t \in [20..40]$

stanowisko1, stanowisko4 $N = 10, \mu = 2$

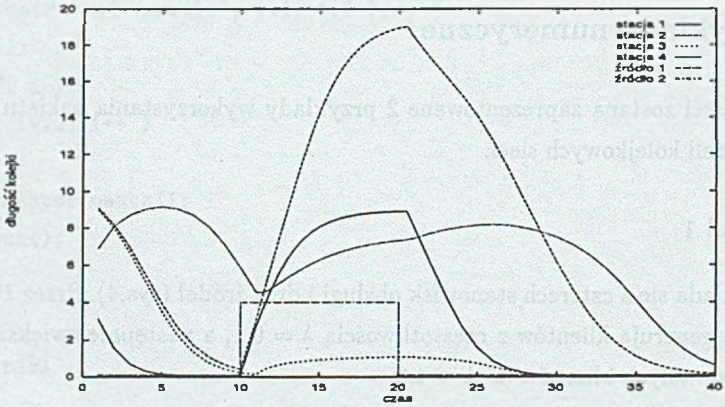
stanowisko2, stanowisko3 $N = 20, \mu = 2$



Rys. 4. Sieć stanowisk dla przykładu 1

Fig. 4. The network for the example 1

Wyniki zostały zaprezentowane na rys. 5.



Rys. 5. Wyniki dla przykładu 1

Fig. 5. Results for the example 1

Przykład 2

Kolejny model składa się z sześciu stacji i trzech źródeł (rysunek 6). Źródła zmieniają swoją aktywność w trakcie trwania symulacji. Stacja nr 6 w momencie zwiększenia aktywności przez źródło C zwiększa także swoją „moc obliczeniową" (tj. rośnie jego μ). Oto parametry poszczególnych stanowisk i źródeł:

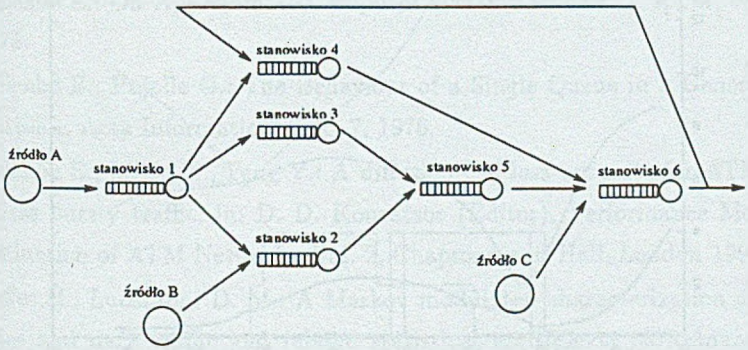
źródłoA
$\lambda = 0.1$ dla $t \in [0..10]$
$\lambda = 4.0$ dla $t \in [10..21]$
$\lambda = 0.1$ dla $t \in [21..40]$

źródłoB
$\lambda = 0.1$ dla $t \in [0..11]$
$\lambda = 4.0$ dla $t \in [11..20]$
$\lambda = 0.1$ dla $t \in [20..40]$

źródłoC
$\lambda = 0.1$ dla $t \in [0..15]$
$\lambda = 2.0$ dla $t \in [15..22]$
$\lambda = 4.0$ dla $t \in [22..30]$
$\lambda = 2.0$ dla $t \in [30..31]$
$\lambda = 0.1$ dla $t \in [31..40]$

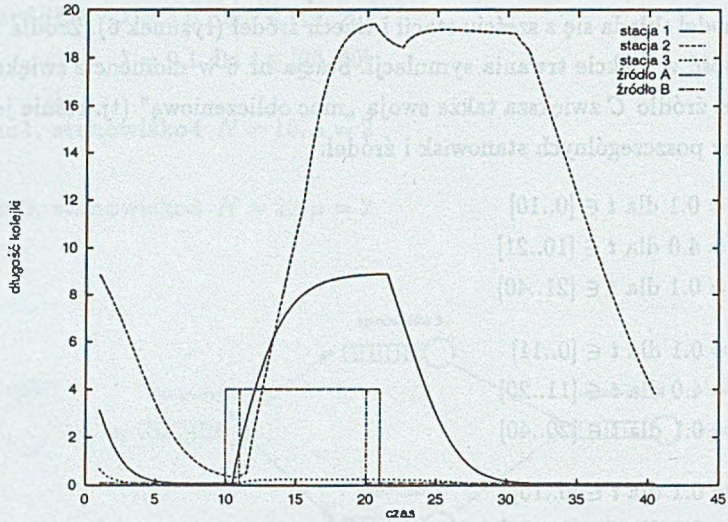
stanowisko1, stanowisko4 $N = 10, \mu = 2$

