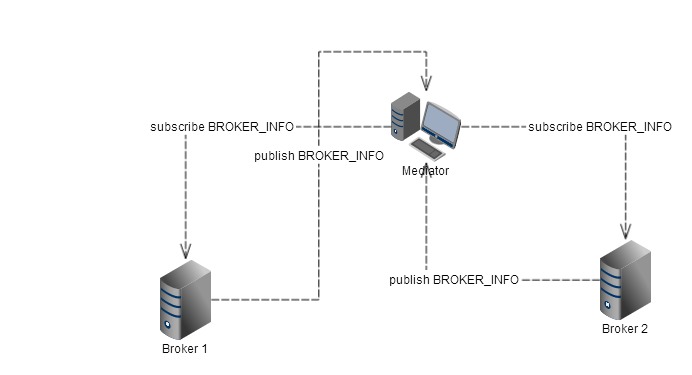
**Load detection:**

Load detection is done by a special broker in the overlay network, called the mediator. The mediator subscribes to all the brokers in the overlay network. Each broker sends BROKER\_INFO (inherent property of publishing values for and list of neighbors). This publication happens after every 20 seconds. The mediator has a defined threshold for . When the threshold is crossed, the mediator detects the broker to be overloaded. On finding the overloaded broker, Mediator does the following:

1. Initiates a new broker with the list of neighbors, same as of the overloaded broker.
2. Triggers the new broker to subscribe to the offloading broker.

The new broker subscribes to overloaded broker for topic CSStobeMigrated.



**Calculation of the root classes to migrate in CSS:**

The offloading broker creates on receiving a subscription message from the load accepting broker. The offloading broker calculates a 2D vector of length N (where N = no. of **individual root classes** in the subscription) from the list of subscriptions for that particular broker. If the broker has the following subcriptions list:

1. [class,’STOCK’],[volume,0]
2. [class,’STOCK’],[volume,10]
3. [class,’STOCK’],[volume,20]
4. [class,’SPORTS’],[type,’racing’]
5. [class,’STOCK’],[volume,100]
6. [class,’STOCK’],[volume,500]

The vector for this list would have two elements, one for the STOCK root class and the other for the SPORTS root class. This vector would be used to keep count of the number of publication messages processed by the broker in a unit of time by each of the root classes. Each time a new publication message is observed, the count for the corresponding root class is increased in the vector. At the end of the unit time, the choking root class is determined. The choking root class is the class which produces highest number of publication messages. This needs to be migrated as a topic to the load accepting broker.

Now consider a scenario when the root class with the highest publication messages contributes to only a fraction of the load on the broker. The load true cause of the load is a combined effect of two or more classes. The process described above would fail in such a scenario.

As we observed and established that this process does not create a balanced load distribution between the offloading broker and the load accepting broker. The aim is therefore to calculate the total load on the offloading broker using:

Where is the number of publication messages corresponding to the class . N represents the total number of elements in the 2D vector. This would give us the total number of publications contributing to the choking. The next task is to identify a safe amount, which when transferred to the load accepting broker, creates a balanced load distribution. We aim to distribute 50% of the current load on the broker. The problem with this aim is that not all root class publication messages would have an equally distributed contribution on the total load. Another way to put it would be, not all the participating root classes would have equal frequency of publication and thus a straight 50% offload is not possible; as breaking up of root classes for transfer is infeasible to achieve. We sort the 2D vector in descending order of publication messages for each root class. We follow the following steps to realize our goal:

The represents the sorted vector and the element is the number of publication messages of the root class. represents the next element of the following root class in the sorted class. At this point we encounter another problem, how long should we keep calculating

In an ideal setup we could do it till we hit . This would overlook an essential detail of the transfer: the last element could be sufficiently large to setback to negative. This could imply two scenarios:

1. Unequal distribution of load
2. The load accepting broker may be overloaded (with sufficiently high last element in )

We improve our previous equations and combine them to obtain:

This keeps the load balancing within desired limits. Once we have the exact number of root classes to migrate, we initiate transfers from the offloading to the load accepting broker.

**Migrate CSS from offloading to the load accepting broker:**By now we have a clear idea of the root classes to transfer to achieve load balancing. We pass a list of strings, containing the root classes. The load accepting broker has the same set of neighbors as that of the offloading broker. With the list of the root classes, the load accepting broker subscribes to each of these individually. Once subscription is successful, the load accepting proper sends s control message: DONE. This indicates that the load has been transferred, now all that remains is that the offloading broker unsubscribes to each root class subscription that has been transferred. At the end of the phase, the clients subscribed to the offloading broker gets automatically adjusted? Explicit subscription messages must be done by the client to refresh the mappings. ?

At the end of a successful transfer the load on the load accepting broker must be calculated using the same concept of our 2D vector:

We need to understand if the load distribution had been uniform. We check the value to understand if we are keeping within the threshold with the new broker. The values are monitored by the mediator in every 20s. If the mediator detects some value higher than threshold, it initiates the load balancing process.

Message convention:  
a [class,eq,<CSStoMigrate+OverloadedBrokerID>] [CSSMigrate,eq,’’] (advertisement of each broker)