Assignment1

May 20, 2020

1 CMPT 371 - Assignment 1

1.0.1 Trevor Grabham - 301281720

1.0.2 Question 1

a) What is the average percentage of the time each line is used?

Each channel supports up to 2.5MB per second

[1]:
```
maximum_rate = 2.5
```

Each channel is on average only sending 0.5MB per second

[2]:
```
average_rate = 0.5
```

[3]:
```
average_percent = (average_rate/maximum_rate)*100
print('The average percentage of time that each line is in use␣
 ↪is',average_percent,'%')
```

```
The average percentage of time that each line is in use is 20.0 %
```

b) What fraction of the information being sent is overhead?

Each frame sends 45 bytes of header

[4]:
```
header_size = 45
```

Each frame sends over 1200 bytes total

[5]:
```
total_size = 1200
```

[6]:
```
fraction_header = header_size/total_size
print('The fraction of overhead data being sent is', fraction_header)
```

```
The fraction of overhead data being sent is 0.0375
```

c) What is the average input bit rate summed over all signals?

There are 80 channels

[7]:
```
number_channels = 80
```

```
The average input bit rate is 0.5MB per second on each channel
```

[8]:
```python
bit_rate_per_channel = 0.5
```

[9]:
```python
average_bit_rate_total = number_channels * bit_rate_per_channel
print('The average bit rate over all of the channels␣
 ↪is',average_bit_rate_total,'MB/s')
```

```
The average bit rate over all of the channels is 40.0 MB/s
```

d) What is the average data transmission rate summed over all signals? (data only)

```
We know that the data makes up 1155 of the 1200 bytes being send over each frame
```

[10]:
```python
data_size = 1155
total_size = 1200
```

```
We therefore know the fraction of the data sent
```

[11]:
```python
fraction_data = data_size/total_size
print('The fraction of data sent is',fraction_data)
```

```
The fraction of data sent is 0.9625
```

```
We know what the overall bit rate of all of the channels combined is
and we know what percentage of the bits make up the data.
Therefore, we are able to tell what the data transmission rate is over all the signals
```

[14]:
```python
average_data_rate_total = average_bit_rate_total*fraction_data
print('The average data transmission rate summed over all signals␣
 ↪is',average_data_rate_total,'MB/s')
```

```
The average data transmission rate summed over all signals is 38.5 MB/s
```

e) What percentage of the line's capacity will be utilized?

It depends on whether we are talking about the amount utilized for bit transfer of data transfer

```
For either of them, the lines total capacity is the same, 50MB/s
```

[13]:
```python
total_capacity = 50
```

```
For the total bit transfer, we will use the input bit rate over all of the lines.
We will then be able to calculate the average percent of the lines usage capacity
```

[18]:
```python
percent_line_capacity = (average_bit_rate_total/total_capacity)*100
print('The average percent of the lines capacity being utilized for bit␣
 ↪transfer is',percent_line_capacity,'%')
```

```
The average percent of the lines capacity being utilized for bit transfer is
80.0 %
```

However, if we are only concerned with the amount that is being used for the data transfer, we will instead want to use the data transmission rate instead

And we already know what the total data transmission rate is for the entire line. Therefore, we are able to calculate the average percentage of the lines capacity being utilized

[16]:
```
percent_line_capacity = (average_data_rate_total/total_capacity)*100
print('The average percent of the lines capacity being utilized for data␣
↪transfer is',percent_line_capacity,'%')
```

```
The average percent of the lines capacity being utilized for data transfer is
77.0 %
```

1.0.3 Question 2

a) How many users would the line support assuming each user has a 8 Mbps connection and uses synchronous TDM to access the line? Assume that only 95% of the capacity of the line may be used for user connections.

We know that the total capacity of the line is 180 Mbps

[19]:
```
total_capacity = 180
```

We also know that each user needs a 8Mbps connection and that the line is divided using synchr so we cannot have the total bandwidth exceed the lines total capacity

[20]:
```
user_capacity = 8
```

Finally, we know that the line can only use 95% of its total capacity for user connections

[21]:
```
usable_capacity = total_capacity * 0.95
```

To get the number of users possible, we can divide the usable capacity by the capacity needed If this number is not an integer, then we will round down as we cannot have partial users and the usable capacity

[23]:
```
max_users = usable_capacity//user_capacity                         # integer␣
↪division to round down
print('The maximum number of users that can be supported by the line␣
↪is',max_users)
```

```
The maximum number of users that can be supported by the line is 21.0
```

b) Suppose that each user only used their 8 Mbps line 28% of the time, 95% of the capacity of the may be used for user connections and synchronous TDM is being used. How many users can the line support for these assumptions?

