Computer Networks Homework 3

Spring 2020

Due: 16 March 2020

1. (10 points) Jain's Fairness Index allows us to calculate how fair the allocation of bandwidth is between multiple flows. Let there be 20 flows entering a router as shown below. What is the fairness index for this set of flows?

889 Kbps	393 Kbps	516 Kbps	723 Kbps
548 Kbps	86 Kbps	906 Kbps	184 Kbps
204 Kbps	520 Kbps	973 Kbps	510 Kbps
921 Kbps	59 Kbps	505 Kbps	705 Kbps
842 Kbps	542 Kbps	770 Kbps	671 Kbps

Solution:

Jain's Fairness Index:

$$f(x_1, x_2, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2} \tag{1}$$

Fairness Index:

$$\frac{(\sum_{i=1}^{20} f)^2}{n \sum_{i=1}^{20} f^2} = \frac{131492089}{161292660} = 0.8152391373544214$$

2. (10 points) Design two distinct and different sets of 5 flows such that each set of flows has a 0.5 fairness index. What conclusion can you draw about fairness?

Solution:

Flow Set 1:

Fairness Index:

[975, 150, 150, 150, 200]

$$\frac{(\sum_{i=1}^{5} f)^2}{n \sum_{i=1}^{5} f^2} = \frac{2640625}{5290625} = 0.4991139988186651 \approx 0.5$$

Flow Set 2:

Fairness Index:

$$[50,\ 600,\ 50,\ 600,\ 50]$$

$$\frac{(\sum_{i=1}^{5} f)^2}{n\sum_{i=1}^{5} f^2} = \frac{1822500}{3637500} = 0.5010309278350515 \approx 0.5$$

Conclusion:

Fairness is not exactly fair in the common sense terminology as you can maintain the same fairness with large variations of flows. In Flow Set 1, we have one flow which a large amount of bandwidth while the other four flows have very small amounts of bandwidth. Then in Flow Set 2 we have two higher flows and 3 lower flows. However, the range difference on these is much smaller than from Flow Set 1. This also allows us to conclude that the fairness index is not easily manipulable from one channel.

3. (5 points) When presenting a Random Early Drop (RED) problem, some textbooks will only tell the student that the average length of the queue is halfway between MaxThreshold and MinThreshold. No numeric values will be provided for any of these numbers. Show how knowing this average queue length and a numeric value for MaxProb is enough to calculate a value for $\bar{p}$.

Solution:

RED Algorithm for $\bar{p}$:

$$\bar{p} = \text{MaxProb} \cdot \frac{\text{AvgLength - MinThreshold}}{\text{MaxThreshold} - \text{Min Threshold}}$$

If we are given the $MaxThreshold$ and $MinThreshold$ we may estimate the average queue Length ($AvgLength$) to be halfway between the $MinThreshold$ an $MaxThreshold$ which results in the following:

$$AvgLength = \frac{MaxThreshold + MinThreshold}{2} \tag{2}$$

With this and the given $MaxProb$ we can easily use the above formula to find $\bar{p}$.

4. Using your answer above, let there be a router implementing RED with MaxProb = 0.02.
   (a) (5 points) Calculate the drop probabilities (p) for Count = 1 and Count = 100.
   (b) (5 points) Calculate the probability that packets are **not dropped** for Count = 1 and Count = 100.
   (c) (5 points) Calculate the probability that none of the first 50 packets are dropped. *Hint: the probability that the first 2 packets are not dropped is the probability that the first packet (Count = 1) is not dropped times the probability that the second packet (Count = 2) is not dropped.*

Solution:

Simplification:

$$\bar{p} = \text{MaxProb} \cdot \frac{\text{AvgLength - MinThreshold}}{\text{MaxThreshold} - \text{Min Threshold}}$$

$$=$$

$$\bar{p} = \text{MaxProb} \cdot \frac{(\frac{MaxThreshold+MinThreshold}{2}) \text{ - MinThreshold}}{\text{MaxThreshold} - \text{Min Threshold}}$$

$$=$$

$$-\frac{1}{2}$$

Calculating p:

```
max_prob = 0.02
p_bar= max_prob * -0.5
for count in range(1,101):
    val = p_bar/(1 - count * (p_bar))
    print(f"Count[{count}]: {val}")
```

(a) Using the above formula the drop probably when Count $= 1$ is -0.01

(b) Using the above formula the drop probably when Count $= 100$ is -0.005

(c) Script:

```
total_prob=0
for count in range(1,51):
    val = p_bar/(1 - count * (p_bar))
    if count == 1:
        total_prob=val
    else:
        total_prob = total_prob * val
    print(f"Count[{count}]: {val}")
    print(f"Current probability: {total_prob}")
print("Final probability of not dropping: {}".format(1.0 - total_prob))
```

This results in the final probability of dropping to be 1.6334665437398676^{105}. When we calculate the probability of it dropping it is $1 - 1.6334665437398676^{-105}$, which becomes:

$$1 - 1.6334665437398676^{-105} \approx 1 \tag{3}$$

5. (10 points) A TCP host is transmitting over a network with an MSS of 1460 B. Assuming it is operating in standard AIMD mode (and not slow start), that the initial congestion window is at 11860 B, and that no segments are lost. How many segments need to be successfully ACKed before the window reaches 20000 B?

Hint: you'll likely want to write a program/script to calculate the partial MSS updates.

Solution:

Congestion Window incrementation:

$$\text{Increment} = \text{MSS} * \frac{\text{MSS}}{\text{CongestionWindow}} \tag{4}$$

Used code to calculate packet window:

```
mss = 1460
cong_window = 11860
max_window = 20000
packet=0

while True:
    if cong_window < max_window:
        packet += 1
        increment = mss * (mss / cong_window)
        cong_window += increment
        if packet \% 5 == 0:
            print(f"Packet[{packet}]: Window Size:[{cong_window}]")
    else:
        print(f"Packet[{packet}]: Window Size:[{cong_window}]. Broke
            Max[{max_window}]")
        break
```

Resulting in:

```
                   Packet[5]:  Window Size:[12732.952225877929]
                  Packet[10]:  Window Size:[13549.070983416868]
                  Packet[15]:  Window Size:[14318.228757473114]
                  Packet[20]:  Window Size:[15047.728404857313]
                  Packet[25]:  Window Size:[15743.153093551971]
                  Packet[30]:  Window Size:[16408.885446834778]
                    Packet[35]:  Window Size:[17048.4409129187]      (5)
                  Packet[40]:  Window Size:[17664.690716118843]
                  Packet[45]:  Window Size:[18260.016069667356]
                  Packet[50]:  Window Size:[18836.417915318852]
                  Packet[55]:  Window Size:[19395.596934547557]
                  Packet[60]:  Window Size:[19939.013123167944]
Packet[61]:  Window Size:[20045.91911629973].  Broke Max[20000]
```