CSE 534
Fundamentals of Computer Networks
Project 2

Sivaram Mothiki
109396387

A1)

1) ndnSIM has been successfully compiled and used under Ubuntu Linux 12.04 (boost libraries 1.48) . To install Boost Libraries we run the below command

```
sudo aptitude install libboost1.48-all-dev
```

2) To install visualizer the following python packages should be installed

```
sudo apt-get install python-dev python-pygraphviz python-kiwi
sudo apt-get install python-pygoocanvas python-gnome2
sudo apt-get install python-gnomedesktop python-rsvg ipython
```

3) We create a directory to download and run the ndnSIM source from git clone

```
git clone git://github.com/cawka/ns-3-dev-ndnSIM.git ns-3
git clone git://github.com/cawka/pybindgen.git pybindgen
git clone git://github.com/NDN-Routing/ndnSIM.git ns-3/src/ndnSIM
```

4) ndnSIM uses standard NS-3 compilation procedure. Normally the following commands should be sufficient to configure and build ndnSIM with python bindings enabled:

```
cd <ns-3-folder>
./waf configure --enable-examples
./waf
```

5) To compile a particular project . Below are the commands we have to run

```
./waf --run=<project name>
./waf --run=<project name> --vis // for visualizer
```

```
   Consumer1                                              Producer 1
#   /------\                                                  /------\
#   | NodeA1 |<--+                                          +-->| NodeB1 |
#   \------/      \                                        /      \------/
#                  \     Router1                    Router2  /
#                   +-->/------ \                    /------\<-+
#                        | NodeA |<================>| NodeB|
#                   +-->\------/                    \------/<-+
#                  /                                         \
#   /------\      /                                           \     /------\
#   | NodeA2 |<--+                                             +-->| NodeB2 |
#   \------/                                                      \------/
Consumer2                                                      producer2
```

Fig 1

Configuration from topology file to set up dumb-bell network

```
router

# node  comment   yPos xPos
NodeA1   NA        1      3
NodeA2   NA        3      3
NodeA        NA    2      5
NodeB        NA    2      7
NodeB1   NA        1      9
NodeB2   NA        3      9

link

# srcNode   dstNode      bandwidth   metric        delay   queue
NodeA1       NodeA       10Mbps       1       10ms 20
NodeA2       NodeA       10Mbps       1       10ms 20
NodeA        NodeB       10Mbps       1       10ms 20
NodeB1       NodeB       10Mbps       1       10ms 20
NodeB2       NodeB       10Mbps       1       10ms 20
```

Created a topology file that describes dumb-bell network topology mentioned in Fig 1 and read the file using AnnotatedTopologyReader .Created two nodes consumer 1 and consumer2 that are hanging to NodeA (router1) . Created producer 1 and producer2 that are hanging to NodeB (router2) . On core nodes (NodeA and NodeB) enabled caching and on end nodes (consumer1 , consumer2, producer1, producer2) disabled caching .The cache size on the core nodes is 3 objects and policy is LRU using SetContentStore method .

Using SetPrefix method we defined a object space that producer cater to Created two ConsumerCBR applications - one to request content from Producer1 and the other from Producer2. In Consumer1, the application is configured to request content from Producer1 starting at time 0.0 and to request content from Producer2 starting at time 1.0. Similarly in Consumer2, the application is configured to request content from Producer1 starting at time 1.0 and to request content from Producer2 starting at time 1.0. This is done using StartSeq parameter in SetAttribute method.

