The Cluster Protocol

Wittawat Tantisiriroj, Jorg Liebeherr, MNG Group

(updated: January 2008)

This is a draft and not a final version.

Table of Contents

1. Introduction

This document describes a protocol, called *Cluster protocol* or *CT protocol*, that organizes sets of nodes of an overlay network in a star topology, called a *cluster*. The node in the center of a cluster is called *cluster head* and the nodes at the periphery are called *cluster members*. Figure 1 depicts a network with three clusters. Each node is the member of only one cluster. Application data is exchanged only between nodes in the same cluster.[1]

The Cluster protocol is suitable for environments where devices have different capabilities with respect to power, data rate, computing resources, or have different roles in an application scenario. For example, in a sensor network, low-power sensor nodes may form clusters with a higher-powered gateway device in

[1] There are additional protocols, e.g., the Backbone-Cluster protocol, where cluster heads join another (backbone) overlay topology. The backbone network enables the exchange of application data between nodes in different clusters.

as the cluster head. In peer-to-peer streaming application, client applications may form a cluster with a content delivery server becoming a cluster head. The design goal of the CT protocol is to provide a protocol solution that is suitable for a wide variety of networking scenarios where cluster formation can be exploited. The CT protocol can run in wired and wireless networks. The range of applications that the protocol is trying to support ranges from peer-to-peer networks over the Internet to sensor networks. The criteria for the formation of clusters can consider geographical proximity, data rate, and user defined metrics.

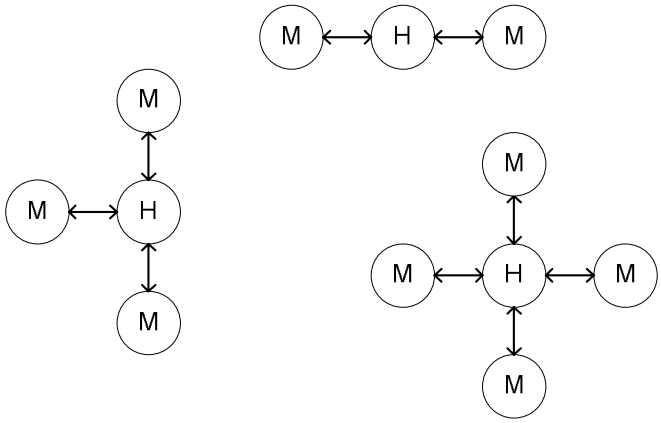


Figure 1. Cluster topology. (Cluster heads and members are labeled with H and M, respectively.)

The configuration of a node determines whether a node plays the role of a cluster head or a cluster member. Some nodes can become either a cluster head or a cluster member, although not at the same time. These nodes are called *hybrid nodes*. Whenever possible, a hybrid node runs as a cluster member. In situations where some cluster members cannot find a head, the hybrid switches its role and runs as a cluster head. Hybrid nodes balance the availability of cluster heads. If the density of a cluster heads is high, most hybrid nodes run as cluster members and utilize the resources made available by cluster heads. If the density of cluster heads is low and cluster members cannot find a cluster head, hybrid nodes switch can help with the creation of new clusters.

The goal of the Cluster Protocol is to match each cluster member with one cluster head. In a network with hybrid nodes, a second goal is to minimize the total number of clusters. Each cluster head makes available a limited set of resources to its cluster members. If these resources are exhausted, a cluster head does not accept new cluster members. Cluster members have preferences and restrictions on the properties of cluster heads that they connect to, in terms of distance to the cluster head, available bandwidth, and others. The protocol matches cluster members to a cluster heads, such that all requirements of cluster members are met, and preferences are accommodated as much as possible.

2. Overview of the Protocol

The clusters formed by the Cluster Protocol consist of logical connections between a cluster head and its cluster members. The formation of a cluster is driven by cluster members, who search for available cluster heads and request membership in a cluster. Cluster members operate independently of other cluster members when searching for a cluster head.

The logical address of a cluster member or head is a randomly selected number. The logical addresses in a cluster must be unique. If a node detects a duplicate address, i.e., some other node has the same logical address, it must leave the overlay network, select a new logical address, and re-join the network.

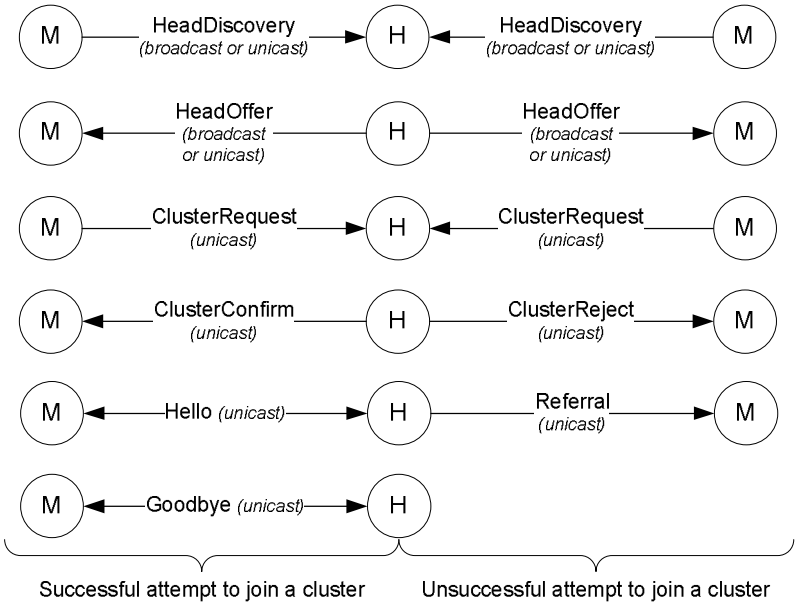


Figure 2. Basic interaction in Cluster Protocol.

