ECS152A write up:

Zuolin Li, 998829351

Hanyuan Dong, 998983129

1. Description(Java):

In this project, we are working on a simulation analysis of token passing protocol based Local-Area-Network (LAN). We assume there are 10 hosts in the ring, one token, one frame and a global even list (GEL).

Event class: arrival event and token event.

Etype: 0 is arrival event, 1 is token event.

Source: arrival: the package from which host. Token: starting token position

Dest: arrival: destination of the package Token:Null

Size: arrival: size of the package Token: Frame Size, all packages size.

Position: position of current token

Free: 1 is free, 0 is not. Only for token.

Steps: track the number of host token has past, used to determine when it is back to original token.

Host class:

ArrayList<Event> queue to store the package arrived.

In the program:

First of all, we initialize the ten hosts, arrange arrival event for each host, and then create a token event, and push them into the GEL. Each host has a queue to store the arrival package. We use a while loop to iterator through the GEL, and stop if the package number reaches 500.

In the loop, we took out the first event in GEL, which is the earliest event happened. If the event type is an arrival event, we add it to the hosts List. And we arrange the next arrival event immediately, and push it to GEL.

If the event type is a token event, we need to check if the token is free. If the token is free, which means we can transmit packages for that host. We need to see if there is package to send in the host. If there is, we remove all the packets in the host to the Frame, and set the next token event. The token event is set not empty, and push to the GEL. If there is not package needs to send, we just pass to next position, and create next token events. If the token is not free, we pass the token to each host, and check if each host has something to receive. If there is, we copy the package to the host and add the through size and delay using current time minus the copied package arrival time, which is the total delay time. We also need to add one to the count. We then generate the next token event, and push to GEL.

Finally, after the loop, we use total throughput-size/time to get the through put, and

total delay/count is the average package delay.

1. Phase II result:

For arrival Rate: 0.01 ,host number: 10

ThroughPut: 70.543 bytes/second.

Average Packet delay: 3.837076540722659E-4 second.

For arrival Rate: 0.05 ,host number: 10

ThroughPut: 373.6999 bytes/second.

Average Packet delay: 3.8113683753222604E-4 second.

For arrival Rate: 0.1 ,host number: 10

ThroughPut: 799.6354 bytes/second.

Average Packet delay: 3.9729600977053713E-4 second.

For arrival Rate: 0.2 ,host number: 10

ThroughPut: 1486.9984 bytes/second.

Average Packet delay: 3.9687847605518077E-4 second.

For arrival Rate: 0.3 ,host number: 10

ThroughPut: 2227.1814 bytes/second.

Average Packet delay: 3.913034105559905E-4 second.

For arrival Rate: 0.5 ,host number: 10

ThroughPut: 3650.4052 bytes/second.

Average Packet delay: 3.9092368712788464E-4 second.

For arrival Rate: 0.6 ,host number: 10

ThroughPut: 4403.099 bytes/second.

Average Packet delay: 3.9399887713254336E-4 second.

For arrival Rate: 0.7 ,host number: 10

ThroughPut: 5007.0726 bytes/second.

Average Packet delay: 3.91854655954862E-4 second.

For arrival Rate: 0.8 ,host number: 10

ThroughPut: 5900.0729 bytes/second.

Average Packet delay: 3.896816339071689E-4 second.

For arrival Rate: 0.9 ,host number: 10

ThroughPut: 6243.1556 bytes/second.

Average Packet delay: 3.868935117153851E-4 second.

For arrival Rate: 0.01 ,host number: 25

ThroughPut: 179.132 bytes/second.

Average Packet delay: 9.569445823088359E-4 second.

For arrival Rate: 0.05 ,host number: 25

ThroughPut: 892.612 bytes/second.

Average Packet delay: 9.716982407084371E-4 second.

For arrival Rate: 0.1 ,host number: 25

ThroughPut: 1948.1295 bytes/second.

Average Packet delay: 0.0010058281826640782 second.

For arrival Rate: 0.2 ,host number: 25

ThroughPut: 3787.7397 bytes/second.

Average Packet delay: 9.64997416732808E-4 second.

For arrival Rate: 0.3 ,host number: 25

ThroughPut: 5475.7769 bytes/second.

Average Packet delay: 0.0010300711763810066 second.

For arrival Rate: 0.5 ,host number: 25

ThroughPut: 9257.1594 bytes/second.

Average Packet delay: 0.0010806168766259979 second.

For arrival Rate: 0.6 ,host number: 25

ThroughPut: 11108.41 bytes/second.

Average Packet delay: 0.0010329998565211883 second.

For arrival Rate: 0.7 ,host number: 25

ThroughPut: 11233.4695 bytes/second.

Average Packet delay: 0.0010085038431645388 second.

For arrival Rate: 0.8 ,host number: 25

ThroughPut: 15635.3323 bytes/second.

Average Packet delay: 0.0010541553357345857 second.

For arrival Rate: 0.9 ,host number: 25

ThroughPut: 17500.688 bytes/second.

Average Packet delay: 0.0011187977693036552 second.

Graphs

N=10 (Hosts), Through Put:

:

N=10, Average Packet Delay:

N=25, Through Put:

N=25, Average Package Delay

Analysis:

From the result, consider the case when number of hosts is 10. Since the time that token ran through the ring is really small compared to the time between each arrival event. When the arrival rate increase, the package arrived each second increases. Which means that the package transferred each second increases, results the total through put increases. However, when the arrival rate increase, it seems that the queueing delay will increase, but the token ran too fast so that each time the token just take one package at a time while transferring. The gap between each arrival event is too large for the time a token ran through the ring. As a result, we see that the average delay for the whole simulation is pretty stable, and there is not much difference.

