Thessalonika Magadia
2 December 2021

User Manual

1. To **compile**, type this into terminal `g++ -o scan scanning.cpp`.
2. Before running, ensure that the text file(s) you need are in the same folder as the `scanning.cpp` file and ensure there are no white spaces in text file. Now to **run** the program type `./scan <textfile.txt> <K>`, where you put the corresponding text file and specified K value. (An example is shown below)

```
g++ -o scan scanning.cpp
./scan customer1.txt 10
```

3. Below is the corresponding **output** of the example.

```
===== root scan results =====
successses      12
collisions      17
idle            6
total           35
efficiency      34.29%
time            350

===== leaf scan results =====
successes       12
collisions      0
idle            1012
total           1024
efficiency      1.17%
time            10240
```

Questions

1. Which of the customers has the most items? How many items do they have?
   Customer 8 has 457 items.

2. When starting at the **leaf level** of the tree, which basket of goods takes the most time to scan? How many time slots does it require?
   All baskets take the same time to scan, 10240ms. Time slots required is 1024.

3. When starting at the **root level** of the tree, which basket of goods takes the **most time** to scan? How many time slots does it require?
   Customer 8's basket takes the most time to scan, 9970ms. Time slots required is 997.

4. When starting at the **root level** of the tree, which basket of goods takes the **least time** to scan? How many time slots are needed?
   Customer 1' basket takes the least time to scan, 350ms. Time slots required is 35.

5. When starting at the **root level** of the tree, which basket of goods generates the **most collisions** during scanning? How many collisions occur?
   Customer 8's basket has the most collisions; 498 collisions occur.

6. When starting at the root level of the tree, which basket of goods generates the **highest proportion of successful slots** (i.e., efficiency) during scanning?
   Customer 8's basket has the best efficiency.

Thessalonika Magadia
2 December 2021

Summary

Case	MAC Protocol	Success	Collision	Idle	Total (time slots)	Efficiency (%)	Time (ms)
Customer1	root	12	17	6	35	34.29	350
	leaf	12	0	1012	1024	1.17	10240
Customer2	root	50	70	21	141	35.46	1410
	leaf	50	0	974	1024	4.88	10240
Customer3	root	132	164	33	329	40.12	3290
	leaf	132	0	892	1024	12.89	10240
Customer4	root	150	191	42	383	39.16	3830
	leaf	150	0	874	1024	14.65	10240
Customer5	root	177	225	49	451	39.25	4510
	leaf	177	0	847	1024	17.29	10240
Customer6	root	329	378	50	757	43.46	7570
	leaf	329	0	695	1024	32.13	10240
Customer7	root	441	484	44	969	45.51	9690
	leaf	441	0	583	1024	43.07	10240
Customer8	root	457	498	42	997	45.84	9970
	leaf	457	0	567	1024	44.63	10240
Customer9	root	433	478	46	957	45.25	9570
	leaf	433	0	591	1024	42.29	10240
Customer10	root	371	421	51	843	44.01	8430
	leaf	371	0	653	1024	36.23	10240

Table 1. MAC Protocol Results from Program
**Time is estimated assuming each time slot is 10ms*

Looking at the results summarized in Table 1, we can observe that scanning from the root gives more consistent results; as the number of items in baskets increase, the efficiency remains relatively the same compared to scanning by the leaf. Generally scanning from the leaf is only beneficial when there are many items in a customer's basket (around half of the total possible unique items is when the efficiency is comparable to scanning from the root). Scanning from the root is more beneficial when there is less items in the basket (about less than half).

When looking comparing collision and idle probes, we can see that root scanning generally has more collisions, while leaf scanning has more idle. For root scanning, collisions and total probes increase with the number of items in the basket, while leaf scanning will always have 0 collisions and 2^K total probes. This is why scanning from the leaf only beneficial when there are a lot of items in the basket: the ratio success/total increases proportionately since total stays constant.