Cluster heads and members exchange protocol messages over a substrate network by broadcast or unicast transmissions. The Cluster Protocol uses broadcast operations[2] if they are available, but it does not require them. The main interactions of the Cluster Protocol are shown in Figure 2.

We can think of the interactions of a member with a head as occurring in two phases. In the first phase, a cluster member that does not have information about any head attempts to locate a cluster head. This phase is referred to as rendezvous process. In the second phase, a cluster member joins and maintains connectivity with a cluster head.

A cluster member in the rendezvous phase tries to identify cluster heads by sending a *HeadDiscovery* message. Information about possible cluster heads can be obtained from a configuration file or a cache file.[3] A cluster member contacts each cluster head in these files by sending a *HeadDiscovery* message via unicast. In addition, if the substrate network supports broadcast transmissions, a cluster member broadcasts a *HeadDiscovery* message. A cluster head that receives a *HeadDiscovery* message responds with a *HeadOffer* message, which contains information about the cluster head. The *HeadOffer* message is transmitted by unicast. A cluster head sends a *HeadOffer message* even if it does not have room for an additional member. When a cluster member receives a *HeadOffer* it enters the second phase of

[2] The scope of the broadcast messages can be limited. In ad-hoc wireless network the scope of a broadcast transmission is limited by the transmission range of the sending node. If the substrate network is an IP network, the scope can be limited using a local broadcast address, or by specifying a limited scope in the TTL field of IP multicast messages.

[3] The configuration file is the XML file for the configuration of a HyperCast overlay socket. The cache file is created by a cluster member that leaves an overlay network. It contains the addresses of the clusters in the head cache of the member.

interactions, where requests to join a cluster head by sending a *ClusterRequest* message. If the head can accommodate a new member, it responds with a *ClusterConfirm* message. The *ClusterConfirm* message confirms the logical link between a cluster head and member. Head and member are now neighbors in a cluster. Once the logical link is established, both the cluster member and cluster head send to each other periodically *Hello* messages that confirm the member-head relationship. If a cluster head cannot take a new member it sends a *ClusterReject* message followed by a *HeadReferral* message. The *ClusterReject* message declines the request to join the cluster. The *HeadReferral* contains a list of other cluster heads that can be contacted by the cluster member. The rejection and the referral messages are separated since referrals are used in other contexts, i.e., whenever heads wants to disseminate information about other cluster heads.

When a node departs from a cluster, it sends a *Goodbye* message to its neighbors. A cluster member sends a *Goodbye* message to its head, and a cluster head sends a *Goodbye* message to all its members. After sending a Goodbye message, a cluster member responds to messages from the cluster head for some time. Whenever the node receives a message from the cluster, responds with a *Goodbye* message. This makes sure that the cluster head receives a *Goodbye* message.

If a node has not received a *Hello* message for a long time from a neighbor, it assumes that the neighbor is no longer present. Eventually, state information about this neighbor is removed. Therefore, when a node fails and simply stops transmitting messages, state information about this node will eventually expire.

Both cluster members and cluster heads keep a list of cluster heads, called the *head cache*, and use incoming protocol messages to maintain the cache. Members use the cache to identify a new cluster head. Cluster heads use the list when they send referrals to other nodes. Entries in the head cache have the following form (*Address*, *HeadInfo*, *Age*), where *Address* is the physical address of the cluster head, *HeadInfo* is state information about the cluster head, and *Age* is the elapsed time since the entry was created or last updated. Each cluster head keeps an entry on its own state in the top position of the cache. Later, we describe the state information in *HeadInfo* in detail and how it is used by cluster members. Entries in the cache are deleted if they have reached a maximum age. Generally, the size of the data cache will be limited. When the cache is full the oldest entry is preempted. There are several ways in which entries are added to the cache. When a node is created, the cache is initialized with information from a configuration file. After that, the cache is updated using the content of incoming *HeadOffer* and *HeadReferral* messages.

When a cluster head receives a *HeadReferral* (or *HeadOffer*) message, it updates its cache with the information contained in the message. When a cluster head sends a *HeadOffer* message it includes the Head information about itself. When a cluster head sends a *HeadReferral*, it includes the head cache information. Each cluster head periodically transmits *HeadReferrals* to all cluster heads in its head cache and to all members.[4] In this fashion, information about cluster heads is disseminated in the same cluster and across multiple clusters.

Remarks:

- A member broadcast *HeadDiscovery* message only if it does not have a head satisfying its criteria. A member can send a *HeadDiscovery* as a unicast message if it wants to acquire information about a specific head, e.g., it wants to get a logical address of a head.
- When a cluster member loses its cluster head and rejoins the cluster, it does not immediately join the rendezvous phase by sending a discovery message. Instead, the member contacts each cluster head in the referral list and attempts to join the corresponding cluster.

[4] A referral message is sent to all cluster members every *MemberReferralInterval* seconds and to the cluster heads in the head cache every *HeadCacheReferralInterval* milliseconds.

- When a node leaves an overlay, the physical addresses of the cluster heads in the head cache is saved in a cache file. When the node joins the overlay again, it loads a list of heads from the cache file. Note that the cache file only contains the physical address, but not the information in *HeadInfo*. A node that reads the cache file sends unicast *HeadDiscovery* messages to obtain HeadInfo inforation and the logical address.
- Since *Referral* messages can be significantly longer than all other messages, the protocol attempts to send referrals only when necessary. For example, one could conceive of a variation of the protocol where a cluster head transmits a referral in response to a *HeadDiscovery* message. However, since a *HeadDiscovery* message may be broadcast, this could result in many heads sending referrals, possibly overwhelming the member that sent the *HeadDiscovery* message.

