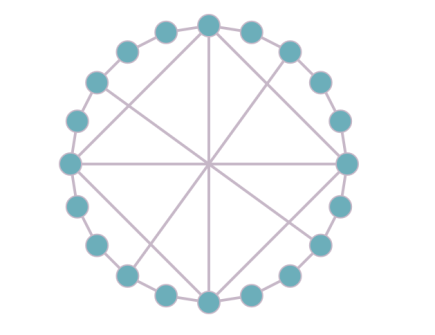
Laboratoria z Technologii Sieciowych

Damian Kokot

1. **Zadanie 1**
   1. **Tworzenie modelu sieci**

W tym zadaniu chciałbym spróbować stworzyć model jak najbardziej symetrycznej sieci i go zbadać pod kątem przepustowości. Stworzę do tego funkcję w języku *Python*.

W pliku GraphGenerator.py umieściłem dwie poniższe funckje do generowania grafu:

def generateGraph(size):

   graph = nx.cycle\_graph(size)

   # Dodawanie krawędzi sąsiedzkich

   for index in range(size - 1):

      graph.add\_edge(index, index + 1)

   # Definiowanie krawędzi przekątnych

   diagonals = [(source, source + 10) for source in [0, 2, 5, 7]]

   diagonals += [(source, (source + 5) % size) for source in [0, 12, 17, 2, 6, 5, 10, 15]]

   for (source, target) in diagonals:

      graph.add\_edge(source, target)

   nx.set\_edge\_attributes(graph, 0.0, 'a')

   return graph

Następnie wyznaczę macierz N losując wartości z pewnego przedziału.

def generateIntensityMatrix(size):

   # Generowanie najkrótszych ścieżek

   N = np.zeros((size, size), dtype=int)

   for i in range(size):

      for j in range(size):

         if i != j:

            N[i][j] += random.randrange(30, 50)

   return N

1. **Zadanie 2**
   1. **Mierzenie niezawodności sieci**

W pliku Tester.py umieściłem poniższe funkcje:

def averageWaitTime(graph, m):

   totalTime = 0

   sumOfIntensity = 0

   for edge in graph.edges:

      edgeData = graph.get\_edge\_data(\*edge)

      sumOfIntensity += edgeData['a']

      totalTime += edgeData['a']/( edgeData['c']/m - edgeData['a'])

   return totalTime / sumOfIntensity

Funkcję która pobiera informację o przepustowości (a(e) i c(e)) i oblicza średni czas oczekiwania zgodnie ze wzorem podanym na liście:

def testModel(graph, intensityMatrix, edgeSpeed, averageDataSize, Tmax, p, attempts):

passedAttempts = 0

delaysTotal = 0

timeoutsCount = 0

tooBigDataCount = 0

for \_ in range(attempts):

updatedGraph = modifyMainGraphModel(graph, p)

if not nx.is\_connected(updatedGraph):

continue

nx.set\_edge\_attributes(updatedGraph, edgeSpeed, 'c')

if not updateAOnPaths(updatedGraph, averageDataSize, intensityMatrix):

tooBigDataCount += 1

continue

waitTime = averageWaitTime(updatedGraph, averageDataSize)

if waitTime < Tmax:

passedAttempts += 1

delaysTotal += waitTime

else:

timeoutsCount += 1

return {

'reliability': getReliability(passedAttempts, attempts),

'timeouts': getTimeoutPercentage(timeoutsCount, attempts - passedAttempts),

'delay': getAverageDelay(delaysTotal, passedAttempts)

}

Funkcję która testuje model sieci usuwając niektóre z krawędzi w grafie, po czym sprawdza czy graf nie jest rozspójniony, czy na każdej krawędzi nie zachodzi a(e) \* m > c(e), po czym mierzy czas i sprawdza czy nie przekroczyliśmy Tmax.

Są też funkcje do tworzenia nowego, przetworzonego grafu

'''

Graph modification functions

'''

def filterRandomEdges(edgesList, p):

return list(filter(lambda edge: random.randrange(1000) <= p, edgesList))

def modifyMainGraphModel(graph, p):

newGraph = nx.Graph()

newGraph.add\_nodes\_from(graph)

newGraph.add\_edges\_from(filterRandomEdges(graph.edges, p))

newGraph.graph = graph.graph

return newGraph

Oraz funkcję, odpowiedzialną za aktualizowanie a(e) w zależności od obecnego.

