Problem Chosen	2021 MCM/ICM Summary Sheet	Team Control Number
A		55280

The Great Virtues of Our Great Grand Giant Supervisor QYH

Summary

This is the **SUMMARY**

Keywords: keywords1,keywords2,keywords3

The Great Virtues of Our Great Grand Giant Supervisor QYH

February 3, 2021

Contents

1 Introduction

1.1 Problem Background

Advancing towards a booming flow of information currents, the world is requiring more and more efficient broadcasting techniques to keep up with its own pace. Meanwhile, the number of Internet users is still under a sharp grow, making the topology of the whole system even more sophisticated. As a result, the urge to found a comprehensive optimum in the transmission system is added to schedule.

Previous broadcasting involved the technique of unicast (or singlecast), by which only one pack of information is sent to one specific user at one time. Although its specific destination minimizes the capacity occupied by headers, this method would lead to great redundancy if the same pack needs to be sent to multiple users along the same path. We must appeal to a technique with greater efficiency in face with the rising transmission demand. Therefore, we need to find a balance between the capacity loss caused by headers and the redundancy of packs.

1.2 Our work

This paper aims to find a comprehensive optimum of a transmitting system. By investigating the combination of singlecasting and multicasting, we gradually gain insights into and are able to develop an advanced

2 Assumptions and Definitions

2.1 Our Assumptions

Def. a node u's distance from another node v: the length of the minimum path from u to v.

Assumptions:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

1) The topology is relatively balanced, namely, the majority of nodes [sharing the same minimum length of path from the source to themselves] in the original graph have approximate number of neighbors. [sharing the same distance from the source]

2) The complexity of the network is in proportion to the scale of the network, i.e. the number of nodes with [larger length of minimum path from the source to themselves] are generally larger than the number of those with smaller length. [larger distance from the source]

3) The network is an undirected and connected graph, i.e. if information can be transferred from router A to router B, it can be transferred backwards. We ditch those nodes which are disconnected to the source.

4) The effect of redundancy is of linear growth rather than of exponential growth, i.e. the loss function brought by redundancy is in proportion to the number of redundant edges, but not in exponential relation to the number of repeatedly utilized edges.

5) The effectiveness in the network is only related to the length of path a data packet covered and has nothing to do with the number of packets or the computing power of routers where packets are transferred.

6) The network uses IP addresses to recognize different routers. We assume the global percentage of IPv6 is approximate to the percentage of IPv6 addresses visiting Google according to the statistical data given by Google, i.e. 32

2.2 Variable Definitions

Symbol	Definition
f_1	a function describing redundancy
f_2	a better function
f_s	
S_i	
U	
I_A	
I_H	
I_G	
N_i	
m	
i	
N	
R	

3 Topologies with Certain Packet Capacity – Task 1

3.1 Calculating the maximum number of receiving points

We study possible paths originating from node A. Since duplicate payloads are not allowed to appear, the data can only access node B C D once, respectively. Then we reach the next level of the question.

Given that the maximum number of branches on the header is 3, we know that the pack can get through at most 2 edges after reaching the second point. That means at most two other points can be reached.

However, there is only one edge (D-H) after node D, indicating that only one other point is available. As to node B and node C, the upper limit can both be reached.

Therefore, the number of receiving points cannot exceed 9. In detail, 3 for B and two other points, 3 for C and two other points, 2 for D and one other point and 1 for the source point A.

We can easily find a none-redundant distribution with 9 receiving points. The example is as follows:

(1). A-B-E-F

(2). A-C-F-J

(3). A-D-H

[ho]

good bye space good

$$goodbye \tag{1}$$

Appendices

Appendix A Code Example

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import networkx as nx
import numpy as np
import seaborn as sns

source = 0
eps = 0.01
SUP = 10.0
infoall = 327
Sh = 1.0

f = open("./In.txt")
s = f.readline()
s = f.readline()
s = f.readline()
p1 = s.find(':')+2
p = s.find('Edges')
node = int(s[p1:p])
print(node)
p = p+7
edge = int(s[p:])
print(edge)
s = f.readline()


G = nx.Graph()
G.add_nodes_from(range(0,node+1))
for i in range(0,edge):
    s = f.readline()
    p = s.find('\t')
    x = int(s[:p])
    y = int(s[(p+1):])
    G.add_edge(x,y)

"""
H = nx.Graph()
H.add_nodes_from(range(0,node+1))
for i in range(0,node):
    if not nx.has_path(G,0,i): continue
    path = nx.shortest_path(G,0,i)
    j = path[0]
    for j in range(1,len(path)):
        H.add_edge(path[j-1],path[j])
"""
H = G

Hqu = [source]
Hds = [13]*(node+1)
Hds[source] = 0.5
Hmk = [0]*(node+1)
max = 0
Nds = [1]
Nsn = [float(len(H[source]))]      #the average number of its sons
head = 0
tail = 0

while head<=tail:
    ngh = H.neighbors(Hqu[head])
    for i in ngh:
        if (i>node): break
        if (Hmk[i]==0):
            tail+=1
            Hqu = Hqu + [i]
            Hds[i] = Hds[Hqu[head]]+1
```