2.1. Head Information and Head Selection

In the Cluster Protocol, it is the cluster members that determine which cluster head to join and when to join it. (An alternative approach would be to leave the management of a cluster to the cluster heads, and have cluster heads issue invitations to potential cluster members.) A cluster member may decide to switch to a different cluster head if the new head satisfies the needs of the member better than the current head. Each cluster head is responsible for maintaining its own head information. The information is initialized from configuration information, and can be updated dynamically.[5] A member uses the *HeadInfo* information about cluster heads in the head cache to decide when to contact a cluster head.

The *HeadInfo* information about a cluster includes:

- **Available members** – The number of members that the cluster head will accept.
- **Location** – The geographical location of the cluster head in a specified coordinate system.
- **Rate** – The data rate that this cluster head offers to each member in kilobits per second;
- **Metric** – An application–defined metric value that specifies a property about the cluster head. The metric can be used to encode information about hardware properties, roles in application scenarios, or the availability of resources. The metric is such that higher value of the metric are considered `better'.

A member node checks the *HeadInfo* information of a node against a set of criteria from its configuration file. The criteria are as follows:

- *MinimumAvailableMember (Default: 1)*: The number of available slots advertised by the cluster head must exceed the number of this parameter. The minimum value is 1.
- *MaxDistance (Default: 100)*: The distance between the advertised location of the head and the current location of the member must not exceed the given maximum distance. A negative value indicates that no limit is specified.
- MinimumRate (Default: 0): The advertised data rate must exceed the minimum rate value.
- MinimumValue (Default: 9): The value gives the smallest acceptable value of the metric.

The Clustering Protocol supports two policies for selecting a cluster head, called *NextFit* and *BestFit*. With *NextFit*, a cluster member can join any cluster head that satisfies the above criteria. With *BestFit*, a cluster member attempts to find the best cluster head with respect to specified criteria. A node can select one of the following criteria: Location, Rate, and Metric. By selecting Location, a cluster member tries to find the cluster head that is closest to its current position. With the Rate criteria, a cluster member tries to join the cluster head that offers the maximum data rate. If Metric is specified, a cluster member seeks a cluster head that maximizes the cluster metric. To prevent that cluster members change their cluster

[5] The head information is a mutable configuration attribute, i.e., a configuration parameter that can be modified after the creation of the overlay node. Mutable attributes must be verified each time they are accessed in the system. (Note: mutable statistics are currently not widely implemented. Another example is the group key attribute in the security architecture).

head too often, the new cluster head must improve the distance, the data rate, or the metric value by more than a given threshold value.

When a node contacts a new cluster head, it selects a head from the head cache. The rules for selecting a cluster head from the cache are as follows. For each head cache entry, a member records the number of attempts that this node has been contacted with a *ClusterRequest* message (The value is reset to 0 if a member receive Cluster Confirm message from this node). An entry is ignored if the number of attempts to contact this node exceeds *HeadTimeout/HeartbeatTime* (default is 3). A node first identifies an entry with the smallest number of attempts. If more than one entry remains, a node discards entries in the order of preference. The lowest preference is for a hybrid node running as a member. Next in the preference order is a hybrid node running as a head. A cluster head has the highest preference. If selection policy is *NextFit*, the node selects any of the remaining entries. If selection policy is *BestFit*, the node selects the best cluster head among the remaining entries with respect to the specified criteria.

The decisions to join or change a cluster are completely left to the cluster members. A cluster head should accept any request to join the cluster that can be accommodated. Also, cluster heads cannot selectively drop cluster members.

2.2. Hybrid nodes

Hybrid nodes (or hybrids) can operate as either cluster members or cluster heads. Whenever possible, a hybrid node will run as a cluster member and join a cluster head. When all nodes are hybrid nodes, the cluster protocol will result in a small number of cluster heads. Ideally, the number of clusters is minimal in the sense that there is no other assignment of hybrids into member and heads that has fewer cluster heads.[6]

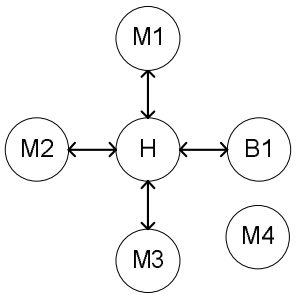
The default state of a hybrid node is that it runs as a cluster head. When a hybrid node is initialized, it starts out as a cluster head. Also, if a hybrid node cannot communicate with any other node, it will eventually become a cluster head. If a hybrid node in cluster head mode that does not have members, and receives a *HeadOffer* from some cluster head, it switches to member mode and attempts to join the sender of the *HeadOffer* as a cluster member. Likewise, if a hybrid node (either a head or a member) receives a *ClusterRequest* message, it must accept this member and remain a cluster head until all members have left the cluster.

Note: A hybrid node running as a member can be forced to switch to a cluster head
Suppose there is one head called H1 which can accept four members and one hybrid called B1 which can also accept five members. First, all members and the hybrid node become members of H1. When a new member, say M4, joins the network, M4 can only find a cluster head if B1 switches to a head.
M4 receives head offer from H and B1.....H4 cannot accept more members .... So: M4 contacts B1 ....
Once it receives request B1 switches to Head

Comment: should M4 be able to force B1 to switch to head?

Comment: Yes, otherwise M4 will be disconnected.
This feature is implemented as of Oct 30, 2006.

