CSE534

Fundamentals of Computer Networks

Project-2

Udit Gupta and Rajesh Golani

Table of Contents

Part A.

A1:

Experience in setting up and getting the dumb bell network going

Instruction regarding ndnSIM installation were pretty clear but they were outdated. Installation was giving problem with newer versions of libncurses which according to documentations should have worked. After installing ncurses 1.49, it got installed without any further problems.

Setting up the dumb bell ring was pretty straight forward task. We followed examples available on ndnSIM page, which was also pointed out in project handout document.

In project handout, it is mentioned that **StartSeq** parameter of the **SetAttribute** should be used to start consumer at specified time. But we were facing some issue with this. From examples page, we found out that same can be accomplished using **start()** method of **ApplicationContainer**. So, we used this instead of one mentioned in handout.

Apart from this, we did not face any issues in making dumb bell going.

Scripts

Topology:

```
# A1-topo.txt
router
# node  comment     yPos    xPos
C1   Consumer1       1       3
C2   Consumer2       3       3
A   Core Node A       2       5
B   Core Node B       2       7
P1   Producer1       1       9
P2   Producer2       3       9
link
#srcNode  dstNode     bandwidth   metric  delay   queue
C1        A           10Mbps      1       10ms    20
C2        A           10Mbps      1       10ms    20
A         B           10Mbps      1       10ms    20
P1        B           10Mbps      1       10ms    20
P2        B           10Mbps      1       10ms    20
```

CC file

```
#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 ("src/ndnSIM/examples/topologies/A1-
topo.txt");
      topologyReader.Read ();

      // Install NDN stack on all nodes
      ndn::StackHelper ndnHelper;
      ndnHelper.SetForwardingStrategy ("ns3::ndn::fw::BestRoute");

      //Setting content store to 3 for core nodes
      ndnHelper.SetContentStore ("ns3::ndn::cs::Lru",
                  "MaxSize", "3");
      ndnHelper.Install (Names::Find<Node> ("A"));
      ndnHelper.Install (Names::Find<Node> ("B"));

      //Setting content store to 0 for edge nodes
      ndnHelper.SetContentStore ("ns3::ndn::cs::Lru",
                  "MaxSize", "0");
      ndnHelper.Install (Names::Find<Node> ("C1"));
      ndnHelper.Install (Names::Find<Node> ("C2"));
      ndnHelper.Install (Names::Find<Node> ("P1"));
      ndnHelper.Install (Names::Find<Node> ("P2"));

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

      // Getting containers for the consumer/producer
      Ptr<Node> consumer1 = Names::Find<Node> ("C1");
      Ptr<Node> consumer2 = Names::Find<Node> ("C2");

      Ptr<Node> producer1 = Names::Find<Node> ("P1");