```
            if Hds[i]-0.5>max:
                for j in range(max,int(Hds[i]-0.5)):
                    Nds = Nds + [0]
                    Nsn = Nsn + [0.0]
                max = int(Hds[i]-0.5)
                Nds[max] = 1
                Nsn[max] += len(H[i])
            else:
                Nds[int(Hds[i]-0.5)]+=1
                Nsn[int(Hds[i]-0.5)]+=len(H[i])
            Hmk[i] = 1
    head+=1

maxs = 0.0
for i in range(0,max+1):
    Nsn[i] = Nsn[i]/float(Nds[i])
    if maxs < Nsn[i]: maxs = Nsn[i]

def KSH(x):
    sns.set_style('white')

    fig, ax = plt.subplots(figsize=(3,2.5))
    # seaborndistplot
    sns.set_palette("hls")  # hls
    sns.distplot(x, color="r", bins=np.arange(0,max+1,1) ,kde=False)
    ax.set_title('Node Distribution By Distance')

    ax2 = ax.twinx()
    sns.kdeplot(x, bw=.75, linewidth = 5, alpha = 0.5)

    plt.savefig("hist.png", dpi=800)
    plt.show()

KSH(Hds)

print(Nds)

maxn = 0
for i in range(0,max):
    if Nds[maxn]<Nds[i]:  maxn=i;
"""
tmp = Hds
q=max

for i in range(max,maxn,-1):
    print(Nds[i-1]," ",Nds[i],end=" ")
    if (Nds[i]<(Nds[i-1])):
        for j in range(0,node+1):
            if abs(tmp[j]-i-0.5)<1e-6:
                tmp[j] = i-1
                Nds[i-1]+=1
        Nds[i] = 0
        if (q==max): q = i-1
        print("1",end="")
    print()

print(Nds)
KSH(tmp)

q=max
for i in range(max,maxn,-1):
    if Nds[i]/float(Nds[i-1])<eps:
        Nds[i-1] = Nds[i-1]+Nds[i]
        Nds[i] = 0
        if (q==max): q = i-1

rate = [0.0]*(max+1)
for i in range(q,0,-1):
    if Nds[i-1]>0: rate[i] = Nds[i]/float(Nds[i-1])
for i in range(1,q+1,1):
    print('%.1f'%rate[i],end=" ")
"""

"""
def vsimple(x,i,n,C):
    b = x[:]
    for ii in range(len(x)-1):
        b[ii] = b[ii+1]/b[ii]
    b = b[:-1]
    def D(b_list,i_in_b):
        mul=1
        sum=0
        for var in b_list[i_in_b:]:
            mul *= var
            sum += mul
        return sum
    return x[i]*(i+n*D(b,i))/(n*(C-n*D(b,i)))
"""

def vsimple(x,i,n,C):
    b = x[:]
    for ii in range(len(x)-1):
        b[ii] = b[ii+1]/b[ii]
    b = b[:-1]

    mul=1
```

```
    sum=1
    for var in b[i:]:
        mul *= var
        sum += mul
    packcost=i-1+n*sum
    return x[i]*(packcost)/(n*(C-packcost))

def v1(x,i,n,C):
    sm=-1
    for var in x:
        sm+=var
    return vsimple(x,i,n,C)*C/sm


maxs=10.0

x = np.arange(0,maxs,eps,float)
for j in range(0,len(x)-1):
    x[j] = (x[j]+x[j+1])/2.0
col = len(x)
col-=1
data = np.empty([maxn,col], dtype = float, order = 'C')
#print(x)

data[0] = [SUP]*col
for i in range(0,maxn):
    for j in range(0,col):
        if x[j]>Nsn[i-1]:
            j = j-1
            continue
        data[i][j] = v1(Nds,i,x[j],infoall)
        if data[i][j]<Sh: data[i][j] = SUP
        if data[i][j]>SUP: data[i][j]= SUP
    for k in range(j+1,col):
        data[i][k] = SUP

for i in range(1,maxn):
    minall = SUP
    q = 0
    #    print(i,': ',end='')
    for j in range(0,col):
 #       print('%.2f'%data[i][j],end=' ')
        if data[i][j]<minall:
            minall = data[i][j]
            q = x[j]
    print(minall,' ',q)
#   print()


fig, ax = plt.subplots(figsize=(3,2))


sns.set_style("white")

sns.heatmap(data, vmax=1.5, vmin=1.0, cmap = "Blues")
ax.set_xlabel('N / Amount',{'family' : 'Consolas'},rotation=0)
ax.set_ylabel('I / Distance',{'family' : 'Consolas'},rotation=90)
label_y = ax.get_yticklabels()
plt.setp(label_y , rotation = 0)
label_x = ax.get_xticklabels()
plt.setp(label_x , rotation = 60)
'''
from matplotlib.ticker import MultipleLocator, FormatStrFormatter
xmajorLocator   = MultipleLocator(1) #xn
xmajorFormatter = FormatStrFormatter('%1.1f') #x
ax.xaxis.set_major_locator(xmajorLocator)
ax.xaxis.set_major_formatter(xmajorFormatter)
'''

plt.savefig('Heapmap.png', dpi=800)
plt.show()
```