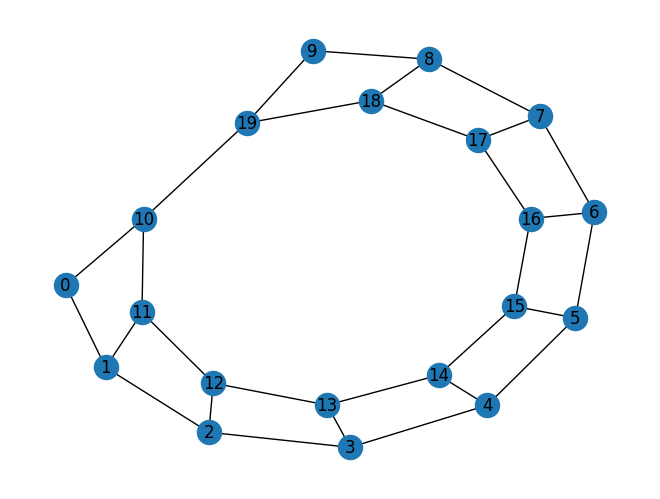
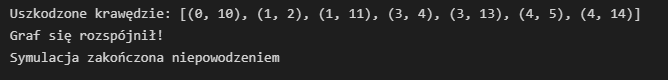
Laboratorium 2

Łukasz Machnik nr 268456

# Eksperymenty:

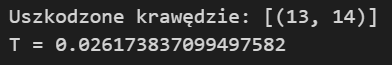
Przyjęta przeze mnie topologia sieci przedstawiona jest na poniższej grafice: Jak widać moja sieć ma 20 wierzchołków i 29 krawędzi - większość wierzchołków jest połączona z 3 innymi. Macierz natężenia wygenerowałem przypisując każdej parze wierzchołek źródłowy – wierzchołek docelowy liczbę pakietów (a(e)) równą ich sumie modulo 6. Założyłem przy tym również że ruch sieciowy od danego wierzchołka do samego siebie wynosi 0. Przepustowość każdej krawędzi (c(e)) jest ustawiona na 512 kb czyli 524288 b. Założyłem również średni rozmiar pakietu (m) równy 1024 b. Przy takich założeniach dla każdej krawędzi realny przepływ jest mniejszy niż przepustowość (c(e) > a(e) \* m).

Jest to topologia którą stosunkowo trudno rozspójnić – potrzeba co najmniej awarii 3 krawędzi naraz i musiałyby to być bardzo konkretne krawędzie – przy dobrej niezawodności połączeń szansa na to jest niewielka. Ja ustaliłem niezawodność krawędzi na poziomie p = 0.99 ale nawet jeśli niezawodność krawędzi była równa 0.75 to graf przestawał być spójny tylko czasami – kiedy zostało zerwanych aż 7 krawędzi naraz. 

Obraz zawierający tekst

Opis wygenerowany automatycznieUstaliłem wartość T\_max na 0.025. W podstawowych warunkach stosunek „dostępnej” przepustowości do tej „zajętej” był jednak dużo lepszy:   
Nawet po uszkodzeniu krawędzi stosunek ten pozostawał bardzo dobry: Obraz zawierający tekst

Opis wygenerowany automatycznie  
Prawdopodobieństwo przekroczenia wartości T\_max jest tak naprawdę bliskie 0 co potwierdziły dalsze próby w których ani razu nie przekroczyłem tej wartości.

Dopiero kiedy stopniowo zacząłem zwiększać wartości w macierzy przepływu (wszystkie wartości zwiększając początkowo o 1, potem o 2, itd.) okazało się że dopiero po zwiększeniu wszystkich wartości o 2 raz na jakiś czas przekraczałem założoną wartość T\_max:   
  
Jednak nadal prawdopodobieństwo tego nie jest wysokie: Dopiero po dodaniu do każdej trasy 3 dodatkowych pakietów ilość prób w których przekroczono T\_max znacznie wzrosła:  jednak wtedy pojawił się też spory odsetek prób w których przekroczono maksymalną przepustowość krawędzi.

Po powróceniu do bazowej konfiguracji sprawdziłem z większą dokładnością jaka jest częstotliwość sukcesu: tak jak pisałem już wcześniej jest to prawdopodobieństwo bardzo wysokie bliskie 100%. Po dodaniu jednej krawędzi (0, 9) wzrosło ono jeszcze bardziej:   
  
  
  
Po dodaniu drugiej krawędzi (10, 13) już wszystkie próby kończą się sukcesem:

# Kod:

Powyższe eksperymenty wykonałem korzystając z programu napisanego w Pythonie używając Jupyter Node’a oraz biblioteki NetworkX. Kod podzieliłem na segmenty które tutaj pokrótce opiszę. Pierwszy segment to importowanie potrzebnych bibliotek, następny to ustawienie kilku parametrów początkowych w tym macierzy natężeń:

N = []      # Macierz natężeń

G = 0       # Łączna liczba pakietów przesyłanych przez sieć

m = 1024    # Średni rozmiar pakietu [b]

S = nx.Graph() # Graf sieci

for i in range(20):

    N.append([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

    for j in range(20):

        if(i != j):

            N[i][j] = (i + j) % 6

            # N[i][j] = (i + j) % 6 + 3

            G += N[i][j]