      Ptr<Node> producer2 = Names::Find<Node> ("P2");

      // Setting up helperts for Consumer 1
      ndn::AppHelper consumerHelperC11 ("ns3::ndn::ConsumerCbr");
      consumerHelperC11.SetAttribute ("Frequency", StringValue ("1"));
      //consumerHelperC11.SetAttribute ("StartSeq", StringValue ("0.0"));
      consumerHelperC11.SetPrefix ("/P1");
      //Schedule Consumer as per specification
      ApplicationContainer consumer = consumerHelperC11.Install (consumer1);
      consumer.Start (Seconds (0.0));
```

```
        ndn::AppHelper consumerHelperC12 ("ns3::ndn::ConsumerCbr");
        consumerHelperC12.SetAttribute ("Frequency", StringValue ("1"));
        //consumerHelperC11.SetAttribute ("StartSeq", StringValue ("1.0"));
        consumerHelperC12.SetPrefix ("/P2");
        //Schedule Consumer as per specification
        consumer = consumerHelperC12.Install (consumer1);
        consumer.Start (Seconds (1.0));

        // Setting up helperts for Consumer 2
        ndn::AppHelper consumerHelperC21 ("ns3::ndn::ConsumerCbr");
        consumerHelperC21.SetAttribute ("Frequency", StringValue ("1"));
        //consumerHelperC11.SetAttribute ("StartSeq", StringValue ("0.0"));
        consumerHelperC21.SetPrefix ("/P2");
        //Schedule Consumer as per specification
        consumer = consumerHelperC21.Install (consumer2);
        consumer.Start (Seconds (0.0));

        ndn::AppHelper consumerHelperC22 ("ns3::ndn::ConsumerCbr");
        consumerHelperC22.SetAttribute ("Frequency", StringValue ("1"));
        //consumerHelperC11.SetAttribute ("StartSeq", StringValue ("1.0"));
        consumerHelperC22.SetPrefix ("/P1");
        //Schedule Consumer as per specification
        consumer = consumerHelperC22.Install (consumer2);
        consumer.Start (Seconds (1.0));


        ndn::AppHelper producerHelper ("ns3::ndn::Producer");
        producerHelper.SetAttribute ("PayloadSize", StringValue("1024"));

        // Register /P1 prefix with global routing controller and
        // install producer that will satisfy Interests in /dst1 namespace
        ndnGlobalRoutingHelper.AddOrigins ("/P1", producer1);
        producerHelper.SetPrefix ("/P1");
        producerHelper.Install (producer1);

        // Register /P2 prefix with global routing controller and
        // install producer that will satisfy Interests in /dst2 namespace
        ndnGlobalRoutingHelper.AddOrigins ("/P2", producer2);
        producerHelper.SetPrefix ("/P2");
        producerHelper.Install (producer2);

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

        Simulator::Stop (Seconds (5.0));

        ndn::AppDelayTracer::InstallAll ("app-delays-trace.txt");
        ndn::CsTracer::InstallAll ("cs-trace.txt", Seconds (1));

        Simulator::Run ();
        Simulator::Destroy ();

        return 0;
}
```