Since we are using synchronous TDM, even when the user is not sending any data, the bandwidth reserved for the user cannot be used by other users, so we would still have the same number of supported, 21

c) Suppose that each user only used their 8 Mbps line 28% of the time, 92% of the capacity of the may be used for user connections and statistical TDM is being used. How many users can the line support for these assumptions?

Under statistical TDM, we will need to calculate the average amount of input data that each use

[29]:
```
average_user_input = 8*0.28                     # 28% of the time, using 8Mbps
```

We also need to update the usable capacity of the line, as we are only able to use 92% now

[30]:
```
usable_capacity = total_capacity * 0.92
```

We now want to pick a number of users, such that the average input rate summed over all of the than the usable capacity of the line. If the number is not an integer, then we will round down

[31]:
```
max_users  = usable_capacity//average_user_input
print('The maximum number of users that the line can support under these␣
 ↪assumptions is', max_users)
```

```
The maximum number of users that the line can support under these assumptions is
73.0
```

The number of users is much larger because on average, the total input rate is only 2.24 Mbps, four times smaller than the average input for the synchronous method

d) Suppose there are 80 users, using 3 Mbps connections at the same time. Each user uses their connection 41% of the time. Find the probability that at any given time, exactly 18 users are transmitting simultaneously. (Hint: use the binomial distribution).

Using the binomial distribution, we will have the following varaibles

[37]:
```
p = 0.41
n = 80
k = 18
```

Where p is the probability of each of the users using their line at any given time,
n is the number of users
k is the number of users using the line that we are the same time

The formula for the binomial distribution follows

[50]:
```
def factorial(n):                         # helper function for the binomial␣
 ↪function
    prod = 1
    for i in range(n):
        prod *= (i+1)
    return prod

def binomial(n,p,k):
    n_choose_k = factorial(n)/(factorial(k) * factorial(n-k))
    return (n_choose_k * p**k * (1-p)**(n-k))
```

Therefore, we can now use the binomial distribution to calculate the probability that exactly
using their lines simultaneously

[52]:
```
probability = binomial(n,p,k)
print('The probability that exactly 18 lines are in use simultaneously is',␣
 ↪probability, 'or', probability*100,'%')
```

```
The probability that exactly 18 lines are in use simultaneously is
0.00023627864472088427 or 0.023627864472088426 %
```

e) Assume that the transmission line in the problem is used within a packet switched network. Would you use Synchronous or Statistical TDM? Why?

For a packet switched network, I would probably use Statistical TDM, because of the variable i
switched network, and because we are working within a packet switched network, we also know tha
expected along the line already, so this allows us to maximize our throughput, as we don't have
any delays that are introduced into the system

f) Assume that the transmission line in the problem is used within a connection oriented circuit switched network. Would you use Synchronous or Statistical TDM? Why?

For a connection oriented circuit switched network, I would probably use Synchronous TDM. This
connection oriented circuit switched network, the main advantage is that we will have very lit
were to use Statistical TDM, it would introduce the possibility of a queueing delay into the sy
Synchronous system, we wouldn't have to worry about that. This of course, comes at the cost of
of the system, but with a connection oriented circuit switched network, we were already capped
anyways

1.0.4 Question 3

a) What does DNS stand for?

DNS stands for Domain Name System

b) State four basic uses for DNS?

1. IP address lookup - looking up IP addresses, given the name of a host
2. Host name lookup - looking up the host name, given a particular IP address
3. Name aliasing - allowing multiple host names to be routed to the same IP address. i.e. googl
There is also the special case when we can use the same host name for mail and web servers
4. Load distribution - returning a set of IP addresses for larger web servers with a lot of tra

c) i) After the query for collie.dog.pet.bc.ca. what DNS name server records are stored in the cache (ignore information about root servers)?

ii) Now consider asking for the IP address of siamese.cat.pet.bc.ca. Which DNS server would the first query be sent to? Why?

iii) Now consider asking for the IP address of siamese.cat.pet.ca. When making a query to .pet.bc.ca. what would be the DNS record being requested in the query?

iv) Draw an annotated diagram to help you explain how the query for the address of siamese.cat.pet.bc.ca. is executed. Show the Resolver, the local DNS server and all DNS servers queried in your diagram. Show all information travelling between the resolver and the DNS servers in your diagram. Be sure to indicate on your diagram what information is requested and returned in each query. State any assumptions you make. Assume the local DNS server makes iterative queries

1.0.5 Question 4

a) What is the propagation delay?

```
Propogation delay is determined by the propogation speed and the distance betweent the points.
2*10^8 m/s and the distance is 5000m
```

[53]:
```
speed = 2*10**8
distance = 5000
```

```
We can now calculate the propogation delay
```