Generuję tę macierz iteracyjnie, jak widać w powyższej pętli.

def resetGraph():

    for i in range(20):

        S.add\_node(i, destinations=N[i])

    for i in range(19):

        n = ((i + 1) % 20)

        p = ((i + 10) % 20)

        S.add\_edges\_from([(i, n), (i, p)], c=524288, a=0, p=0.99) # c = 512 kb

    S.remove\_edge(9, 10)

    S.add\_edge(19, 10, c=524288, a=0, p=0.99)

W następnym segmencie definiuję funkcję która ustawia graf do wartości początkowej, od razu ją też wywołuję i wypisuję liczbę krawędzi żeby upewnić się że jest ona zgodna z poleceniem, następnie rysuję graf:

resetGraph()

print(S.number\_of\_edges())

nx.draw(S, with\_labels=True)

# nx.draw\_circular(S, with\_labels=True)

Definiuję funkcję która przeprowadza jedną sekundę symulacji. Losuje czy i które krawędzie zostały uszkodzone, usuwa je a następnie dla każdej pożądanej ścieżki zapamiętuje ile pakietów próbuje się przesłać używając danej krawędzi. Jeżeli po usunięciu krawędzi graf się rozspójnił to bez kontynuowania testów zwraca kod błędu = 1. Jeżeli przepustowość którejkolwiek krawędzi zostanie przekroczona to również zwraca kod błędu = 2. W celu przeprowadzenia niektórych eksperymentów odkomentowywałem poszczególne linijki wypisujące mi wyniki symulacji. Symulacja zakończona sukcesem zwraca 0:

def simulate():

    toDelete = []

    for e in list(S.edges()):

        edge = S[e[0]][e[1]]

        edge['a'] = 0

        if (random.random() > edge['p']):

            toDelete.append((e[0], e[1]))

    if len(toDelete) > 0:

        # print("Uszkodzone krawędzie: " + str(toDelete))

        S.remove\_edges\_from(toDelete)

    # else:

        # print("Brak uszkodzonych krawędzi")

    if nx.is\_connected(S) == False:

        # print("Graf się rozspójnił!")

        return 1

    for i in range(20):

        nodeDest = S.nodes()[i]['destinations']

        for j in range(20):

            packageLoad = int(nodeDest[j])

            if packageLoad > 0:

                path = nx.shortest\_path(S, i, j)

                prevNode = path.pop(0)

                for nextNode in path:

                    S[prevNode][nextNode]['a'] += packageLoad

                    if S[prevNode][nextNode]['a'] \* m > S[prevNode][nextNode]['c']:

                        # print("Przekroczono dostępną przepustowość na krawędzi " + str(e[0]) + " <-> " + str(e[1]))

                        return 2

                    prevNode = nextNode

    return 0

W kolejnym segmencie ustawiam liczbę prób do obliczenia średniej. W niektórych podpunktach zadowalałem się pojedynczą próbą ale w ostatnich podpunktach gdzie jest pytanie o prawdopodobieństwo ustawiłem tę wartość jak widać poniżej na 10000. Następnie obliczam wartość T\_avg tak jak było to podane w poleceniu. Tu także w zależności od podpunktu odkomentowywałem poszczególne linijki wypisujące mi rozwiązanie:

graphNotConnectedCount = 0

capacityOverflowCount = 0

T\_maxOverflow = 0

successCount = 0

numberOfTries = 10000

for a in range(numberOfTries):

    resetGraph()

    # S.add\_edge(0, 9, c=524288, a=0, p=0.99)       # Krawędzi dodane w ramach ostatniego podpunktu

    # S.add\_edge(10, 13, c=524288, a=0, p=0.99)

    ret = simulate()

    if ret == 0:

        T\_avg = 0

        SUM\_e = 0

        for e in list(S.edges()):

            edge = S[e[0]][e[1]]

            dividor = (edge['c'] / m - edge['a'])

            if dividor != 0:

                SUM\_e += edge['a'] / dividor

            else:

                SUM\_e += edge['a'] / 0.000001

        T\_avg = SUM\_e / G

        # print("T = " + str(T\_avg))

        # print("Symulacja zakończona powodzeniem")

        if T\_avg > 0.025:

            T\_maxOverflow +=1

        else:

            successCount += 1

    elif ret == 1:

        graphNotConnectedCount += 1

    elif ret == 2:

        capacityOverflowCount += 1

    # else:

    #     print("Symulacja zakończona niepowodzeniem")

print("Prawdopodobieństwo sukcesu: " + str(successCount / numberOfTries \* 100) + "%")

# Wnioski:

Odpowiednia topologia zapewniająca dużą redundancję oraz wysoka przepustowość i niezawodność połączeń zapewniają bezproblemowe działanie sieci. Na „chłopski rozum” najbardziej bezawaryjna sieć to taka w której każdy węzeł jest połączony z każdym innym jednak dla wielu węzłów jest to niewykonalne w związku z dużymi kosztami takiego rozwiązania. Jak pokazał mój eksperyment w wypadku 20 węzłów w zupełności wystarczy żeby każdy z nich był połączony tylko z 3 sąsiadami i to już zapewnia stosunkowo bezawaryjne działanie sieci.