**Inter Host Load Balancing:**

Components added :

1. Load Balancer
2. External controller

Load Balancer

1. Load Balancer is added as a node in between the eNB and the MME.  
   All eNB’s are to create a context for every attach request from the UE. All attach requests are forwarded with MME\_ID as null to the load balancer.
2. Every message received at the load balancer will be identified using the message-type field in the message.If the message type is 11 (attach request) ,the load balancer will choose one of the MME’s accessible from the load balancer and create a NAT/connection tracking entry in order to route the reply messages.The request is then sent to the chosen MME.
3. MME after receiving the attach request … processes it and sends a NAS request to the UE with the MME\_ID and a MME\_UE\_KEY. This message would be routed through the load balancer.The message type of this request is 13 and MME\_UE\_KEY and MME\_ID can be obtained from standard locations in the request.
4. After obtaining the MME\_UE\_KEY an entry in the table is made so that further requests having the same MME\_UE\_KEY can be routed to the same MME.
5. All further requests can be distinguished by the message type and the MME\_UE\_KEY can be obtained from the corresponding standard locations.

External Controller

An external controller performs the following roles.

1. Decides if states need to be migrated from one host to another.
2. Bring down a host.
3. Updating the load balancer with the modified states.

The number of hosts running every hour can be controlled using an external controller.

The controller when it sees that the existing number of hosts are being underutilized can decide to replicate states from one MME to the other MME’s and thus enabling it to bring down that MME.

**Note:** The criterion for bringing down the host depends on the when the host might be required and time required to bring up the server and install the states into it.

In order to perform migration, the controller can send some message to an MME asking it to migrate a certain number of states to another MME. Once the state gets migrated, the destination MME should update the MME\_ID for the MME\_UE\_KEY in its context and propagate this change to the controller. Then controller then modifies the table in the load balancer with the modifies MME\_ID.

Note: The MME ID need to be translated/modified for the control messages for which the MME is already brought down with the new MME ID on which the states are migrated to.

Once all the states in a host have been migrated, the controller can bring it down.

Thus further requests from the UE would be redirected to the correct MME for processing.

The above approach enables us to have the following advantages.

1. eNBs are loosely coupled with the MME’s . Hence changes to the MMEs can be performed easily as the load balancer acts as an abstraction . No messaging changes are required as modifications to the connections are handled at the load balancer.
2. As no changes are required to be propagated to the UE and the eNB, other protocols (like REST) can be used to update the MME changes to the external controller, thereby reducing the number of control messages by grouping messages.

Algorithm at the load balancer

1. Read message type from the message type header of the packet.
2. If message type is 11 (attach request) go to step 8
3. If message type is 13 go to step 11
4. Else fetch MME\_UE\_KEY from the location as specified by the message type
5. Lookup the table with the MME\_UE\_KEY For the MME\_ID.
6. Send request to the MME\_ID or eNodeB based on the message type..
7. Got to stop
8. Choose an active MME\_ID from the list of MME’s reachable from the load balancer.
9. Forward the message to the MME corresponding to the MME\_ID.
10. Goto stop
11. Fetch the MME\_UE\_ID from the packet and make an entry in the table with the key as MME\_UE\_ID and the value as MME\_ID from where the message arrived.
12. Forward the packet to the eNodeB.
13. stop

**State Migration:**

For state migration how are states selected and to which host?

The trigger will happen from the EPS Controller.

State migration

=========================

There are 2 components to it

1. Steps occurring during the migration.
2. When and what triggers the migration.

I) Steps during migration.

Migration is of the form key:imsi from: remotehost to:current

@ Current Host

1. Make a tcp connection bw current host and the remote host.
2. If TCP connection does not get established , terminate the request with some error.
3. If TCP connection is established , then send a request to the RemoteHost with the following format.

<64 bit key>

4) if Connection resets/ Times out . Return failure.

5) Parse the response sent by the server.