Kept generation rate for interest packets in each ConsumerCBR to 1 interest packet per second. Using SetForwardingStrategy set the fowarding strategy to bestroute . Used GlobalRoutingHelper to populate FIBs(Forwarding Index Base)

Code to read the topology and set up dumb -bell network accroding to the given specifications

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/ndnSIM-module.h"

using namespace ns3;

int
main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);

  AnnotatedTopologyReader topologyReader ("", 25);
  topologyReader.SetFileName ("scratch/topo-6-node.txt");
  topologyReader.Read ();

  // Install NDN stack on all nodes
  /*ndn::StackHelper ndnHelper;
  ndnHelper.SetForwardingStrategy ("ns3::ndn::fw::BestRoute");
  ndnHelper.SetContentStore ("ns3::ndn::cs::Lru",
                                   "MaxSize", "10000");
  ndnHelper.InstallAll ();

  // Installing global routing interface on all nodes
  ndn::GlobalRoutingHelper ndnGlobalRoutingHelper;
  ndnGlobalRoutingHelper.InstallAll ();*/

  // Getting containers for the consumer/producer
  Ptr<Node> consumer1 = Names::Find<Node> ("NodeA1");
  Ptr<Node> consumer2 = Names::Find<Node> ("NodeA2");

  Ptr<Node> producer1 = Names::Find<Node> ("NodeB1");
  Ptr<Node> producer2 = Names::Find<Node> ("NodeB2");

  Ptr<Node> router1 = Names::Find<Node> ("NodeA");
  Ptr<Node> router2 = Names::Find<Node> ("NodeB");

  ndn::StackHelper ndnHelper;
```

```
ndnHelper.SetForwardingStrategy ("ns3::ndn::fw::BestRoute");
ndnHelper.SetContentStore ("ns3::ndn::cs::Nocache");
ndnHelper.Install (consumer1);
ndnHelper.Install (consumer2);
ndnHelper.Install (producer1);
ndnHelper.Install (producer2);
ndnHelper.SetContentStore ("ns3::ndn::cs::Lru", "MaxSize", "3");
ndnHelper.Install (router1);
ndnHelper.Install (router2);

ndn::GlobalRoutingHelper ndnGlobalRoutingHelper;
ndnGlobalRoutingHelper.InstallAll ();

ndn::AppHelper consumerHelper ("ns3::ndn::ConsumerCbr");
consumerHelper.SetAttribute ("Frequency", StringValue ("1")); // 100 interests a second
consumerHelper.SetAttribute ("StartSeq", StringValue ("0"));

consumerHelper.SetPrefix ("/NodeB1");
consumerHelper.Install (consumer1);
//ApplicationContainer consumerNode1=consumerHelper.Install (consumer1);
//consumerNode1.Start (Seconds (0.0));

ndn::AppHelper consumerHelper2 ("ns3::ndn::ConsumerCbr");
consumerHelper2.SetAttribute ("Frequency", StringValue ("1")); // 100 interests a second
consumerHelper2.SetAttribute ("StartSeq", StringValue ("1"));

consumerHelper2.SetPrefix ("/NodeB2");
consumerHelper2.Install (consumer1);


consumerHelper.SetAttribute ("Frequency", StringValue ("1")); // 100 interests a second
consumerHelper.SetAttribute ("StartSeq", StringValue ("1"));

consumerHelper.SetPrefix ("/NodeB1");
consumerHelper.Install (consumer2);
//ApplicationContainer consumerNode1=consumerHelper.Install (consumer1);
//consumerNode1.Start (Seconds (0.0));

consumerHelper2.SetAttribute ("Frequency", StringValue ("1")); // 100 interests a second
consumerHelper2.SetAttribute ("StartSeq", StringValue ("0"));
```

```
  consumerHelper2.SetPrefix ("/NodeB2");
  consumerHelper2.Install (consumer2);

  //consumerHelper.SetPrefix ("/NodeB2");
  //consumerHelper.Install (consumer2);

  //ApplicationContainer consumerNode2=consumerHelper.Install (consumer2);
  //consumerNode2.Start (Seconds (1.0));
  ndn::AppHelper producerHelper ("ns3::ndn::Producer");
  producerHelper.SetAttribute ("PayloadSize", StringValue("1024"));

  ndnGlobalRoutingHelper.AddOrigins ("/NodeB1", producer1);
  producerHelper.SetPrefix ("/NodeB1");
  producerHelper.Install (producer1);


  ndnGlobalRoutingHelper.AddOrigins ("/NodeB2", producer2);
  producerHelper.SetPrefix ("/NodeB2");
  producerHelper.Install (producer2);