A2:

Modifications you had to implement in order to introduce the new packet tag.

To add new packet tag to data packet, we carried out following changes:

- We created two files for tag creation namely **ndn-cache-hit-tag.cc** and **ndn-cache-hit-tag.h**
  - These files were places in **ndnSIM/ns-3/src/ndnSIM/utils**
- After this, we change **onInterest()** method in **ndn-forwarding-strategy.cc** to add tag to data packet. This tag will contain 1, if packet was served from cache hit else 0.
- After that, we changed **ndn-consumer.h** and **ndn-consumer.cc** located in **ndnSIM/ns3/src/ndnsim/apps** to include cache hit field for App delay tracer.
- Same way, we changed three methods in **ndn-app-delay-tracer.h** and **ndn-app-delay-tracer.cc** located in **src/ndnSIM/utils/tracers** to process and print cache hit tag.
  - Changed methods were : **PrintHeader(), LastRetransmittedInterestDataDelay () and FirstInterestDataDelay ().**

OnInterest() method

In OnInterest() method, we changed code at following places:

- We created CacheHitTag. This is the tag we want to add to data packet.
- Then inside condition that checks whether data is available in content store, we added created CacheHitTag to the packet with value of 1 to indicate cache hit.
- Following is the snippet for the of the method with changes made by us highlighted.

```
Void ForwardingStrategy::OnInterest (Ptr<Face> inFace, Ptr<Interest> interest)
{
  NS_LOG_FUNCTION (inFace << interest->GetName ());
  m_inInterests (interest, inFace);

  Ptr<pit::Entry> pitEntry = m_pit->Lookup (*interest);
  bool similarInterest = true;
  if (pitEntry == 0)
    {
      similarInterest = false;
      pitEntry = m_pit->Create (interest);
      if (pitEntry != 0)
        {
          DidCreatePitEntry (inFace, interest, pitEntry);
        }
      else
        {
          FailedToCreatePitEntry (inFace, interest);
          return;
        }
```

```
    }

  bool isDuplicated = true;
  if (!pitEntry->IsNonceSeen (interest->GetNonce ()))
    {
      pitEntry->AddSeenNonce (interest->GetNonce ());
      isDuplicated = false;
    }

  if (isDuplicated)
    {
      DidReceiveDuplicateInterest (inFace, interest, pitEntry);
      return;
    }

  Ptr<Data> contentObject;
  contentObject = m_contentStore->Lookup (interest);
  if (contentObject != 0)
    {
      FwHopCountTag hopCountTag;
      if (interest->GetPayload ()->PeekPacketTag (hopCountTag))
        {
          contentObject->GetPayload ()->AddPacketTag (hopCountTag);
        }

      std::cout<<"IPK-CHIT "<<Simulator::Now ()<<" "<<Names::FindName (pitEntry-
>GetFibEntry ()->GetFib()->GetObject<Node>())<<" "<<interest->GetName()<<" -1
"<<interest->GetNonce()<<" "<<hopCountTag.Get()<<std::endl;

      CacheHitTag cacheHitTag;
      cacheHitTag.Increment();
      contentObject->GetPayload ()->AddPacketTag (cacheHitTag);
      //pitEntry->AddIncoming (inFace/*, Seconds (1.0)*/);
      pitEntry->AddIncoming (inFace, hopCountTag.Get());

      // Do data plane performance measurements
      WillSatisfyPendingInterest (0, pitEntry);

      // Actually satisfy pending interest
      SatisfyPendingInterest (0, contentObject, pitEntry);
      return;
    }

  if (similarInterest && ShouldSuppressIncomingInterest (inFace, interest,
pitEntry))
    {

        ....
}
```