[6] This problem can be related to the minimum cover set problem in computer science theory.



- A hybrid node sends *HeadOffers*, only if it is in one of the following states: *Member, Head Without Member*, and *Head With Member*.
- A hybrid node sends *Referrals*, only when it is in one of the following states: *Head Without Member* and *Head With Member.*

When multiple hybrid nodes in cluster head mode can communicate directly with each other, some of the nodes will be permitted to run as members, while others must remain cluster heads. Without additional mechanisms it is feasible that hybrid nodes flip back and forth between a member and a cluster head modes. Since it is desirable that the nodes quickly converge to an agreement, the cluster protocol implements a *backoff* mechanism, where nodes defer the transmission of *HeadOffer* messages.

When a hybrid node running as a head receives a *HeadOffer* message from another hybrid node that is also running as a head within *OfferCollisionWindow* milliseconds after it sent its own *HeadOffer* message, it ignores the *HeadOffer* message, sets a variable *OfferCollisionCounter* to one. Now, the hybrid node randomly schedules the next transmission time of the *HeadOffer* message. If T was the time its last *HeadOffer* transmission, the next HeadOffer transmission is scheduled at a random time in the interval

$$[T+ OfferCollisionWindow,\ T+ OfferCollisionWindow + 2^{OfferCollisionCounter} \times HeartbeatTime].$$

If there is another *HeadOffer* received within *OfferCollisionWindow* milliseconds after the transmission, the *OfferCollisionWindow* is incremented and another *HeadOffer* transmission is scheduled. A *HeadOffer* is viewed as successful if a node does not receive a *Headoffer* during a collision window after the transmission of an *HeadOffer*. In this case, the *OfferCollisionCounter* is reset to zero.

Each hybrid node must periodically sent *HeadOffer* and *Referral* messages even if it is running as a cluster member. In this way, information about the hybrid nodes is disseminated. The *HeadInfo* information included in these messages identifies whether a node is a hybrid node or a head node. For hybrid nodes, the information also specifies if this node is currently in member mode or in cluster head mode.

3. States and State Transitions

In this section, we specify the finite state machines of the cluster protocol. Table 1 summarizes the states of the protocol.

Table 1. State description.

State Name	State Definition
Stopped	The node is not running

Member Candidate Without Head	A cluster member in a rendezvous phase. The node transmits *HeadDiscovery* messages, and listens to *HeadOffer* messages.
Member Candidate With Head	A cluster member that has knowledge of cluster heads, but is not a member of a cluster. A node in this state attempts to join a cluster by sending *ClusterRequest* messages.
Member	A cluster member that is a member of a cluster head. The node exchanges *Hello* messages with its cluster head.
Head Without Member	The node is a cluster head without a member.
Head With Member	The node is a cluster head with at least one member.

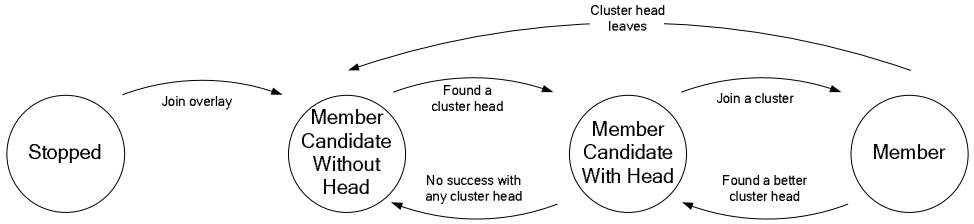


Figure 3. State transition diagram of a member.
(From each state, there is an additional edge with label Leave overlay to state Stopped.)

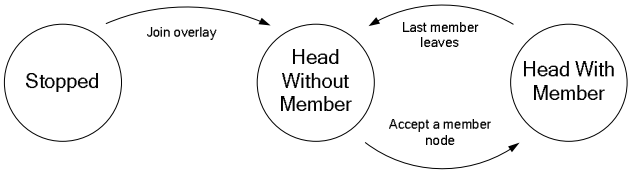


Figure 4. State transition diagram of a cluster head.
(From each state, there is an additional edge with label Leave overlay to state Stopped.)

Figures 3 and 4 show the state transition diagram for member nodes and cluster nodes. The figures do not show that a node can return to state *Stopped* from any state by leaving the overlay network. As shown in Figure 3, a new member node initially becomes a member candidate. A candidate tries to learn about new cluster heads and build up information in the head cache. Initially, a candidate does not have a head (*Member Candidate without Head).* Here, the node issues *HeadDiscovery* messages. Once a node learns about cluster heads by receiving *HeadOffer* or *Referral* messages, it becomes a *Member Candidate with Head.* This state, the node goes through its head cache and makes attempts to join one of the heads. When a head rejects a request, it will also send a referral message, which may add new information to the head cache. However, if the head cache is exhausted and no head has accepted the new member, the node must return to *Member Candidate without Head* and find new cluster heads. If a cluster head responds to a *ClusterRequest* with a *ClusterConfirm* message, the node is now a member of

a cluster. The node will remain in this state unless it finds a better cluster head, the cluster head leaves, or the application terminates.

The state transition diagram of a cluster head, shown in Figure 4, is quite simple. It merely distinguishes between a head that has no members and a head that has members. Even these two states could be summarized, as the behavior of a cluster node is identical in both states. (We distinguish the states since hybrid nodes behave differently in this state).