stanowisko2, stanowisko3 $N = 20, \mu = 2$

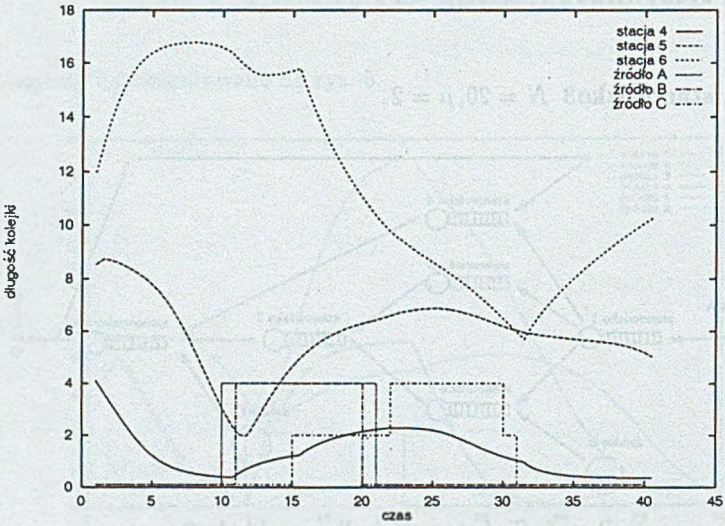


Rys. 6. Sieć stanowisk dla przykładu 2
Fig. 6. Network for the example 2

Wyniki dla tego przykładu zostały zamieszczone na rysunkach 7 i 8.



Rys. 7. Wyniki stanowisk 1, 2, 3 dla przykładu 2
Fig. 7. Results for stages 1, 2, 3 for the exaple 2



Rys. 8. Wyniki stanowisk 4, 5, 6 dla przykładu 2
Fig. 8. Results for stages 4, 5 and 6 for the exaple 2

7. Wnioski

Współczesne sieci komputerowe charakteryzują się niejednorodnym natężeniem przepływu strumieni informacji. Powoduje to konieczność badania poszczególnych jej elementów składowych, a także algorytmów sterowania pracą sieci.

Zaproponowany pakiet oprogramowania umożliwia przeprowadzanie takich badań. Zastosowanie języka C++ ułatwia konstruowanie nowych modeli stanowisk i algorytmów kontroli połączeń oraz umożliwia dołączenie ich do istniejącej już sieci.

Użycie metody aproksymacji dyfuzyjnej skraca czas przeprowadzania obliczeń oraz zmniejsza wymagania sprzętowe w porównaniu do metod symulacyjnych lub innych metod analitycznych.

LITERATURA

[1] Cox R. P., Miller H. D.: The Theory of Stochastic Processes. Chapman and Hall, London 1965.

[2] Czachórski T.: A method to solve diffusion equation with instantaneous return processes acting as boundary conditions. Bulletin of Polish Academy of Sciences, Technical Sciences 41 (1993), no. 4.

[3] Czachórski T.: Modele kolejkowe systemów komputerowych. Skrypt Politechniki Śląskiej, nr 1844, Gliwice 1994.

[4] Gelenbe E.: On Approximate Computer Systems Models. J. ACM, vol. 22, no. 2, 1975.

[5] Gelenbe E., Pujolle G.: The Behaviour of a Single Queue in a General Queueing Network. Acta Informatica, Fasc. 7, 1976.

[6] Gelenbe E., Mang X., Feng Y.: A diffusion cell loss estimate for ATM with multiclass bursty traffic. in: D. D. Kouvatsos (Editor), Performance Modelling and Evaluation of ATM Networks, Vol. 2, Chapman and Hall, London 1996 (in print).

[7] Heffes H., Lucantoni D. M.: A Markov modulated characterization of packetized voice and data traffic and related statistical multiplexer parformance. IEEE J. SAC, SAC-4(6), pp. 856-867, 1986.

[8] Nagai K., Akimaru H., Okuda T.: A simplified performance evaluation for bursty multiclass traffic in ATM systems. IEEE Transactions on Communications, COM-42 No. 5, pp. 2078-2083, 1994.

[9] Newell G. F.: Applications of Queueing Theory. Chapman and Hall, London 1971.

[10] Kobayashi H., Ren Q.: A diffusion approximation analysis of an ATM statistical multiplexer with multiple state solutions: Part I: Equilibrium state Solutions. Proc. ICC'93, pp. 1047-1053, 1993.

[11] Sriram K., Whitt W.: Characterizing superposition arrival processes in packet multiplexers for voice and data. IEEE J. SAC, SAC-4(6), pp. 833-846, 1986.

[12] Draga M., Pecka P., Rewer A.: Zorientowany obiektowo język do opisu kolejkowych modeli sieci komputerowych. III Seminarium Sieci Komputerowe, Gliwice, 1996.

Recenzent: Dr inż. Ewa Starzewska-Karwan

Wpłynęło do Redakcji 29 grudnia 1997 r.

Abstract

The present computers networks are characterized by very variable intensity of the information streams. It is neccessary to analize not only an individual station, but the hole network or at least its big part.

In this paper a library of C++ classes is presented. It helps to model a computer network, especially its transient behaviour.

A single station is analized by a diffusion aproximation method. Entire network is computed by a method presented in [5].

There are three numerical examples (see fig. 2, 4, 6). Results are presented at fig. 3, 5, 7, 8.