  // Calculate and install FIBs
  ndn::GlobalRoutingHelper::CalculateRoutes ();

  Simulator::Stop (Seconds (5.0));

  //ndn::L3AggregateTracer::InstallAll ("aggregate-trace.txt", Seconds (0.5));
  ndn::AppDelayTracer::InstallAll ("app-delays-trace.txt");
  Simulator::Run ();
  Simulator::Destroy ();

  return 0;
}
```

A2)

A new Packet tag which keeps track of cache hit is created using the following steps.

1) Created a new CacheHitTag class in ndn-cachehit-tag.h.In this class declared a member named cache_hit which is set to 1 using contructor when and the object is created whenever there is a cache hit.

2)Created ndn-cachehit-tag.cc which is the implementation of ndn-cachehit-tag.h and the class CacheHitTag . Increment method was declared and left blank.

3) Modified ndn-app-delay-tracer.h and ndn-app-delay-tracer.cc to add extra column for cache hit which is responsible for tracing and generating appdelaytracer.txt.The following funtions were modified PrintHeader, LastRetransmittedInterestDataDelay and FirstInterestDataDelay to add extra column for cacheHitcount .

4) Modified ndn-consumer.h and ndn-consumer.cc , these are the class files that are responsible for calling ndn-app-delay-tracer.cc methods to print output . Added a tag for cahchehit similar to hopcount .

5) Modified the OnInterest method in the ndn-forwarding-strategy.cc. We added code to create a cache hit packet tag and add it to the packet . it checks whether there is a cache hit in the contentstore and add the cahce hit packet tag to the packet

B1)

Initially ran the experiment with various values cache disabled and slowly increased the cache size and collected the data for each cache size .

Cache Size	Hop Counts	Cache Hits	Cache Miss
0	368402	0	49358
50	348302	12448	36910
100	334722	17765	31593
500	276790	34096	15260
1000	239534	39205	10153
2000	205686	40137	9221

Below chart confirms that hop count decreases as there is increase in cache size

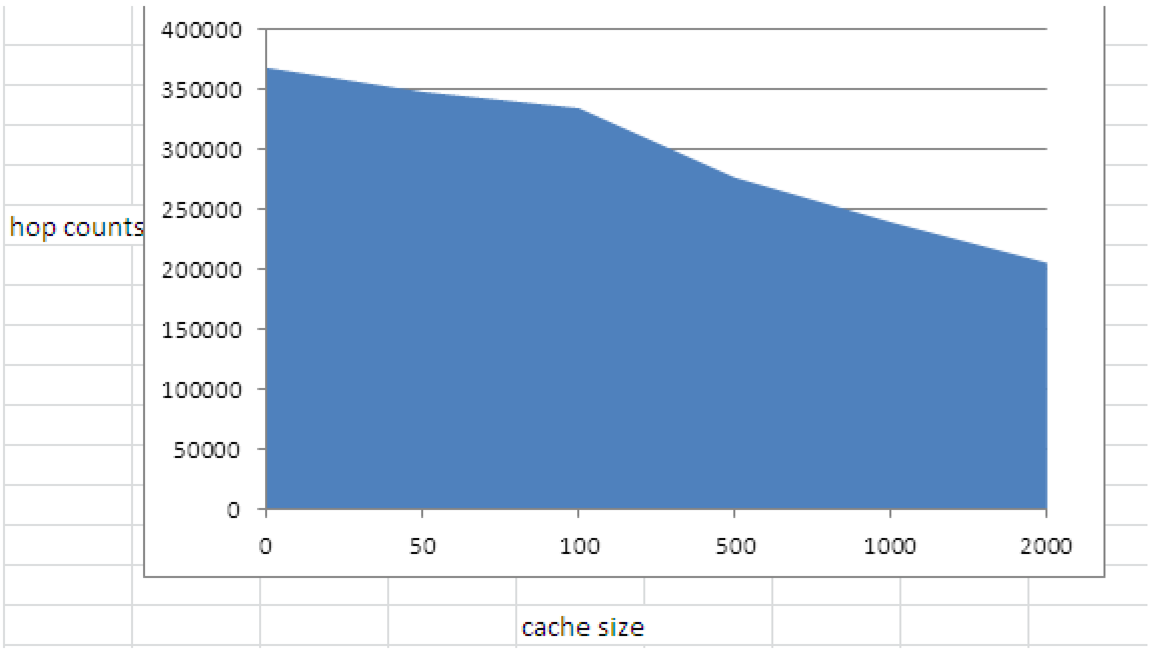


Fig 1