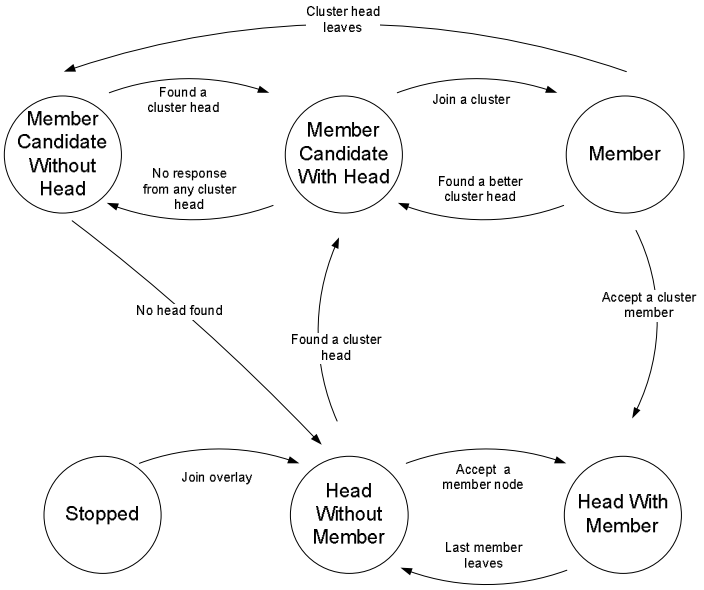


Figure 5. State transition diagram of a hybrid.
(From each state, there is an additional edge with label Leave overlay to state Stopped.)

Figure 5 depicts the state transition diagram for hybrid nodes. The diagram can be viewed as a superposition of the diagrams for members and nodes, with a few crucial differences. First, when a node starts, it initially starts out as a head. When a hybrid node is a head, but does not have a member, it can switch to the member mode, when it learns about a head and can join the head. Note that this is not possible after a hybrid node has accepted a member.

- A change from Head Without Member to Member Candidate With Head happens when a hybrid node found a head.
- A change from Member to Head With Member occurs when a hybrid running as a member receives a *ClusterRequest* and subsequently accepts a member. Note that a hybrid node in member mode behaves differently than a member node, i.e., it sends *HeadOffer* messages and it responds to *ClusterRequest* messages.
- When a hybrid in member node and receives a *ClusterRequest* it can accept the request and switch into the mode of a cluster head. It is not allowed that a Member Candidate accepts a member. This avoids a situation where hybrid nodes flip back and forth between a member and a cluster head modes.

Comment: Add example to show the situation where hybrid nodes flip back and forth between a member and a cluster head modes

Note that a hybrid node never stays in state *Member candidate without head* . A hybrid node will become a *Head Without Member* immediately after it becomes *Member candidate without head*. In other words, for hybrid nodes, the state Member candidate without head could be deleted. We maintain the state for better comparison with the other state diagrams.

Need a description of the timers

State: **Any**

Event	*Action*
Receives a message from a node *m* with a source logical address identical to its own logical address	Randomly select a new logical address and send a Rejoin message to *m*
Rejoin message received from *m*	Randomly select a new logical address
Heartbeat Timer expires	Remove an expired entry in a head cache. For each entry in a head cache If an head information field for a head *h* is missing because a head *h* is added from a cache file or a configuration file Send a HeadDiscovery message to the head *h*

Comment: Sync from code. There is no *CacheEntryTimeout* timer, but the expired entries are removed every Heartbeat.

State: **Stopped**

Event	Action
Join Overlay	Load head cache from a cache file and the configuration file If NodeType == Member → **Member Candidate Without Head** else → **Head Without Member**

State: **Member Candidate Without Head**

Event	Action
Initial tasks	-
Heartbeat Timer expires	If NodeType == Hybrid → **Head Without Member** If there is a head entry in head cache Select a head in head cache as a new head based on preferences discussed in Section 2.1 → **Member Candidate With Head** Else Broadcast *HeadDiscovery* message *m* to all nodes if broadcast operation is avaiable.
HeadOffer message received from h	Update information and timestamp of the head entry of h in the head cache
HeadReferral message received from h	Update information and timestamp of the head cache with all entries in the Referral list
Hello message received from *h*	Send *Goodbye* message to h
ClusterRequest message received from *m*	Send *Goodbye* message to m
Leave Group	→ **Stopped**

State: **Member Candidate With Head**

Event	Action
Initial tasks	-

Heartbeat Timer expires	If current head is expired Select a head *h* in the head cache as a head candidate If no head found If My Mode = Hybrid → **Head Without Member** Else → **Member Candidate Without Head** Send *ClusterRequest* message *m* to the head *h* Increment attempt counter of the entry of head *h* in the head cache
HeadOffer message received from h	Update information and timestamp of the head entry of h in the head cache
HeadReferral message received from h	Update information and timestamp of the head cache with all entries in the Referral list
ClusterConfirm message received from h	If h is its current head → **Member** Else Send *Goodbye* message to m
ClusterReject message received from h	Set Attempts field of the entry of head *h* in a head cache to (HeadTimeout/HeartbeatTime)+1 (default: 4). → **Member Candidate Without Head**
ClusterRequest message received from *m*	Send *Goodbye* message to m
Hello message received from *h*	If h is not its current head Send *Goodbye* message to h
Goodbye message received from h	Remove h from a head cache If h is its current head → **Member Candidate Without Head**
Leave Group	→ **Stopped**

Comment: Sync from code. Send Goodbye message unless h is its current head

State: **Member**

Event	Action
Initial tasks	Set *HeadTimeout* Timer
Heartbeat Timer expires	If SelectionPolicy is BestFit Select a best head *h* in head cache as a head If *h* is better than current head → **Member Candidate With Head** If NodeType == Hybrid Broadcast *HeadOffer* message *m* to all nodes If its current head is expired Set Attempts field of the entry of head *h* in a head cache to (HeadTimeout/HeartbeatTime)+1 (default: 4). → **Member Candidate Without Head**
HeadOffer message received from h	Update information and timestamp of the head entry of h in the head cache
HeadReferral message received from h	Update information and timestamp of the head cache with all entries in the Referral list
ClusterRequest message received from *m*	If NodeType == Hybrid Send *ClusterComfirm* message to m → **Head With Members**

Comment: Sync from code. There is no *HeadTimeout* timer, but the expired head is removed every Heartbeat.

