Data Structure used in This Assignment:

cbwa548 Chengbin Wang
z331313

***Store the graph:**
-associate Class: weighted Graph
-structure used 2 dim table : Edge [][] graph;
where Edge stores all the information about a link between nodes(v1,v2,maxlink,probagation delay)

***establish virtual Link:**
-associate Class: tempDijstra
-structure used
Priority Queue: PriorityQueue<Vertex> priq = new PriorityQueue<Vertex> (max_size,comparator);
-Storing nodes according to their priority, update using ordinary Dijstra;
array : static int st[] store previous node ; static float dist[] store cost from starting node;
Stack : just use to backtrack and find the virtual Link path calculated by Dijstra

***implement Cost Function:**
-associate Class: tempDijstra
-associate function Update(float dist,int i,int j){

```
if(CMD.equals("SDP"))  return dist + graph[i][j].getProbagation();
else if (CMD.equals("LLP")) {
        float rate = ((float)graph[i][j].runTimeLink/graph[i][j].maxlink);
                }else{
}else{  //This is SHP return dist++;}
```

-changing the cost function when using different algrism: generally speaking,
SHP: find nearest path to goal, update cost by one each visiting;
SDP:update each visiting by probagation delay between coming node and next node;
LLP:find all paths and check bottle neck of each link, the smallest bottleneck wins over others.

***Handelling Block:**
-associate Class:WeightedGraph
-structure used is Priority Queue:
PriorityQueue<Edge> dieQ = new PriorityQueue<Edge> (max_size,comparator);
-dieQ stores all processing link in the network: recorded in (v1 v2 TimeToDie);
The TimeToDie value (TTD) is the priority, implemented in
Comparator<Edge> comparator = new EdgeComparator();
-As time passed, before processing new VC, it delete expired link from network, delete the record from priority queue and change the value of Runtime-link number of each edge in the graph.

***random selecting:**
random appears when tie comes, imported random function and 3 algorithm will randomly choose path when tie happens

COMPARE: LLP SHP SDP

---------LLP----------

total number of virtual circuit requests	5884
number of successfully routed requests:	5809
percentage of routed request:	98.72536
number of blocked requests:	75
percentage of blocked requests:	1.2746431
average number of hops per circuit:	3.9499054
average cumulative propagation delay per circuit:	244.06766

---------SHP----------

total number of virtual circuit requests	5884
number of successfully routed requests:	5326
percentage of routed request:	90.516655
number of blocked requests:	558
percentage of blocked requests:	9.483345
average number of hops per circuit:	2.5229065
average cumulative propagation delay per circuit:	165.80548

---------SDP----------

total number of virtual circuit requests	5884
number of successfully routed requests:	5340
percentage of routed request:	90.754585
number of blocked requests:	544

percentage of blocked requests:	9.245411
average number of hops per circuit:	3.135206
average cumulative propagation delay per circuit:	140.21143

Summary:

In general LLT has highest cnnection success rate, averaged 98.7 percent of the cases, it is obvious that LLT has that value. Because, LLT is always trying to find the least congestion path in the whole node, this VC is hard to block for LLT unless all the paths towards destination are blocked.
In return LLT has the significant highest propagation, because it doesn't care about number of nodes it visit and propagation delay of path, it is has significant higher delay than other two and it visits far more node in average along its path.

SDP and SHP have similar success rate, SHP in average visits bit less nodes and SDP has bit less propagation delay. Simply because they set node number and propagation delay as cost priority in Dijkstra respectively. They both doesn't consider congestion in the path, so success rates have been cut down.

The similarity of data in SDP and SHP could be explained as follows : in general there are more propagation delay when visiting more nodes(as this case current topology). There are no extreme example in this case that propagation delay between nodes varies significantly. So propagation delay and average nodes number are similar in these two algorithm, and success rate are relatively similar low.

The random value in selecting tie condition are quite uniform and with small variance. so all three algorithm are quite stable.