[55]:
```
propogation_delay = distance/speed
print('The propogation delay will be', propogation_delay*10**6,'us')
```

```
The propogation delay will be 25.0 us
```

b) What is the maximum number of bits that have left A before the first bit arrives at B? This number of bits is called the bandwidth delay product

```
The number of bits that have left A by the time the first bit arrives at B will be equal to th
the connection times the amount of time that it will take the first bit to reach B (the propog
calculate the maximum number of bits as follows
```

[56]:
```
transmission_rate = 24*2**20          # bits per second
bandwidth_delay_product = int(transmission_rate*propogation_delay)       # cast␣
 ↪it to an int to round down
print('The maximum number of bits that have left A before the first bit arrives␣
 ↪at B is', bandwidth_delay_product,'bits')
```

```
The maximum number of bits that have left A before the first bit arrives at B is
629 bits
```

c) We can think of each bit occupying a piece of the link. If the bandwidth delay product tells us that there are 100 bits in the transmission medium then we can think of each bit occupying 1/100 of the length of the transmission media. What is the width of each bit (the length of the transmission medium it fills)?

```
We need to know how many bits are in the medium at any given time (the bandwidth_daley_product
medium. Given that the medium is 5000m long and we have 629 bits in the medium at any given ti
width of each bit as follows
```

[59]:
```
bit_width = distance/bandwidth_delay_product
print('Each bit is theoretically',bit_width,'m long')
```

```
Each bit is theoretically 7.94912559618442 m long
```

d) Consider sending the file through the packet switched network without message segmentation (as a single packet). Assume that each of the intermediate hosts are store and forward nodes. How long does it take to send the file from A to B?

To begin with, we know that the file is 3GB or 3*2^30 bits, the propogation delay on each link
processing delay at each link is 0.003s, and there is no queueing delay. We also know that the
links.

[61]:
```
file_size = 3*2**30
propogation_delay = 0.004
processing_delay = 0.003
number_links = 9
```

We also need to find out the transmission delay. For the case when we do not segment the file,
size be equal to the file size. We also have a transmission rate of 90Mbps. Therefore, we are
transmission delay as follows

[62]:
```
transmission_rate = 90*2**20        # bits per second
transmission_delay = file_size/transmission_rate
```

Finally, to get the overall delay, we realize that at each of the intermediate links as well a
we encounter a propogation delay and transmission delay. We also encounter a processing delay
intermediate links as well as the end host, B. We will have a count of 10 for each delay. We c
total delay as follows

[66]:
```
total_delay = (number_links +␣
 ↪1)*(processing_delay+propogation_delay+transmission_delay)
print('The total amount of time that it will take to send the file from A to B␣
 ↪will be',total_delay,'s, or', int(total_delay//60), 'm', total_delay%60, 's')
```

```
The total amount of time that it will take to send the file from A to B will be
341.4033333333333 s, or 5 m 41.40333333333331 s
```

e) Assume that the file is segmented into packets containing 12000 bits each. Each packet has a header of 200 bits. If a packet is partially full of data the remainder of the data field is filled with zeros before the resulting full size packet is transmitted. How long does it take for the file to be transmitted through the network (assume no queuing delays)

We will have the same file size, propogation delay, processing delay, and number of intermedia
However, we will have a new transmission delay, as we are sending smaller packets, this time o
header that consists of 200 of those bits

[70]:
```
packet_size = 12000
transmission_delay = packet_size/transmission_rate
```

We can now calculate the amount of delay experienced by each packet, using the same reasoning

[85]:
```
packet_delay = (number_links +␣
 ↪1)*(processing_delay+propogation_delay+transmission_delay)
```

However, now we must also take into account, how many packets we have to send into the system

[75]:
```
data_per_packet = 12000-200
number_of_packets = file_size/data_per_packet
if number_of_packets%1 != 0:                # rounds up if not a whole number
    number_of_packets = number_of_packets//1 + 1
```

Finally, we take into account how long it will take to send the first n-1 packets into the syst
time it will take for the nth packet to reach host B. The rate at which we send packets into t
the transmission rate and the size of the packets. Once we get this rate, we then have to mult
packets that we are sending into the system

One assumption made is that the processing delay does not effect the rate at which we can send the file into the system. If we take into account that we cannot start recieving the bits of the next packet until the current packet has finished being processed then we would have to add a term of (number_of_packets-1)*processing_delay onto the n_minus_one_time

[84]:
```
packet_rate = packet_size/transmission_rate
n_minus_one_time = packet_rate * (number_of_packets-1)
total_time = n_minus_one_time + packet_delay
print('Using packets of 12000 bits, it will take', total_time,'s to transmit␣
 ↪the file from A to B')
```

```
Using packets of 12000 bits, it will take 34.7831093343099 s to transmit the
file from A to B
```