Comment: Even in BestFit policy, a member will not switch head until next Heartbeat time

Comment: Same as above

	Else Send *Goodbye* message to m
Hello message received from h	If h is its current head Reset *HeadTimeout* Timer Send *Hello* Message to h Else Send *Goodbye* message to m
Goodbye message received from h	Remove h from a head cache If h is its current head → **Member Candidate Without Head**
Leave Group	Send *Goodbye* Message to it current head → **Stopped**

Comment: A member will send a goodbye to an invalid head

State: **Head Without Member**

Event	Action
HeadOffer message received from h	If MyMode == Head or h.NodeType != HybridMember Update information and timestamp of the head entry of h in a head cache Else If h.NodeType == Head → **Member Candidate With Head** Else if h is Hybrid (Head) If ListeningPeriod Timer is not set → **Member Candidate With Head** Else Set a Collision Flag to true
HeadReferral message received from *h*	If MyMode == Head or h.NodeType != HybridMember Update information and timestamp of the head cache with all entries in the Referral list Else If NodeType of any head in Referral is Head → **Member Candidate With Head** Else if h is Hybrid (Head) If ListeningPeriod Timer is not set → **Member Candidate With Head** Else Set a Collision Flag to true **(for Jorg: recheck once the other pieces are clarified.)**
Leave Group	→ **Stopped**

Comment: Needs to be defined? What happens when the conflict flag is set?

Comment: See ListeningPeriod Timer expires.

Comment: Need to work on it. The part is quite simple, so we should be able to describe it simpler than what we have now.

State: **Head With Member**

Event	Action
HeadOffer message received from h	Update information and timestamp of the head entry of h in a head cache
HeadReferral message received from *h*	Update information and timestamp of the head cache with all entries in the Referral list
Leave Overlay	Send *Goodbye* message to all of its member → **Stopped**

State: **Head Without Member, Head With Member**

Event	Action

Heartbeat Timer expires	Remove an expired member If its Neighborhood table is empty → **Head Without Member** Send *Hello* message to all members If NodeType == Head Broadcast *HeadOffer* message *m* to all nodes Else If no *HeadOfferSend* Timer is scheduled Set a *HeadOfferSend* Timer to expire after a randomly selected time between 0 to HeadOfferPeriod ms	**Comment:** Sync from code. There is no *WaitingForMember* timer, but the expired entries are removed every Heartbeat.
HeadOfferSend Timer expires	Set the Collision Flag to false Broadcast *HeadOffer* message *m* to all nodes Set a ListeningPeriod Timer to expire after *OfferCollisionWindow* ms	**Comment:** Will be clarified with other comments.
ListeningPeriod Timer expires	If the Collision Flag is false Reset HeadOfferPeriod to HeartbeatTime Else Double HeadOfferPeriod Set a HeadOfferSend Timer to expire after a randomly selected time between 0 to HeadOfferPeriod ms	**Comment:** See above
HeadInfoExchange Timer expires	Send *HeadReferral* message to all of heads in its head cache.	
ReferralPush Timer expires	Send *HeadReferral* message to all of its members	
Hello message received from m	If m is its member Update timestamp of member m Else Send *Goodbye* Message to m	
Goodbye message received from *m*	Delete m from member table	
HeadDiscovery message received from *m*	If it can accept a new member Send *HeadOffer* message to m Else Send a small size *HeadReferral* message to m	**Comment:** If LimitedReferralSize is > 0, a full head will send a referral message with LimitedReferralSize entry when it receives a HeadDiscovery message
ClusterRequest message received from *m*	If detect duplication of logical address Send *Rejoin* message to m If it can accept a new member Send *ClusterConfirm* message to m If State is Head Without Member → **Head With Member** Else Send *ClusterReject* message to m Send *HeadReferral* message to m	

4. Attributes

4.1. Example of a member node configuration

```
<Node>
    <CT>
        <HeartbeatTime>1000</HeartbeatTime>
```

```
            <MemberTimeout>3000</MemberTimeout>
            <HeadTimeout>3000</HeadTimeout>
            <MemberReferralInterval>5000</MemberReferralInterval>
            <HeadCacheReferralInterval>1000</HeadCacheReferralInterval>
            <CacheEntryTimeout>10000</CacheEntryTimeout>
            <OfferCollisionWindow>500</OfferCollisionWindow>
            <Verification>neighborcheck</Verification>
            <StatName>Node</StatName>
            <ReferralEnable>true</ReferralEnable>
            <LimitedReferralSize>1</LimitedReferralSize>
            <CacheFile>.Cachefile</CacheFile>
            <HeadCacheSize>10</HeadCacheSize>
            <HeadNum>1</HeadNum>
            <Head>
                <UnderlayAddress>
                    <INETV4AndOnePort>127.0.0.1:9800</INETV4AndOnePort>
                </UnderlayAddress>
            </Head>
            <NodeType>
                <Member>
                    <Criteria>
                        <Member>
                            <MinimumAvailableMember>1</MinimumAvailableMember>
                        </Member>
                        <Location>
                            <Coordinate>500,500</Coordinate>
                            <MaxDistance>100</MaxDistance>
                        </Location>
                        <Bandwidth>
                            <MinimumRate>0</MinimumRate>
                        </Bandwidth>
                        <Metric>
                            <MinimumValue>9</MinimumValue>
                        </Metric>
                    </Criteria>
                    <SelectionPolicy>
                        <NextFit/>
                    </SelectionPolicy>
                </Member>
            </NodeType>
        </CT>
    </Node>
```

Comment: See section 7. timers for the details

Comment: See October 25, discussion
Where does coordinate belong to?