<64 bit key><retcode><data>

Retcode can be used to convey what happened at the server end. Having a positive value indicates things went well

<data> the entire context data.

6) Make an entry into the table with the <key> and <data>.

@remote Host

1. Accept TCP connection.
2. After connection gets established read <key>
3. Perform table data = get<key>.
4. If data = Null set retcode as -1
5. Else retcode = sizeof(Data)
6. Write back <key><retcode><data>
7. If write succeeds . delete <key>
8. If write fails, ignore the migration request.
9. Wait for the next connection.

//Comments

1. There is a possibility that after migration the current host is not able to make an entry into its table.(Refer step 6) . Need to modify the protocol to handle this scenario.
2. Hash table performance degrades after it is half full.

II) When should we trigger migration?

We should consider the following arguments before migration.

1. Does the current host have enough slots in its DB to store the newly incoming entry.
2. If we don't have enough slots, we can have 2 approaches.  
   i) Create a new table with double the slots and pop entries from the previous table and push them into the new table ,delete the old table and rename the new one to the old table. It uses a lot of extra space (2x) .Deleting and creating new tables pose a challenge of finding large continuous chunks of memory.

ii) Linear hashing: Table insertions are no longer O(1). This could impact the performance as we do a lot of get requests to the database.

c) Information regarding the available table slots must be conveyed to the controller who issues the migrate command on the current host accordingly.

Migration is performed as a part of load balancing.

Some factors to be considered from the hash table’s perspective (There are many more factors that need consideration...but this is just from the hash table’s perspective).

1. A hash table does not work well if it is loaded more than 50%. For performance reasons, it is better to keep it loaded at a maximum of ~50%.
2. In some cases the above condition could be over ridden. Eg, if a host needs to be removed, then in that case we could very well cross the 50% load on the hash table on another host.
3. Dynamic resizing of the hash table should be avoided as much as possible as it consumes a lot of time and could end up blocking requests for table access which will impact the performance of the system.

**Algorithm to choose host (Processed by the Controller):**

1. The controller needs to maintain a table containing <hosts> - < stats> for each participating hosts.
2. Controller can periodically poll for stat changes. Need to decide what the stats will be.
3. Some of the stats can be :  
   a) if a Host has very low number of Contexts send it over there.

b) Load on the Hash table. (if load is more than 50% it is better to avoid that host)

c)

4) Choose the best host from the list. Some things to consider

1. How to know the slice ID of the request (From attach request)? What if the chosen Host does not support a particular slice?

5) Once the appropriate Host is chosen , The inter host load balancer will take care of forwarding the corresponding packets.

**Algorithm to choose when to migrate states (Processed by the Controller):**

1. State migration will be required when more Hosts are added or hosts are removed.
2. We should not end up migrating the states too frequently as no activity can be performed during the duration of migration.

**Slicing using Consistent Hashing (Alternate approach to Load balancing):**

In order to use Consistent hashing for load balancing, we need to establish a unique Identifier for all messages from a UE during a session ( ie, Attach ,Service and Detach requests should have the same identifier used in all the messages) .For now let us go with the assumption that we have such an identifier (let us call it Session-ID ).

To use consistent hashing we have to use a hash function to map the Session-ID to a key space. Any reasonably performing hash function will do.

Let us define the key space as [0, 1)

Hash function being H( Session-Id) -> [0, 1)

As with consistent hashing ,The hosts need to be mapped to the same key space as well. We could use the Host’s Hostname as the key

Therefore,

H’(Hostname) -> [0, 1)

These form the basis of consistent hashing of the requests across a given set of Hosts.

By definition of consistent hashing,

All Session-Id’s such that

H’(hostname\_n0) < H(Session-Id) <= H’(Hostname\_n1)

Will map to Hostname\_n1

This approach works very well when the the number of Session-Id’s are getting evenly distributed among all the available hosts. However this is not the case always ,Hence we need to introduce the concept of virtual nodes.