As the cache size increases more number of packets get stored in the nodes and chances of finding a particular packet in the closer node increases .So the probability of finding a packet in the closer nodes cache increases than the producer of the object .

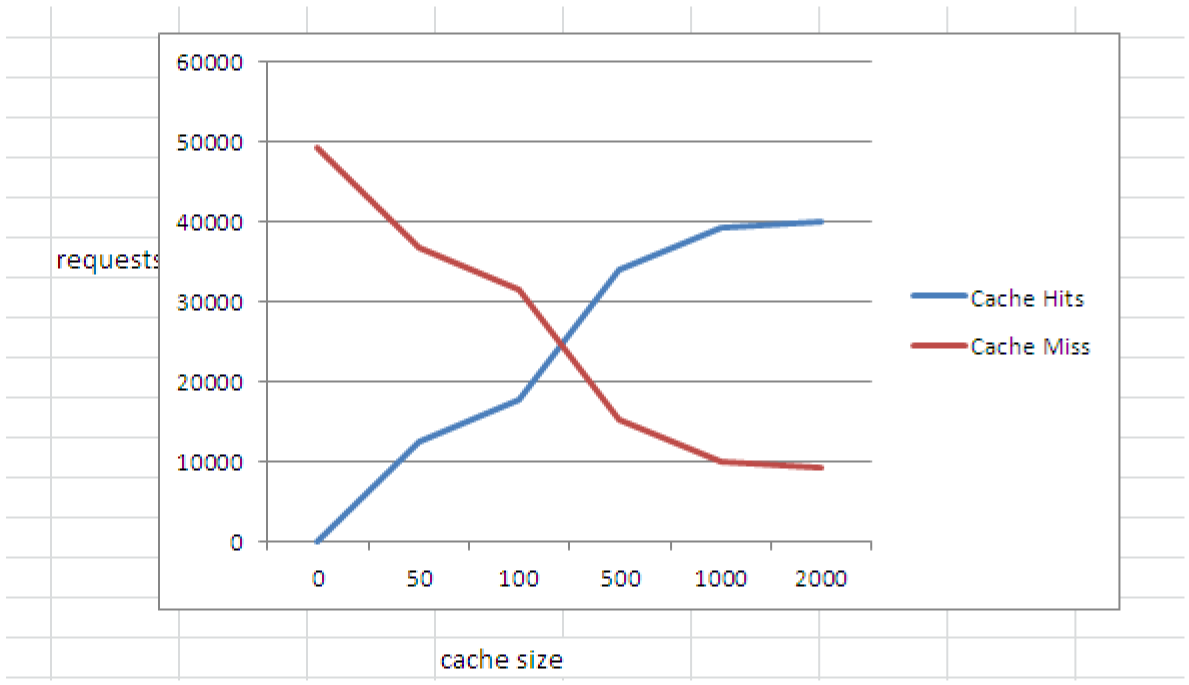


Fig 2

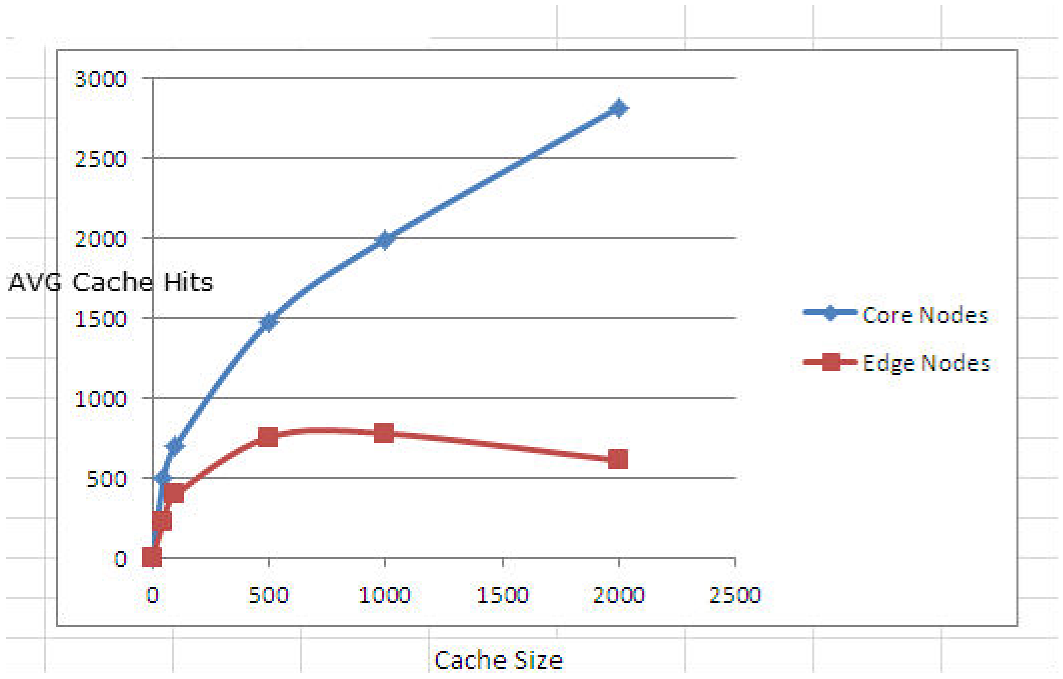
If we observe the pattern of the graphs above we can convey that they are not following any monotonic or constant pattern .

As we can see at some point the graphs will gradually level off . The same is applied to cache hits.

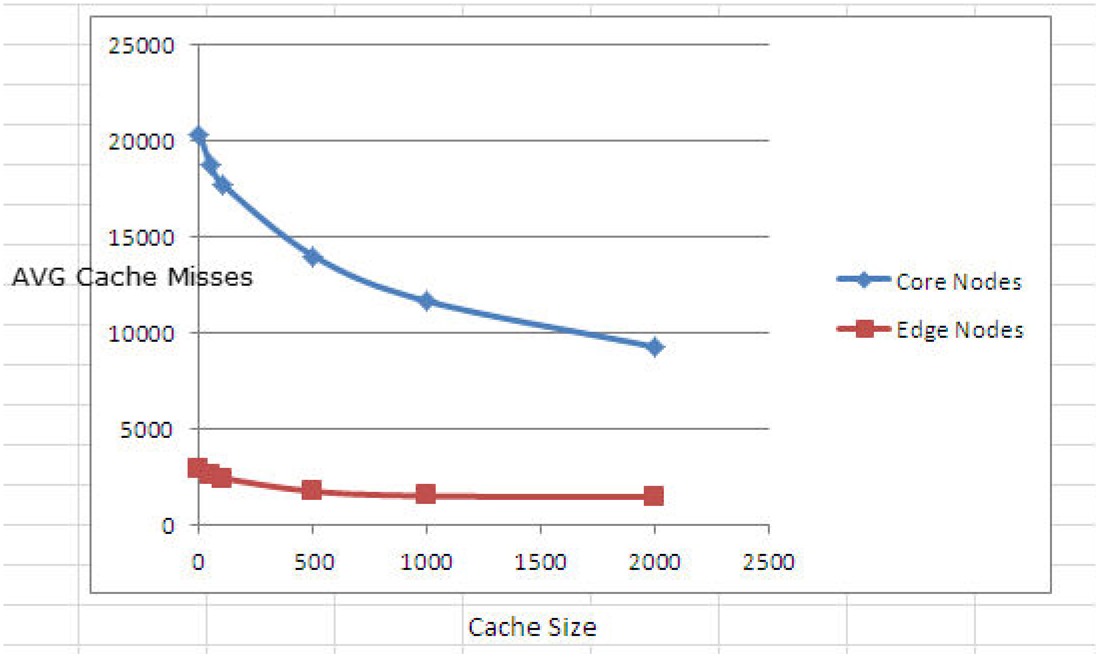
As the cache size increases the we can see there is a decrease in cache misses and increase in cache hits . The popularity profile defines how often a particular object is being requested . More popular objects will be requested more times and cahces will be filled with these popular objects which will eventually decrease the number fo hops and cache misses and increase the cache hits . If we increase the values of q and s we are increasing the popularity profile and even small cache sizes will have more cache hits and less hop counts as time proceeds . There wil be a direct correlation between popularity profile and cache hits .

