Technologie sieciowe

sprawozdanie lista 2

Autor: Jakub Duda 236778

1. Cel

Przedmiotem pierwszej części zadania było napisanie programu szukającego niezawodności rozważanego modelu sieci. Sprawdzanie jak zmieni się badana niezawodność po modyfikacji krawędzi z różnymi prawdopodobieństwami nierozerwania w dowolnym interwale.
Druga część zadania polegała na zaproponowaniu topologi grafu, który będzie miał 10 wierzchołków i mniej niż 20 krawędzi oraz macierz natężeń strumienia pakietów, funkcję przepustowości, funkcje przepływu. Następnie napisaniu programów, dzięki którym będzie można testować zaproponowany przykład tzn. który dla wybranych reprezentacji zadanych odpowiednimi macierzami będzie obliczał średnie opóźnienie pakietu. Kolejno trzeba było napisać program, który będzie szacował niezawodność sieci przy zadanym maksymalnym opóźnieniu i prawdopodobieństwie nierozerwalności.

2. Realizacja

2.1 Część pierwsza – spójność sieci

Program został napisany w języku **Java** przy pomocy biblioteki **jgrapht**. Uruchomić go można z parametrem "`--load`", wtedy program będzie wczytywał graf z pliku, poprzez wywołanie metody **loadGraph("***data.txt*"**)**. Plik tekstowy powinien zawierać w pierwszej linii ilość wierzchołków, w kolejnej linii dodajemy krawędzie, wpisując wierzchołek startowy, końcowy i prawdopodobieństwo. Drugim sposobem jest uruchomienie bez podania parametru, wtedy program stworzy zaimplementowany graf. Na początku z metody `createWeightedGraph()` program tworzymy graf, który ma 20 wierzchołków i 19 krawędzi z prawdopodobieństwem niezawodności ***h***=0.95. Po stworzeniu grafu i nadaniu mu wszystkich parametrów testujemy model sieci. W tym celu wywołujemy metodę `test(interwal)` z liczbą interwałów, jakie chcemy wykonać. Metoda kopiuje nasz graf i dla każdej krawędzi losuje liczbę z przedziału od 0 do 1, jeśli wylosowana liczba była większa niż prawdopodobieństwo niezawodności krawędzi usuwaliśmy tę krawędź. Następnie sprawdzaliśmy spójność modelu. Sprawdzając czy z wybranego wierzchołka da się dotrzeć do dowolnego innego przy pomocy wbudowanej funkcji biblioteki **jgrapht** (`ConnectivityInspector`). Przypadki, gdy graf nie był spójny, są zliczane. Program wyświetla prawdopodobieństwo niezawodności sieci przed podzielenie różnicy wszystkich wywołanych przypadków i tych gdzie graf nie był spójny przez liczbę wszystkich przypadków.

2.2 Część druga – opóźnienie

Program został napisany w języku **Java** przy pomocy biblioteki **jgrapht**. Zaproponowana przeze mnie topologia jest grafem Petersena. Wybrałem ten graf, ponieważ spełnia on wszystkie warunki zadania. Liczba wierzchołków jest równa dziesięciu, natomiast liczba krawędzi wynosi 15, a co najważniejsze droga między dwoma dowolnymi wierzchołkami nie jest dłuższa niż dwa. Uruchamiając program z parametrem „`--load`” parametry zostaną wczytywane z pliku za pomocą metody `loadGraph("data.txt")`. Na początku podaje się liczbę wierzchołków. Następnie wprowadza się krawędzie, podając dwa wierzchołki, które łączy oddzielając je „-”. Po dodaniu wszystkich krawędzi powinna znaleźć się litera *C*, a pod nią przepustowości danych krawędzi w następujący sposób *1-2-10000000* gdzie *1* jest wierzchołkiem, z którego wychodzi

krawędź, 2 jest wierzchołkiem, do którego wchodzi krawędź, a następna liczba reprezentuje przepustowość. Kolejno występuje literka N, a po niej natężenia podane w taki sam sposób jak przepustowość, gdzie ostatnia liczba jest natężeniem. Model sieci może być także tworzony przez program. W tym celu należy uruchomić program bez parametrów. Wykorzystuje się wtedy następujące metody `createWeightedGraph()`, `createCapacityMatrix()`, `createIntensityMatrix()`, które kolejno tworzą graf, macierz przepustowości, macierz natężeń.

Po stworzeniu grafu sprawdzamy czy przepustowość jest większa niż faktyczna liczba pakietów wprowadzana do kanału komunikacyjnego. W tym celu wywołujemy metodę `traverseGraph(G)`, gdzie parametr *G* jest stworzonym przez nas grafem. W tej metodzie dla wszystkich różnych wierzchołków znajdujemy najkrótszą drogę, poprzez wbudowaną w bibliotekę **jgraph** funkcję **DijkstraShortestPath.** Następnie iterując po krawędziach znalezionej drogi, sumujemy im natężenie, zadane wcześniej w macierzy natężeń i sprawdzamy czy nie jest ono większe od przepustowości wierzchołka. Jeśli tak to zwracamy `false` i komunikat o niepowodzeniu. W przeciwnym wypadku `true`.