Various Selection Policies Example

1) Next Fit

```
                    <SelectionPolicy>
                        <NextFit/>
                    </SelectionPolicy>
```

2) Location

```
                    <SelectionPolicy>
                        <BestFit>
                            <Location>
```

```
                    <ThresholdDistance>5</ThresholdDistance>
                </Location>
            </BestFit>
        </SelectionPolicy>
```

3) Bandwidth

```
        <SelectionPolicy>
            <BestFit>
                <Bandwidth>
                    <ThresholdRate>10</ThresholdRate>
                </Bandwidth>
            </BestFit>
        </SelectionPolicy>
```

4) Metric

```
        <SelectionPolicy>
            <BestFit>
                <Metric>
                    <ThresholdValue>1</ThresholdValue>
                </Metric>
            </BestFit>
        </SelectionPolicy>
```

4.2. Example of a head node configuration

```
<Node>
    <CT>
        <HeartbeatTime>1000</HeartbeatTime>
        <MemberTimeout>3000</MemberTimeout>
        <HeadTimeout>3000</HeadTimeout>
        <MemberReferralInterval>5000</MemberReferralInterval>
        <HeadCacheReferralInterval>1000</HeadCacheReferralInterval>
        <CacheEntryTimeout>10000</CacheEntryTimeout>
        <OfferCollisionWindow>500</OfferCollisionWindow>
        <Verification>neighborcheck</Verification>
        <StatName>Node</StatName>
        <ReferralEnable>true</ReferralEnable>
        <LimitedReferralSize>1</LimitedReferralSize>
        <CacheFile>.Cachefile</CacheFile>
        <HeadCacheSize>10</HeadCacheSize>
        <HeadNum>1</HeadNum>
        <Head>
            <UnderlayAddress>
                <INETV4AndOnePort>127.0.0.1:9800</INETV4AndOnePort>
            </UnderlayAddress>
        </Head>
        <NodeType>
            <Head>
                <Criteria>
                    <Member>
                        <MaximumMember>20</MaximumMember>
                    </Member>
```

Comment:
If ReferralEnable is not true, all referral messages are disable.
If LimitedReferralSize is > 0, a full head will send a referral message with LimitedReferralSize entry when it receives a HeadDiscovery message

```
                    <Location>
                        <Coordinate>500,500</Coordinate>
                    </Location>
                    <Bandwidth>
                        <OfferRate>56</OfferRate>
                    </Bandwidth>
                    <Metric>
                        <OfferValue>9</OfferValue>
                    </Metric>
                </Criteria>
            </Head>
        </NodeType>
    </CT>
</Node>
```

Note:
The distance between two points is approximately calculated by spherical law of cosines under an assumption that the earth is a perfect sphere. Distance = Radius of sphere * arcos [sin(latitude1) * sin(latitude2) + cos(latitude1) * cos(latitude2) * cos(longitude2 - longitude 1)]. In this implementation, Radius of the earth is approximated to 3963.0 miles. The unit of distance is mile and the unit of latitude and longitude are radians.

5. Tables

5.1. HeadCache Table

Physical Address	Logical Address	Node Type	Timestamp	Available Members	Current Members	Location	Offered Rate	Metric	Attempts
...	...	...	...			...	...	...	

Physical Address: variable size (PASize), the physical address of a cluster head node.
Logical Address: variable size (LASize), the logical address of a cluster head node.
Head Node Type: byte = 1 byte, the Node Types of head are either Head [1], Hybrid (Head) [2], or Hybrid (Member) [3]
Timestamp: long = 8 bytes, the last time this entry has been updated.
Available Member: int = 4 byte, the number of members that this cluster head node can accept currently.
Location: (float, float) = 8 bytes, the location of this cluster head node in a specified format.
Offered rate: int = 4 bytes, the data rate that this cluster head offers to each member in kilobits per second (kbps).
Metric: byte = 1 bytes, the value representing a user defined metric of this cluster head node.
Attempts: byte = 1 byte, the number of attempts to request from this cluster head and it is set to 0 if the request is accepted.

Comment: Jorg, please check whether you understand it or not

5.2. Neighborhood Table

Logical Address	Physical Address	Timestamp
...	...	...

Logical Address: variable size (LASize), the logical address of a neighbor.
Physical Address: variable size (PASize), the physical address of a neighbor.
Timestamp: long = 8 bytes, the last time this entry has been updated.

6. Message Format

This section list the detailed message formats used in the Clustering Protocol. The common format for all protocol messages is shown in Figure 6.

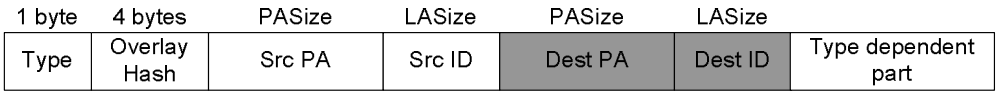


Figure 6. Format of Cluster protocol messages.

Type: The types of Clustering Protocol message are:

Table 2. Protocol Message Types.