AppDelayTracer

After making above mentioned changes, AppDelayTracer was displaying the cache hit field that we added. Here is that output for the script that we created in A1:

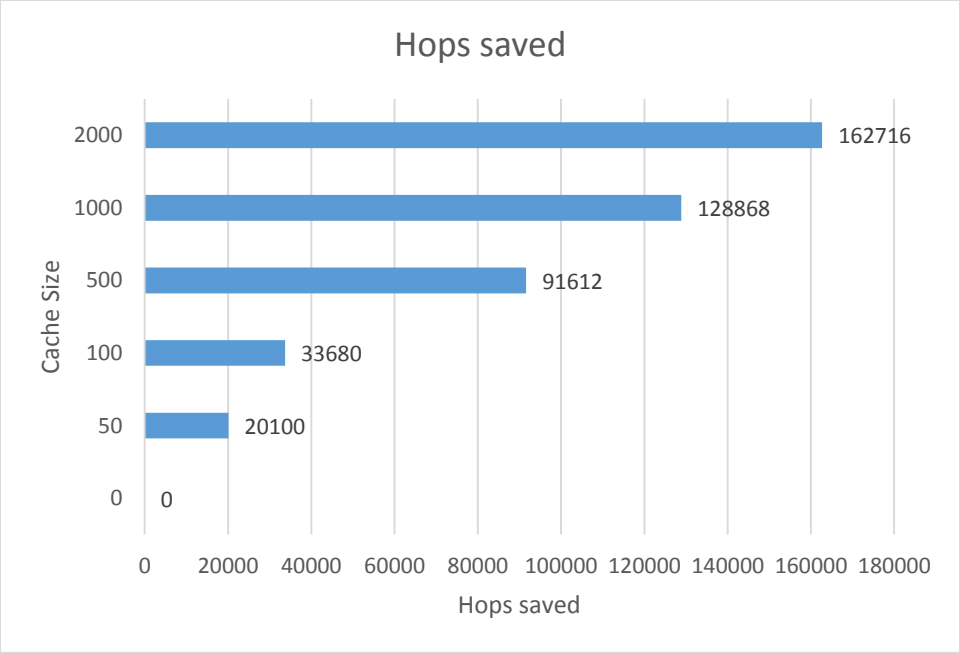
Time	Node	AppId	SeqNo	Type	DelayS	DelayUS	RetxCount	HopCoun	CacheHit
0.0626016	C1	1	0	LastDelay	0.0626016	62601.6	1	6	0
0.0626016	C1	1	0	FullDelay	0.0626016	62601.6	1	6	0
0.0634472	C2	1	0	LastDelay	0.0634472	63447.2	1	6	0
0.0634472	C2	1	0	FullDelay	0.0634472	63447.2	1	6	0
1.02089	C1	2	0	LastDelay	0.0208896	20889.6	1	2	1
1.02089	C1	2	0	FullDelay	0.0208896	20889.6	1	2	1
1.02089	C2	2	0	LastDelay	0.0208896	20889.6	1	2	1
1.02089	C2	2	0	FullDelay	0.0208896	20889.6	1	2	1
1.06261	C1	1	1	LastDelay	0.0626064	62606.4	1	6	0
1.06261	C1	1	1	FullDelay	0.0626064	62606.4	1	6	0
1.06345	C2	1	1	LastDelay	0.0634528	63452.8	1	6	0
1.06345	C2	1	1	FullDelay	0.0634528	63452.8	1	6	0
2.02089	C1	2	1	LastDelay	0.0208912	20891.2	1	2	1
2.02089	C1	2	1	FullDelay	0.0208912	20891.2	1	2	1
2.02089	C2	2	1	LastDelay	0.0208912	20891.2	1	2	1
2.02089	C2	2	1	FullDelay	0.0208912	20891.2	1	2	1
2.06261	C1	1	2	LastDelay	0.0626064	62606.4	1	6	0
2.06261	C1	1	2	FullDelay	0.0626064	62606.4	1	6	0
2.06345	C2	1	2	LastDelay	0.0634528	63452.8	1	6	0
2.06345	C2	1	2	FullDelay	0.0634528	63452.8	1	6	0
3.02089	C1	2	2	LastDelay	0.0208912	20891.2	1	2	1
3.02089	C1	2	2	FullDelay	0.0208912	20891.2	1	2	1
3.02089	C2	2	2	LastDelay	0.0208912	20891.2	1	2	1
3.02089	C2	2	2	FullDelay	0.0208912	20891.2	1	2	1
3.06261	C1	1	3	LastDelay	0.0626064	62606.4	1	6	0
3.06261	C1	1	3	FullDelay	0.0626064	62606.4	1	6	0
3.06345	C2	1	3	LastDelay	0.0634528	63452.8	1	6	0
3.06345	C2	1	3	FullDelay	0.0634528	63452.8	1	6	0
4.02089	C1	2	3	LastDelay	0.0208912	20891.2	1	2	1
4.02089	C1	2	3	FullDelay	0.0208912	20891.2	1	2	1
4.02089	C2	2	3	LastDelay	0.0208912	20891.2	1	2	1
4.02089	C2	2	3	FullDelay	0.0208912	20891.2	1	2	1
4.06261	C1	1	4	LastDelay	0.0626064	62606.4	1	6	0
4.06261	C1	1	4	FullDelay	0.0626064	62606.4	1	6	0
4.06345	C2	1	4	LastDelay	0.0634528	63452.8	1	6	0
4.06345	C2	1	4	FullDelay	0.0634528	63452.8	1	6	0

Part B.

B1.