Opóźnienie ***T*** liczymy, wywołując metodę `delay(G)`. Podając jej jako argument stworzony lub wczytany graf. Metoda ta w pętli zlicza wszystkie natężenia z zadanej macierzy natężeń. Następnie iterujemy po wszystkich krawędziach. Jeśli maksymalna liczba przysłanych pakietów jest większa niż faktyczna liczba pakietów, to dodajemy do ***SUM_e*** natężenie krawędzi podzielone przez różnicę maksymalnej przepustowości pakietów i faktycznej liczby pakietów. Program zwraca opóźnienie ***T***, które jest wynikiem działania ***1/n*SUM_e***, gdzie ***n*** to suma elementów macierzy natężeń.

Następnie program w dowolnym przedziale czasowym szacuje niezawodność zadanej sieci. Wykonanie testu odbywa się przez wywołanie metody `test(interwal, p, T_max)` z trzema parametrami. Pierwszym jest ilość wykonywanych powtórzeń, drugim z parametrów jest zadane zmiennoprzecinkową liczbą prawdopodobieństwa niezawodności krawędzi, trzecim parametrem jest maksymalny czas opóźnienia. Program w pętli dla zadanej liczby symulacji najpierw kopiuje graf, kolejno dla wszystkich krawędzi losuje liczbę z przedziału od 0 do 1, jeśli będzie większa od zadanego prawdopodobieństwa, usuwa krawędź. Po przejściu przez wszystkie krawędzie program sprawdza czy graf jest dalej spójny za pomocą funkcji `isGraphConnected()`. Jeśli tak to sprawdza metodą `traverseGraph()` czy wszystkie pozostałe krawędzie mają przepływ mniejszy od przepustowości. Następnie sprawdza czy opóźnienie nie jest większe od maksymalnego W przypadku gdy któraś z metod zwróci *false*, dodajemy przypadek jako nieudany. Program zwraca prawdopodobieństwo nierozspójnienia sieci, które jest ilorazem liczby udanych symulacji i wszystkich symulacji.

3. Wnioski

Podczas symulacji sieci pierwszym programem dokonaliśmy 1000 prób. Dało nam to 651 przypadków, gdy sieć była niespójna. Prawdopodobieństwo niezawodności tego modelu sieci wyniosła 0.3848.

Następnie w taki sam sposób badaliśmy, jak zmieni się prawdopodobieństwo po dodaniu krawędzi ***e***(1,20) z prawdopodobieństwem ***h***=0.95. Dodanie tej jednej krawędzi spowodowało wzrost prawdopodobieństwa spójności sieci do 0.7378.

Następne badanie odbywa się po dodaniu krawędzi ***e***(5,15) z prawdopodobieństwem ***h(e***(5,15))=0.7 i krawędzi ***e(***1,10) z prawdopodobieństwem ***h(e***(1,10))=0.8. Szukając prawdopodobieństwa niezawodności sieci, otrzymaliśmy wynik równy 0.868.

Kolejno dodajemy 4 losowe krawędzie wszystkie z prawdopodobieństwem ***h***=0.4. Zwrócony wyniki prawdopodobieństwa badanego modelu wyniósł 0.9186.

Program pierwszy służy do badania wpływu krawędzi i prawdopodobieństwa ich niezawodności na spójność modelu sieci. Dla zadanego grafu możemy stworzyć symulację modelu sieci, dodając wierzchołki i krawędzie uzyskując w wyniku prawdopodobieństwo nierozpojenia zadanej sieci.

Podczas symulacji sieci drugim programem badamy opóźnienie **T** dla średniego pakietu ***m***, przepustowości ***c*** i macierzy natężeń ***N*.**			
m=1500b	***c*∈*(***10Mb**,**1Gb**)**	***N*∈**(10,100)	**T**=43,62ms
m=2000b	***c*∈*(***10Mb**,**1Gb**)**	***N*∈**(10,100)	**T**=60,00ms
m=2500b	***c*∈*(***10Mb**,**1Gb**)**	***N*∈**(10,100)	**T**=77,53ms
m=3000b	***c*∈*(***10Mb**,**1Gb**)**	***N*∈**(10,100)	**T**=96,37ms
m=2000b	***c*∈*(***10Mb**,**1Gb**)**	***N*∈**(1,10)	**T**=27,53ms
m=2000b	***c*∈*(***10Mb**,**1Gb**)**	***N*∈**(10,30)	**T**=36,99ms
m=2000b	***c*∈*(***10Mb**,**1Gb**)**	***N*∈**(30,60)	**T**=40,74ms
m=2000b	***c*∈*(***10Mb**,**1Gb**)**	***N*∈**(60,100)	**T**=48,60ms

Następnie badamy prawdopodobieństwo nierozspójnienia sieci **P** dla zadanego modelu sieci przy interwale **i,** prawdopodobieństwie **p** i maksymalnym opóźnieniu **T**			
i=1000	**P**=0.6	**T**=200ms	**P**=0.428
i=1000	**P**=0.8	**T**=200ms	**P**=0.905
i=1000	**P**=0.4	**T**=200ms	**P**=0.056
i=1000	**P**=0.7	**T**=20ms	**P**=0.179
i=1000	**P**=0.7	**T**=30ms	**P**=0.192
i=1000	**P**=0.7	**T**=40ms	**P**=0.258
i=1000	**P**=0.7	**T**=50ms	**P**=0.707

Dzięki drugiemu programowi możemy testować i sprawdzać modele sieci pod względem ich niezawodności i czasom opóźnienia. Nadając rożne parametry.