Eg,

Let’s say out of 100 Session-Id’s and two hosts H1 and H2,

And by chance

80 Session-Ids map to H1 and 20 to H2, we have encountered a hotspot in H1.

Now how do we reduce the load on H1?

The load on H1 could be reduced, if there was a Host\_name\_n’ such that

H’(h’) < H’(H1) such that H’(h’) = median of of all H(session-Id’s) present in H1

Let us name H’(h’) as V2

Ie,

It is a virtual node of H2

We can make entry of V2 along with H1 and H2

Therefore the final list of hosts will be

V2 H1 H2

Where Session-Ids mapped to V2 will be stored in H2.

Thus after adding V2 the new load is as follows

H1 - 40 H2 - 60

Similarly we can add a Virtual node V1 bw H1 and H2 .

Distributing the number of Session-Ids mapped to H2 with V1.

Thus number of Session Ids mapped to H2 will become 10 and V1 as 10

Now new list of nodes will be

V2 H1 V1 H2

And number of Session Ids in H1 = 40 (from H1) + 10 (from V1)= 50

And number of session ids in H2 = 40 (from V2) + 10 (from H2) = 50

Thus by adding a large number of virtual nodes we can evenly distribute the session Ids

This comes with a downside.The downside being the lookup for each session id which is O(log(C)) The C is no longer the number of hosts but number of hosts + number of virtual nodes.Since it is of logarithmic complexity, its effect is mitigated.

Now that we have established the basic Idea of Virtual Nodes let us look at how we can incorporate Slicing into the scenario.

Assumptions made:

1. Every Session-Id belongs to some slice id.
2. Every request contains a slice id .

i) The primary idea is that each slice is a consistent hash containing Hosts and virtual nodes associated with it.

Eg,

Let us consider the following slices:

S1 - Hosts H1,H2 support S1

S2 - Hosts H1,H3 support S2

We are going to have two consistent hashes

CS1 for S1

CS2 for S2

In CS1 we are going to have nodes H1 and H2 and some virtual nodes of H1 and H2

In CS2 we are going to have nodes H1 and H3 and some virtual nodes of H1 and H3

// as an example below could be a combination of Hosts and its virtual nodes observed

CS1 = [V2 H1 V1 H2]

CS2 = [V3 H1 V1 H3]

ii) Now any request that comes to the load balancer can be routed to the corresponding Consistent hash using the Slice ID.

Eg

R(sessionid, sliceid) -> Request (session-id , sliceid)

R( 1 , 1) will go to CS1

R( 1 , 2) will go to CS2

iii) As seen earlier , with large number of virtual nodes, the load on a host gets evenly distributed within a consistent hash. However this is not a good approach when hosts are shared across multiple consistent hashes as load needs to be balanced across slices.

Eg as in the earlier case

In CS1 = 50% of the sessionid’s will go to H1 and 50% to H2 .

Similarly in CS2 = 50% of the session Id’s will go to H1 and 50% to H3 .

We can clearly see and in balance in the number of session-id’s going to H1

iv) The way we can solve the above problem is by choosing the appropriate number of sessionId’s that need to be split when we add a Node

Let s be the number of number of slices a Host is associated with ,

Instead of splitting the session id’s 50% , We could use the below formula to select the number of session ids.

[1 - s/ S] where S is the sum of all s values of all the Hosts in the Consistent Hash.

Ie,

A Host with higher s value would be able to only collect a lower number of session id’s during a split.

Eg,

Let us look at the earlier scenario

In CS1:

H1 will have s = 2 , H2 will have s = 1

Hence H1 will get [ 1 - 2/3 ] of the session Id’s = ⅓ rd

Similarly H2 will get [ 1 - ⅓] of the session id’s = ⅔ rd

Similarly in CS2

H1 will have s = 2 , H2 will have s = 1

Hence H1 will get [ 1 - 2/3 ] of the session Id’s = ⅓ rd

Similarly H3 will get [ 1 - ⅓] of the session id’s = ⅔ rd

Therefore in total all the hosts will have equal number of session Ids.