'''

Updating edge attributes on path, to update weights on edges in graph

'''

def updateAOnPaths(graph, averageDataSize, intensityMatrix):

nx.set\_edge\_attributes(graph, 0.0, 'a')

for source, row in enumerate(intensityMatrix):

for target, weight in enumerate(row):

if source == target:

continue

path = nx.shortest\_path(graph, source, target)

# Adding weight on path

for nodeIndex in range(len(path) - 1):

graph[path[nodeIndex]][path[nodeIndex + 1]]['a'] += weight

node = graph[path[nodeIndex]][path[nodeIndex + 1]]

if node['a'] \* averageDataSize > node['c']:

return False

return True

1. **Obserwacje**

Obserwacji dokonywano przy ustaleniu p=90% i Tmax= 0,00005, prędkości=3Mb/s

* 1. **Zwiększanie macierzy natężeń**Aby zrealizować to zadanie należy stopniowo zwiększać ilość pakietów w macierzy N.

|  |  |
| --- | --- |
| **range(x, y)** | **Output** |
| x=0, y=30 | Reliability: 87.90% and average time passed: 0.000034  where in 0.00% cases there was connection timeout |
| x=30, y=60 | Reliability: 86.30% and average time passed: 0.000035  where in 0.00% cases there was connection timeout |
| x=60, y=90 | Reliability: 86.80% and average time passed: 0.000037  where in 0.00% cases there was connection timeout |
| x=90, y=120 | Reliability: 86.60% and average time passed: 0.000039  where in 1.39% cases there was connection timeout |
| x=120, y=150 | Reliability: 85.60% and average time passed: 0.000041  where in 3.35% cases there was connection timeout |
| x=0, y=500 | Reliability: 31.80% and average time passed: 0.000048  where in 78.01% cases there was connection timeout |

Przy wyborze Tmax kierowałem się średnim czasem, który ustaliłem na 0,0005. Jak widzimy, przy zwiększaniu macierzy ilość przypadków otrzymania „Timeouta” zwiększała się. Oczywiście zmniejszyła się również niezawodność sieci ze względu na ilość połączeń. Po ostatnim teście możemy wywnioskować, że mając duże obciążenie bardzo łatwo doprowadzić do „connection timeout” mając duże obciążenie sieci.

* 1. **Zwiększanie przepustowości łącza**Tu postanowiłem zmieniać tylko funkcję c(e). Niech „s” oznacza prędkość łącza. Zaznaczmy, że wielkość pakietu to 10b

|  |  |
| --- | --- |
| **Prędkość** | **Output** |
| s=2 | Reliability: 0.00% and average time passed: 0.000000  where in 84.90% cases there was connection timeout |
| s=2,5 | Reliability: 60.90% and average time passed: 0.000045  where in 31.20% cases there was connection timeout |
| s=3 | Reliability: 87.20% and average time passed: 0.000035  where in 0.00% cases there was connection timeout |

Nie zmieniając Tmax możemy zauważyć, że prędkości maleją wraz ze zwiększeniem przepustowości. Kolejną rzeczą jest fakt, że skoro ilość przypadków rozspójnienia sieci jest praktycznie stała, to ilość „timeoutów” spada, a co za tym idzie wzrasta niezawodność

* 1. **Zwiększanie ilości połączeń w sieci**W tym zadaniu postanowiłem dodać krawędzie do grafu, tak, aby najlepiej ulepszyć graf.  
     Dla lepszej jakości wyników, ustaliłem prędkość na 5Mb/s Program został uruchomiony dla poszczególnych wartości p:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **p=0,7** | **p=0,8** | **p=0,9** | **p=0,95** |
|  | 0.60% | 7.40% | 39.50% | 76.20% |
|  | 33.20% | 65.70% | 88.70% | 97.70% |
|  | 27.40% | 60.90% | 87.40% | 97.00% |
|  | 30.70% | 58.20% | 86.20% | 96.80% |

Dodane krawędzie mają znaczenie dla poprawności działania sieci. Zauważmy, że im bardziej uzależniamy jeden wierzchołek od całości tym lepiej dla naszego modelu, bo trudniej jest rozspójnić graf sieci. Ciekawym faktem jest, że poprzez dodanie krawędzi nie łączącym kluczowy punkt jak w teście 1 a jak w teście 2 uzyskujemy większą spójność.

Wniosek nasuwa się taki, że mając wierzchołek st. 2 prawdopodobieństwo wyeliminowania go z grafu jest równe , a dla wierzchołka st. 3 prawdopodobieństwo wyeliminowania go to , a zatem nieco mniejsze.

1. **Wnioski**

Budowanie modelu sieci to nie prosta sprawa. Na poprawność działania może wpływać wiele czynników. Jednak bardzo ciekawym doświadczeniem jest wyciągnięcie wniosków jak ważne jest uniezależnianie całości od jednego punktu, tzn. gdyby wszystkie pakiety z Europy miałyby wędrować przez jeden punkt np. w Londynie, a wszystkie pakiety w Ameryce przez np. Waszyngton, to w przypadku awarii jednego z ośrodków albo połączenia, moglibyśmy stracić całkowicie łączność z Stanami Zjednoczonymi.