When the number of hosts is 25, the through put increases as the arrival rate increases. It is the same reason as we stated above. However, the average package delay increases a little while arrival rate increases. (But still not too much difference). I think this might be that when the host number increases, the total time or delay for token to run through a ring increases, which gives the hosts more time to get a package from the GEL. The delay of 25 is greater than the delay of 10, because the total distance for each token need travel increases, either does the package.

Code:

**import** java.util.\*;

**import** java.text.DecimalFormat;

**public** **class** Phase2 {

**private** **static** **double** *Dp*=1e-5;

**private** **static** **double** *R*=1.25e7;

**private** **static** ArrayList<Event> *GEL*=**new** ArrayList<Event>(); **private** **static** ArrayList<Event> *Frame*=**new** ArrayList<Event>();

**public** **static** **void** main(String[] args) {

**int**[] host\_Number={10,25};

**double**[] arrival\_Rate={0.01,0.05,0.1,0.2,0.3,0.5,0.6,0.7,0.8,0.9};

**for**(**int** i=0;i<2;i++){

**for**(**int** j=0;j<10;j++) *runSimulation*(host\_Number[i],arrival\_Rate[j]);

}

}

**public** **static** **void** runSimulation(**int** hostNumber,**double** arivRate){

*GEL*.clear();

*Frame*.clear();

Host[] hosts=**new** Host[hostNumber];

**double** delay=0;

**double** time=0;

**double** throuSize=0;

**int** count=0; *initialize*(hosts,hostNumber,arivRate);

**while**(count<500){

Event outEvent=*GEL*.remove(0);

time=outEvent.eventTime;

**if**(outEvent.eType==0){

hosts[outEvent.source].queue.add(outEvent); *genArrEvent*(outEvent,arivRate); }

**else**{

**int** bytes=0;

**int** hostNum=outEvent.position;

**if**(outEvent.free==**true**){

**if**(hosts[hostNum].queue.size()!=0){

**while**(!(hosts[hostNum].queue.isEmpty())){ Event temp=hosts[hostNum].queue.remove(0); bytes+=temp.size; *Frame*.add(temp); }

*genToken*(outEvent,bytes,1,hostNumber); }

**else**{

*genToken*(outEvent,0,0,hostNumber);

}

}

**else**{ bytes=outEvent.size; **if**(hostNum==outEvent.source){ *Frame*.clear(); bytes=0;

}

**for**(**int** i=0;i<*Frame*.size();i++){

**if**(*Frame*.get(i).dest==hostNum){

Event copy=*Frame*.get(i);

delay+=time-copy.eventTime;

throuSize+=copy.size;

count+=1;

}

}

*genToken*(outEvent,bytes,1,hostNumber);

}

}

}

System.***out***.println("For arrival Rate: "+arivRate+" ,host number: "+hostNumber);

System.***out***.println("ThroughPut: "+*round*(throuSize/time)+" bytes/second.");

System.***out***.println("Average Packet delay: "+delay/count+" second.");

System.***out***.println();

}

**public** **static** **void** genToken(Event pre,**int** b,**int** free,**int** hostNumber){

Event event=**new** Event();

event.eType=1;

event.source=pre.source; event.eventTime=pre.eventTime+*Dp*+b/*R*;

event.size=b;

event.free=**false**;

event.steps=pre.steps+1;

event.position=(pre.position+1)%hostNumber;

**if**(pre.steps==hostNumber || free==0){

event.source=(pre.source+1)%hostNumber; event.free=**true**;

event.steps=0;

}

*GEL*.add(event);

Collections.*sort*(*GEL*);

}

**public** **static** **void** genArrEvent(Event pre, **double** arivRate){

Event event=**new** Event();

event.eType=0;

event.source=pre.source;

event.dest=(**int**)(Math.*random*()\*10);

**while**(event.dest==event.source)

{event.dest=(**int**)(Math.*random*()\*10);}

event.eventTime=pre.eventTime+*negEx*(arivRate);

event.size=(**int**)(Math.*random*()\*1455);

*GEL*.add(event);

Collections.*sort*(*GEL*);

}

**public** **static** **void** initialize(Host[] hosts,**int** hostNumber, **double** arivRate){ //initialization

**for**(**int** i=0 ; i<hostNumber; i++){

hosts[i]=**new** Host();

Event event=**new** Event();

event.eType=0;

event.source=i;

event.dest=(**int**)(Math.*random*()\*10);

**while**(event.dest==i) event.dest=(**int**)(Math.*random*()\*10);

event.eventTime=*negEx*(arivRate);

event.size=(**int**)(Math.*random*()\*1455+64);

*GEL*.add(event);

}

Event tokenP=**new** Event();

tokenP.eType=1;

tokenP.free=**true**;

tokenP.position=1;

tokenP.source=1;

tokenP.eventTime=0;

*GEL*.add(tokenP);

Collections.*sort*(*GEL*);

}

**public** **static** **double** round(**double** d){

DecimalFormat temp=**new** DecimalFormat("#.####");

**return** Double.*valueOf*(temp.format(d));

}

**public** **static** **double** negEx(**double** rate){

**double** u= Math.*random*();

**return** ((-1/rate)\*Math.*log*(1-u));

}

}

**public** **class** Event **implements** Comparable<Event>{

**public** **double** eventTime;

**public** **int** eType;

**public** **int** source;

**public** **int** dest;

**public** **int** size;

**public** **int** position;

**public** **boolean** free;

**public** **int** steps;

@Override

**public** **int** compareTo(Event t){

**double** comparetime=((Event)t).eventTime;

**return** (**int**) ((**this**.eventTime-comparetime)\*100000000);

}

}

**public** **class** Host {

**public** ArrayList<Event> queue=**new** ArrayList<Event>();

}