B2)

From the graph we can observe that the cache hits no the core nodes increased much quicker than the edge nodes as the cache size increases . But from cache size 1000 on wards the number of cache hits on the edge nodes gradually decreased this confirms that the objects are being fetched from the core nodes itself for more cache sizes .



If we consider the stats for cache misses initially the cache misses are very high for core nodes as they are not the producers of the content . But as cache size

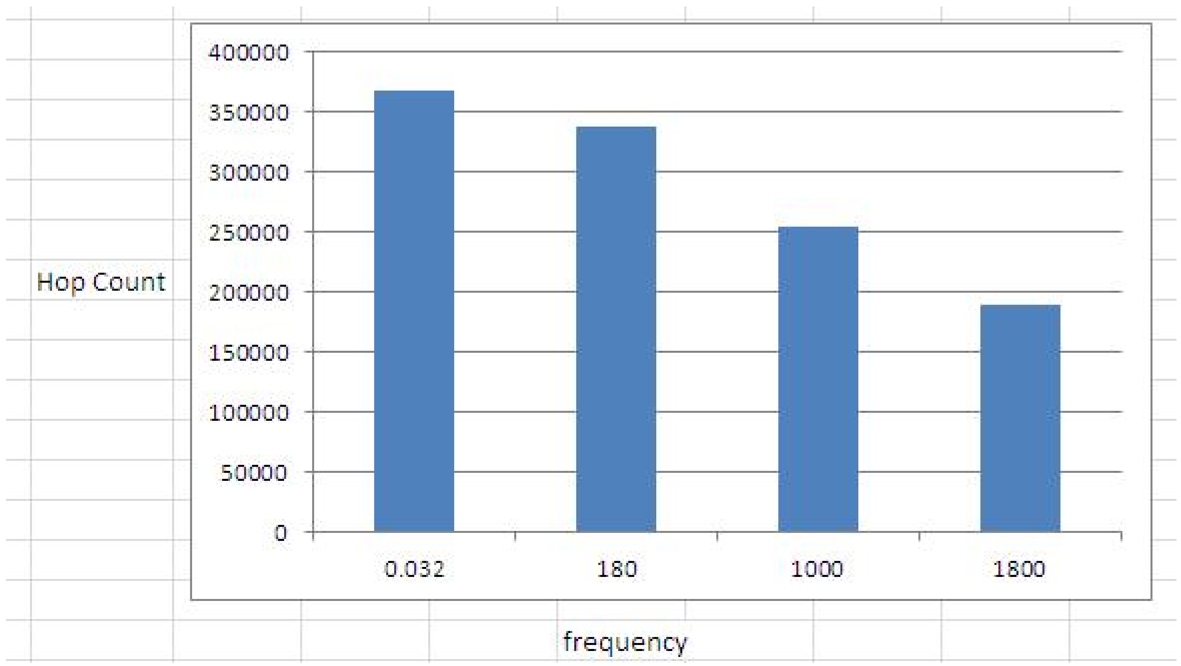


increased the cache misses decreased quickly . From the observations core nodes have more cache hits than edge nodes . By maintaining larger cache size in the network will definitely improve cache hits on the core nodes and decrease hop counts as well and having small cache sizes on the edge nodes might not have that much effect on the network as most of the cache hits are handled by core nodes .

C1)
