Technologie Sieciowe. Lista 2

Valerii Bahrov

April 2020

Zadanie

1. Rozważmy model sieci $S =< G, H >$. Przez $N = [n(i, j)]$ będziemy oznaczać macierz natężeń strumienia pakietów, gdzie element $n(i, j)$ jest liczbą pakietów przesyłanych (wprowadzanych do sieci) w ciągu sekundy od źródła $v(i)$ do ujścia $v(j)$.

   - Zaproponuj topologię grafu G ale tak aby żaden wierzchołek nie był izolowany oraz aby: $|V| = 20$, $|E| < 30$. Zaproponuj N oraz następujące funkcje krawędzi ze zbioru H: funkcję przepustowości c (rozumianą jako maksymalną liczbę bitów, którą można wprowadzić do kanału komunikacyjnego w ciągu sekundy), oraz funkcję przepływu a (rozumianą jako faktyczną liczbę pakietów, które wprowadza się do kanału komunikacyjnego w ciągu sekundy). Pamiętaj aby funkcja przeplywu realizowała macierz N oraz aby dla każdego kanału e zachodziło: $c(e) > a(e)$.
   - Niech miarą niezawodności sieci jest prawdopodobieństwo tego, że w dowolnym przedziale czasowym, nierozspójniona sieć zachowuje $T < T_{max}$, gdzie: $T = \frac{1}{G} * \sum_{e \in E} \left(\frac{a(e)}{\frac{c(e)}{m} - a(e)} \right)$, jest średnim opóźnieniem pakietu w sieci, $\sum_{e \in E}$ oznacza sumowanie po wszystkich krawędziach e ze zbioru E, G jest sumą wszystkich elementów macierzy natężeń, a m jest średnią wielkością pakietu w bitach. Napisz program szacujący niezawodność takiej sieci przyjmując, że prawdopodobieństwo nieuszkodzenia każdej krawędzi w dowolnym interwale jest równe p. Uwaga: N, p, T_{max} oraz topologia wyjsciowa sieci są parametrami.
   - Przy ustalonej strukturze topologicznej sieci i dobranych przepustowościach stopniowo zwiększaj wartości w macierzy natężeń. Jak będzie zmieniać się niezawodność zdefiniowana tak jak punkcie poprzednim ($Pr[T < T_{max}]$).
   - Przy ustalonej macierzy natężeń i strukturze topologicznej stopniowo zwiększaj przepustowości. Jak będzie zmieniać się niezawodność zdefiniowana tak jak punkcie poprzednim ($Pr[T < T_{max}]$).

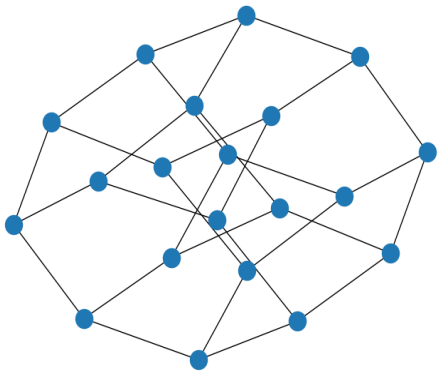
- Przy ustalonej macierzy natężeń i pewnej początkowej strukturze topologicznej, stopniowo zmieniaj topologię poprzez dodawanie nowych krawędzi o przepustowościach będących wartościami średnimi dla sieci początkowej. Jak będzie zmieniać się niezawodność zdefiniowana tak jak punkcie poprzednim ($Pr[T < T_{max}]$).

2. Napisz sprawozdanie zawierające opis zrealizowanych programów, komentarz do przeprowadzonych badań oraz wnioski.

Topologia grafu

Do wykonania zadania będziemy stosować Pythona. Jako podstawowy graf weźmiemy graf Desargues'a, bo chociaż on nie spełnia wszystkich warunków, ale jest bardzo przydatny do badania właściwości naszej topologii. Na początku stworzymy graf i macierz $N[n(i,j)]$. Funkcja C to jest przepustowość pomiędzy dowolnymi wierzchołkami i oraz j, jej wartością jest stała. Funkcja $a\_function$ okresla ile danych jest przesyłane bezpośrednim połączeniem sąsiadujących komórek i oraz j:

```
C = 2048
NODES = 20
def create_graph():
    graph = nx.desargues_graph()
    nx.set_edge_attributes(graph, 0, 'a')
    nx.set_edge_attributes(graph, C, 'c')
    return graph
```



```
def a_function(graph, matrix):
    nx.set_edge_attributes(graph, 0, 'a')
    for i, row in enumerate(matrix):
        for j, n in enumerate(row):
            path = nx.shortest_path(graph, i, j)
            for k in range(len(path) - 1):
                graph[path[k]][path[k + 1]]['a'] += n
```

```
def create_matrix(nodes):
    size = nodes
    matrix = np.zeros((size, size), dtype=int)
    for i in range(size):
        for j in range(size):
            if i != j:
                matrix[i][j] = rand.randint(10, 20)
    return matrix
```

Oszacujmy niezawodność sieci

Najpierw zdefiniujmy funkcję T która będzie liczyła średnie opóźnienie pakietu w sieci:

```
def delay(graph, matrix):
    G = matrix.sum()
    m = 1
    return (1/G * sum([graph.get_edge_data(*e).get('a') /
                       ((graph.get_edge_data(*e).get('c') / m)
                       - graph.get_edge_data(*e).get('a')) for e in
     graph.edges()]))
```