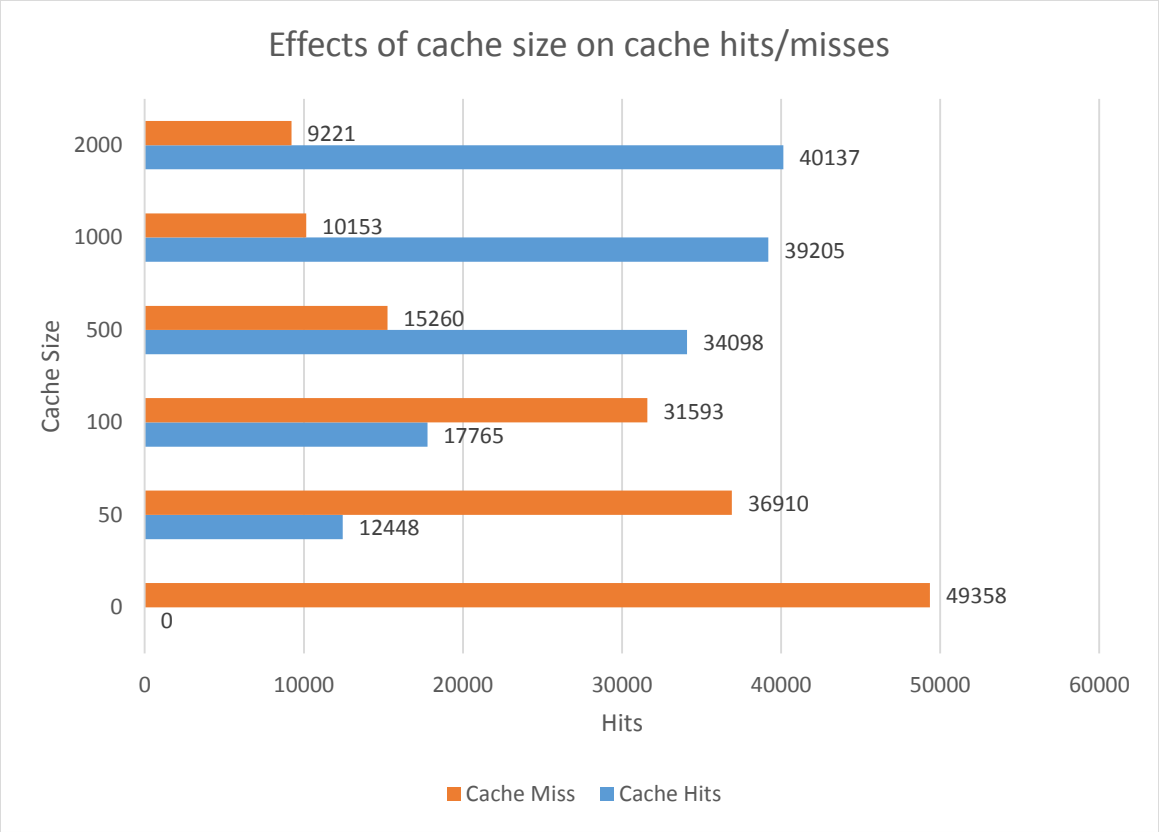
Compare and report on the total number of hops saved as we increase the caches sizes

Following chart depicts hops that are being saved as we increase the cache size. Here, we can see that as we increase the cache size hops that being saved are increasing.



Number of requests satisfied from cache hits vs. those satisfied from the owning repository nodes

If we increase the cache size, we can notice that more and more requests are being served from cache. But the effectiveness of this increase is not increasing monotonically.



Is there a consistent, significant, monotonic pattern of savings in terms of reduced number of hops?

If we notice first graph closely, we see that as we increase cache size we are getting saving more hops. But at one point, the number of hop savings are not getting increased with relative to cache size.

For example, from graph if we cache size from 100 to 500, we are saving 60000 hops but when we increase cache size from 1000 to 2000, we are getting ~34000 hops saving.

Thus there is no consistent, significant, monotonic pattern.

Do you think this trend would continue if we were to further increase the cache size, or would it level off beyond a certain point?

No, this trend will not continue. As we further increase the cache size, its effect on hops saving will decrease and at certain point it won't be effective anymore.

Do you think that your results have any correlation with the "popularity profile" defined by the ndn-consumer-zipf-madelbrot application?

Yes, our results have correlation with the popularity profile defined in ndn-consumer-zipf-mandelbrot.

As we increase the value of q and s, more interest request will come for the same data packets. This will result in more cache hits than the results. Due to this, small size at the nodes would be sufficient to reduce the hops. So, if we increase the q and s, we might get more hop savings than we got in our results.