We have taken a constant of 50000 interest packets for the experiment and used the following Frequency and simulation times .
These were calculated by the formula time = 50000/(frequency * 31)

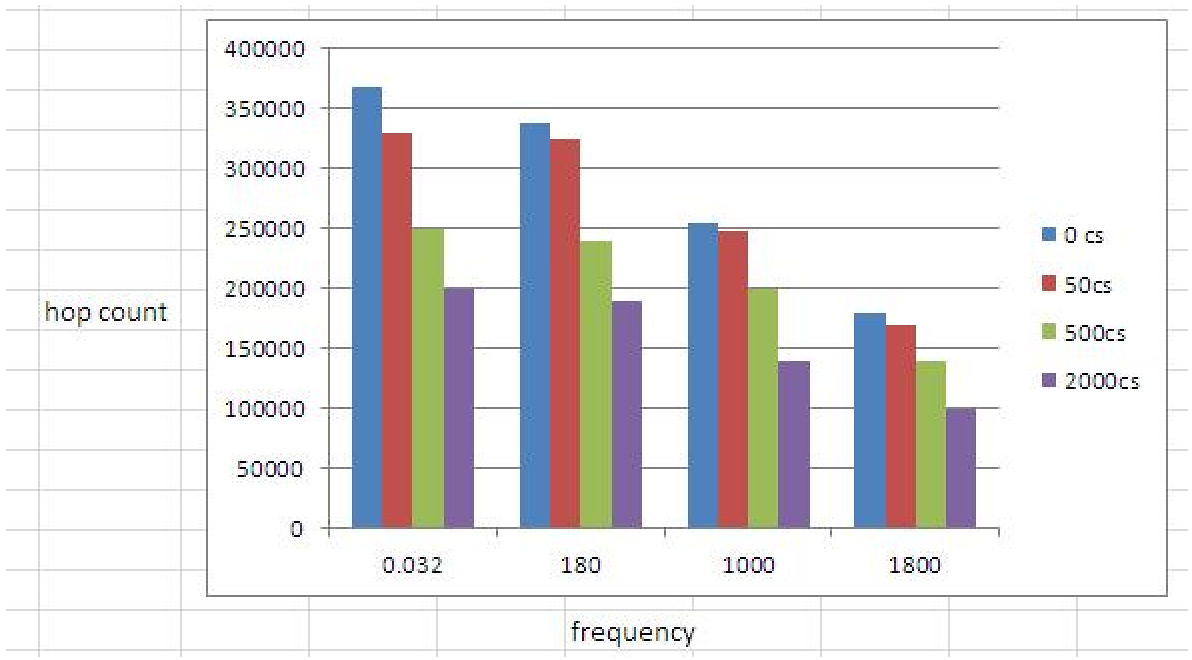
Frequency	Time
0.032	50000
180	8.96
1000	1.61
1800	0.896

Frequency vs Hop Count



We can clearly see that the number of hops reduces linearly as traffic size increases almost linearly

C2)
Analysed hop counts with changing frequency and cache sizes .



From the above performed experiments we have observed that hop count decreases with increase in cache size and Hop count decreases with increase in frequency in case of PIT aggregation . But you can clearly observe that increase in cache size in lower frequencies saves lot of hops. If there is more cache size then hop saves have no significant difference by varying frequencies . If you can see for Cache size of 500 and above if we keep increasing the frequency there is a very little difference in the hop counts. But in case of PIT aggregation it is linearly decreasing .