Teraz możemy zdefiniować funkcję, która będzie liczyła niazawodność sieci, gdzie prawdopodobieństwo nieuszkodzenia każdej krawędzi w dowolnym interwale jest równe p oraz T_{max} to jest maksymalne opóźnienie pakietu. N to jest nasza macierz oraz g naszym grafem:

```
def reliability(g, N, p, T_max):

    graph = nx.Graph(g)
    edges = list(graph.edges())

    for e_start, e_end in edges:
        if rand.random() > p:
            graph.remove_edge(e_start, e_end)

    if not nx.is_connected(graph):
        return False

    a_function(graph, N)
    for e in nx.edges(graph):
        if graph.get_edge_data(*e).get('a') >= graph.get_edge_data
    (*e).get('c'):
            return False
    else:
        return delay(graph, N) < T_max
```

Testowanie

Niezawodności sieci

Zobaczmy jakie wyniki dostaniemy po oszacowaniu niezawodności sieci:

```
P = 0.95
def test_reliability():
    counter = 0
    for _ in range(1000):
        if reliability(g, m, P, 0.005):
            counter += 1
        else:
            continue
    print("Reliable is ", (counter/1000)*100

for _ in range(10):
    test_reliability()
```

Wyniki:

```
Reliablitty is  99.6
Reliablitty is  99.8
Reliablitty is  99.8
Reliablitty is  99.6
Reliablitty is  99.5
Reliablitty is  99.6
Reliablitty is  100.0
Reliablitty is  99.8
Reliablitty is  99.7
Reliablitty is  99.8
```

Stopniowe zwiększanie wartości w macierzy natężeń N

Na początku zdefiniujmy funkcję, która stopniowo zwiększa naszą macierz:

```
def increase_value(matrix, size):
    for i in range(size):
        for j in range(size):
            if i != j:
                matrix[i][j] += 4
    return matrix
```

Teraz zbadajmy jak zmieniła się niezawodność sieci:

```
for _ in range(10)
    increase_matrix(20)
    test_reliability()
```

Wyniki:

```
Reliablitty is  99.7
Reliablitty is  97.3
Reliablitty is  86.2
Reliablitty is  43.8
Reliablitty is  0.0
Reliablitty is  0.0
Reliablitty is  0.0
```

```
Reliablitty is  0.0
Reliablitty is  0.0
Reliablitty is  0.0
```

Jak widać z wyników, im większa wartość natężeń tym mniejsza niezawodność sieci. Działa w taki sposób zatem, że po pewnej liczbie kroków funkcja $a(e)$ jest zawsze będzie większa od funkcji $c(e)$.

Stopniowe zwiększanie przepustowości

Najpierw zdefiniujmy funkcję, która będzie zwiększała wartość przepustowości:

```
% C = 2048
def increase_capacity(size):
    global C
    C += size
```

Teraz zbadajmy jak zmieniła się niezawodność:

```
for _ in range(10):
    increase_capacity(200)
    test_reliability()
```

Wyniki:

```
Reliablitty is  99.7
Reliablitty is  97.3
Reliablitty is  100.0
Reliablitty is  100.0
Reliablitty is  100.0
Reliablitty is  100.0
Reliablitty is  100.0
Reliablitty is  100.0
Reliablitty is  100.0
Reliablitty is  100.0
```

Widzimy, że wartość funkcji $a(e)$ powoduje zmniejszanie wartości funkcji T, zatem zwiększania niezawodności.

Stopniowe zwiększanie ilości krawędzi

Najpierw zdefiniujmy funkcje, które będą dodawać do naszego grafu nowe krawędzie oraz sprawdzać, czy krawędź jest przeciążona po dodawaniu nowej krawędzi:

```
def add_edge(graph, matrix):

    e_start, e_end = rand.randint(0, NODES - 1), rand.randint(0,
    NODES - 1)
    while graph.has_edge(e_start, e_end):
        e_start, e_end = rand.randint(0, NODES - 1), rand.randint
    (0, NODES - 1)

    graph.add_edge(e_start, e_end, c=C, a=0)
    if is_overloaded(graph, matrix):
        return add_edge(graph, matrix)
    return graph
```

```
def is_overloaded(graph, matrix):
    tmp_graph = nx.Graph(graph)
    a_function(tmp_graph, matrix)
    for e in tmp_graph.edges():
        if tmp_graph.get_edge_data(*e).get('c') <= tmp_graph.
    get_edge_data(*e).get('a'):
            return True
    return False
```

Teraz zbadajmy jak zmieniła się niezawodność:

```
for _ in range(10):
    add_edge(g, m)
    test_reliability()
```

Wyniki:

```
Reliablitty is  99.8
Reliablitty is  99.8
Reliablitty is  100.0
Reliablitty is  99.8
Reliablitty is  99.9
Reliablitty is  99.3
Reliablitty is  99.8
Reliablitty is  99.9
Reliablitty is  99.9
Reliablitty is  99.9
```

Jak widać, im większa liczba krawędzi tym większa wartość niezawodności sieci.

Podsumowanie

Architektura sieci powinna być dobrana odpowiednio do potrzeb.Najlepsze grafy do budowy modeli sieci komputerowych to grafy regularne.