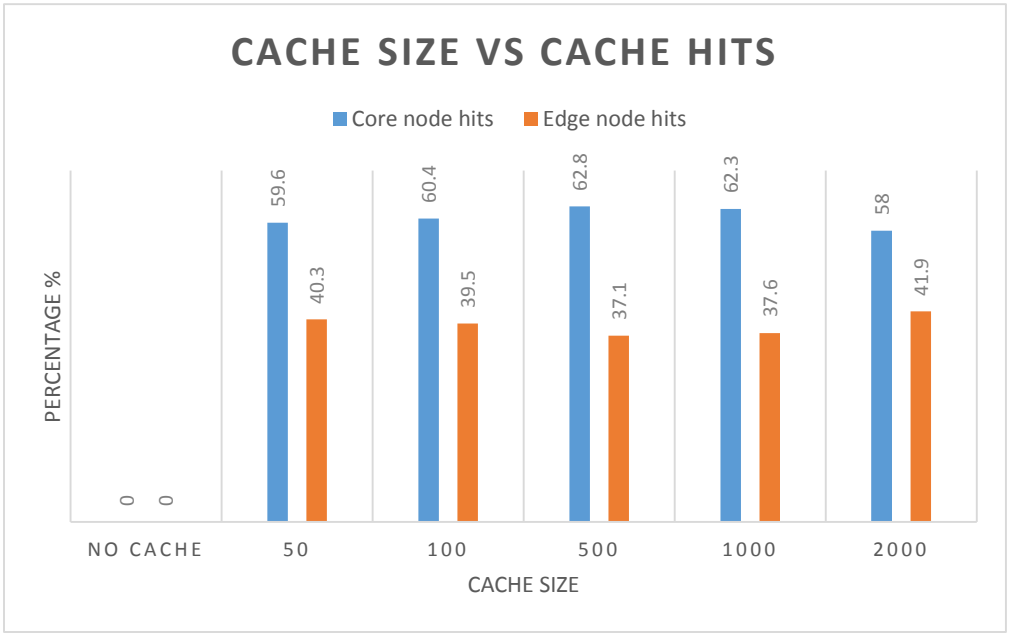
Same way if we decrease the q and s, interest for the same data packets will decrease resulting in more cache misses and less hop savings. At that time, same cache size as we used for our experiments will not result in same hop savings.

B2.

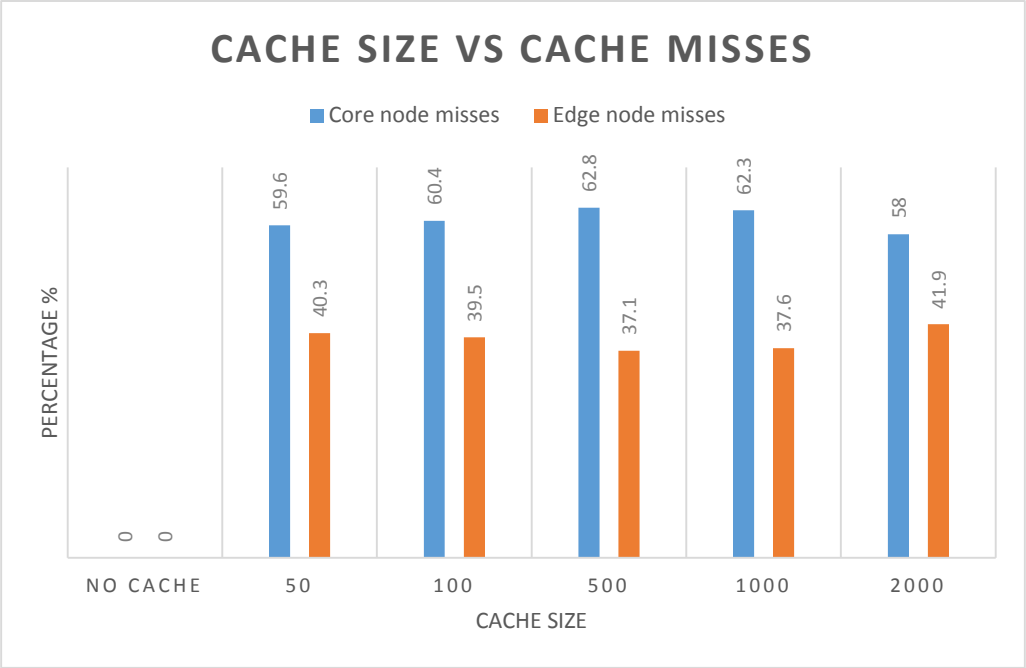
Do the core nodes have more cache hits compared to the edge nodes?

Following are the two graphs that compares the effect of cache introduction to both core nodes and edge nodes.

- First graph provides comparison of cache hits for different cache sizes between core nodes and edge nodes. From the graph we can see that cache hits at core nodes are increasing as we increase the cache size. This is because now core nodes can store more objects and serving more requests without forwarding these to edge nodes.



- Second graph compares the results of cache misses for different cache sizes between core nodes and edge nodes. From the graph, we can see that there is not much of change in cache misses on changing cache sizes.



Do you think it would make sense to have larger cache sizes in the core nodes, and smaller ones at the edge; or vice versa?

It would make sense to have larger cache size at core nodes than the edge nodes. In out topology, there is no direct link between two edge nodes. Every interest request has to go through one or more core nodes. So, if we increase the core nodes cache size the chances of interest being found in core nodes increases. This will also result in more cache hits and less hop count.

Part C

C1.

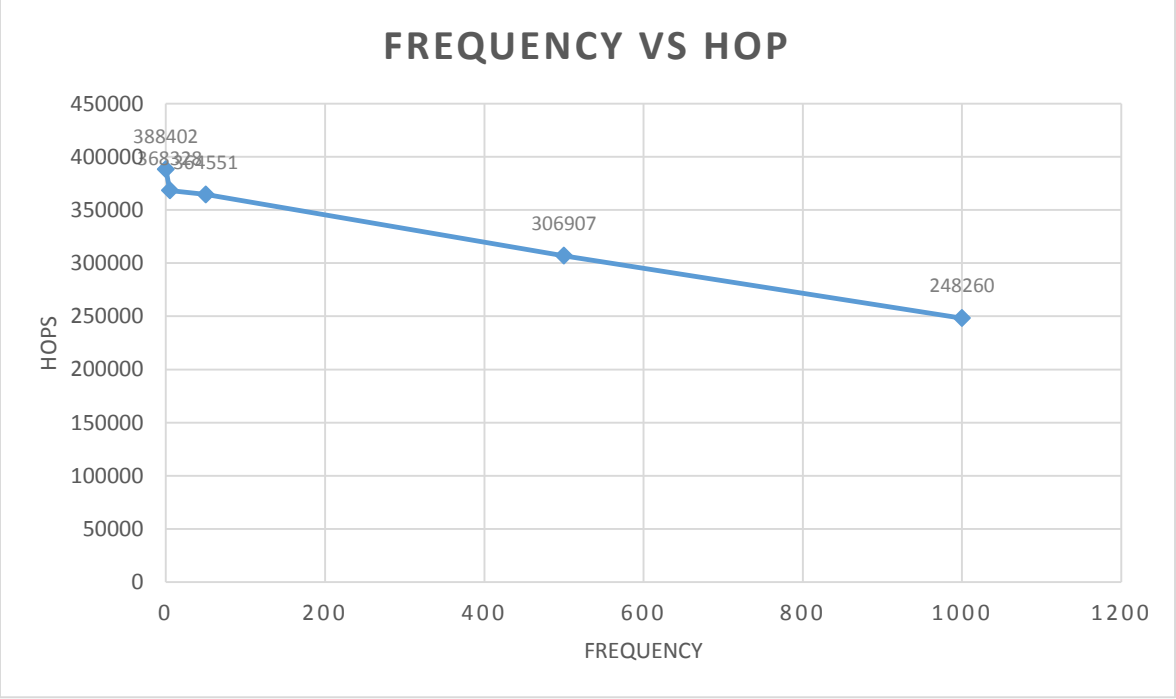
Report on what traffic loads you used; how you adjusted the Interest packet generation times & the simulation end times for these loads, and so on.

We used following information to conduct experiments for this section.

Frequency	Stop time	Simulation time	Hops
0.032	50000	55000	388402
5	322.58	500	368328
50	32.258	50	364551
500	3.225	15	306907
1000	1.6129	15	248260

Effectiveness of PIT

Here is the graph depicting the results of the above traffic loads. We can notice from the graph that as we increase frequency, PIT aggregation is increasing which is being resulted into decrease in hop count. This decrease in hop count is linear i.e hop count is being decreased linearly to increase in frequency.

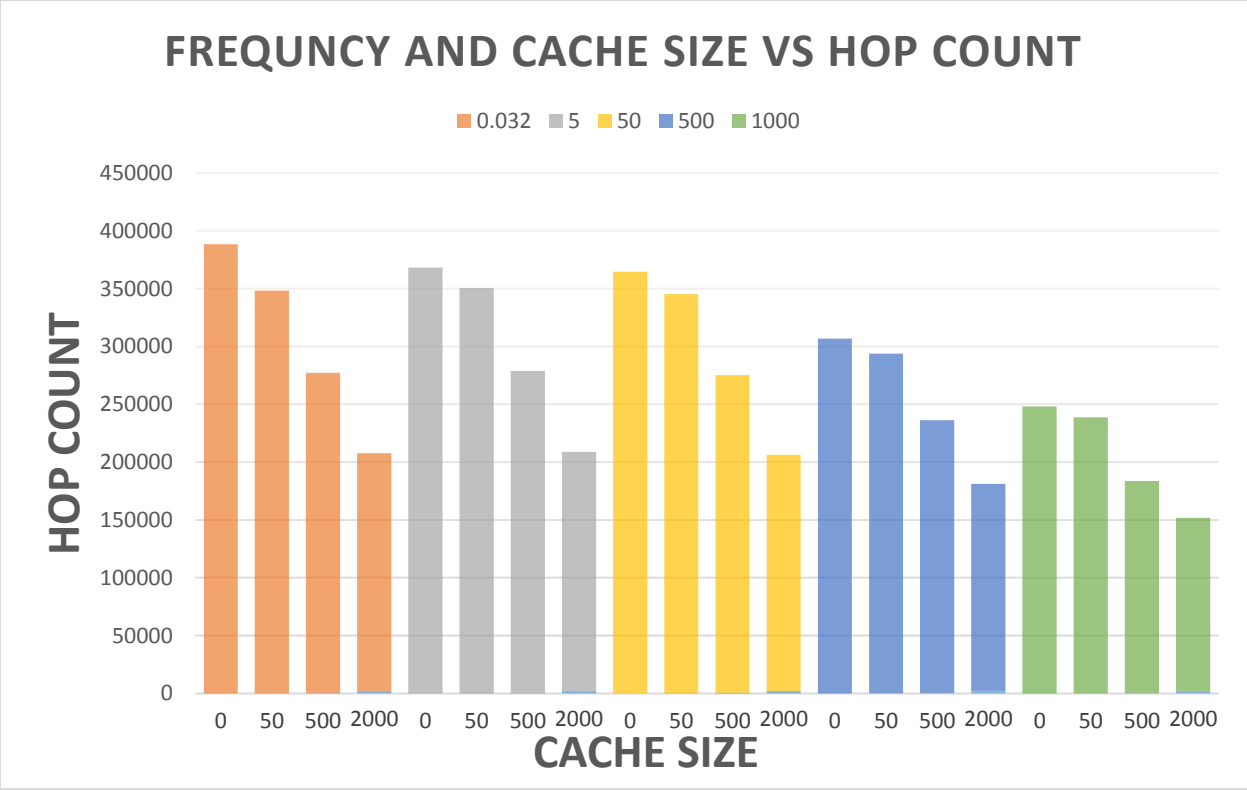


C2.

Following table displays used parameters for frequency, cache size and resulting hop counts.

Frequency	Cache size	Hop counts
0.032	50	348302
0.032	500	276790
0.032	2000	205686
5	50	350526
5	500	278488
5	2000	206812
50	50	345242
50	500	274716
50	2000	204230
500	50	293855
500	500	235779
500	2000	179089
1000	50	238791
1000	500	183043
1000	2000	149892

The graph for above data is displayed below:



We can make following observations from the above graph:

- Increase in Frequency is more effective on large values than the cache is. Initially for small changes in frequency is significantly affective to hop count. But changing frequency in large numbers is not effective. While changing cache size is initially effective but that effect gets decreased as we keep increasing cache size.
- From the graph it is clear that PIT aggregation and cache does not have additive effect on hop count i.e. introduction of both does not result in same decrease hop count for both PIT aggregation and cache.

REFERENCES:

1. Kazi & Badr 13, “Pit & Cache Dynamics in CCN”