Message NodeType	NodeType Field
HeadDiscovery	0
HeadOffer	1
ClusterRequest	2
ClusterConfirm	3
ClusterReject	4
Hello	5
Goodbye	6
HeadReferral	7
Rejoin	8

Overlay Hash: A 4-byte long hash value which is derived from the values of all attributes specified in *HashAttributes* of the configuration file.
Src PA: The physical address of the sender of this message
Src ID: The logical address of the sender of this message
Dest PA: The physical address of the destination of this message
Dest ID: The logical address of the destination of this message

Dest PA and Dest ID fields are omitted in the rendezvous messages *HeadDiscovery* and *HeadOffer*.

6.1. HeadDiscovery Message

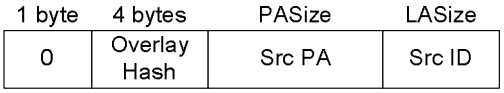


Figure 7. HeadDiscovery message.

The HeadDiscovery Message is sent by a cluster member to request the information from existent cluster heads. The message is transmitted by broadcast or unicast, where the address is obtained from the configuration file, from attributes HeadNum and Head attributes.

6.2. HeadOffer Message

1 byte	4 bytes	PASize	LASize	InfoSize
1	Overlay Hash	Head PA	Src ID	HeadInfo

Figure 8. HeadOffer message.

The *HeadOffer* Message is sent by a cluster head to provide its information, i.e., NodeType, Location, Rate, and Metric, etc., to both cluster members and cluster head. A cluster head sends a HeadOffer periodically and in response to a *HeadDiscovery* message. Also, hybrid nodes (even if they are not running as cluster heads) issue periodic *HeadOffers*.

HeadInfo: Contains information about a cluster head. The field is discussed below in Section **6.10**

6.3. ClusterRequest Message

1 byte	4 bytes	PASize	LASize	PASize	LASize
2	Overlay Hash	Src PA	Src ID	Dest PA	Dest ID

Figure 9. ClusterRequest message.

The *ClusterRequest* message is sent by a cluster member to a cluster head to request membership in a cluster.

6.4. ClusterConfirm Message

1 byte	4 bytes	PASize	LASize	PASize	LASize
3	Overlay Hash	Head PA	Src ID	Dest PA	Dest ID

Figure 10. ClusterConfirm message.

The *ClusterConfirm* message is sent by a cluster head to accept the request to join a cluster. The message is sent in response to a *ClusterReqest* message.

6.5. ClusterReject Message

1 byte	4 bytes	PASize	LASize	PASize	LASize
4	Overlay Hash	Head PA	Src ID	Dest PA	Dest ID

Figure 11: ClusterReject message.

The *ClusterReject* is sent by a cluster head to reject the request to join the cluster. The message is sent in response to a *ClusterReqest* message when a cluster head cannot accept a new member.

6.6. Hello Message

1 byte	4 bytes	PASize	LASize	PASize	LASize
5	Overlay Hash	Src PA	Src ID	Dest PA	Dest ID

Figure 12. Hello message.

Cluster head and its members exchange *Hello* messages periodically.

6.7. Goodbye Message

1 byte	4 bytes	PASize	LASize	PASize	LASize
6	Overlay Hash	Src PA	Src ID	Dest PA	Dest ID

Figure 13. Goodbye message.

The *Goodbye* message is sent by either a leaving cluster head or by a cluster member to inform that they are no longer neighbors. When a cluster member or cluster head leaves, it sends a *Goodbye*. Subsequently, it responds to each received messages (e.g., *Hello*) with a *Goodbye*.

This message can be used by a cluster head to control its membership, i.e., reduce the number of members.

6.8. HeadReferral Message

1 byte	4 bytes	PASize	LASize	PASize	LASize	
7	Overlay Hash	Src PA	Src ID	Dest PA	Dest ID	Head List

1 byte	PASize	LASize	InfoSize
Number of enties	Head PA 1	Head LA 1	HeadInfo
	...		
	Head PA n	Head LA n	HeadInfo

#List entries

Figure 14. HeadReferral message.

The *HeadReferral* Message is sent by a cluster head to either cluster member or cluster heads (in its head cache) to exchange information about other cluster heads. The sending cluster heads include its HeadInfo content about itself into the referral when it sends a referral. The sending cluster head includes itself in the HeadReferral message. However, its own HeadInfo is not put in any specific position.

6.9. Rejoin Message

1 byte	4 bytes	PASize	LASize	PASize	LASize
8	Overlay Hash	Head PA	Src ID	Dst PA	Dst ID

Figure 15. Rejoin message.

The Rejoin Message is sent by a cluster head to force a node to rejoin the overlay network. The message should only be used when a cluster head discovers that there are nodes with duplicate addresses. In this case, the cluster sends a rejoin message to one of the duplicates. When a node receives a duplicate message, it leaves the overlay, and re-joins the overlay. In the process of re-joining it selects a new identifier, which will in all likelihood remove the duplicate.

6.10. HeadInfo Field

1 byte	8 bytes	4 bytes	4 bytes	8 bytes	4 bytes	1 byte
Node Type	Timestamp	Available Member	Current Members	Location	Rate	Metric

Figure 16. Format of the *HeadInfo* Field.

The *HeadOffer* and *Referral* messages contain information about cluster heads, which is formatted as a *HeadInfo* fields. The content of the *HeadInfo* field is written into the *HeadCache* Table.

NodeType: The type of the node that is described in the field. The available types are given in Table

Table 3. NodeType in HeadInfo field.

Head NodeType	**Head NodeType Field**
HeadOnly	1
Hybrid (Head)	2
Hybrid (Member)	3

Timestamp[7]: The last time this entry has been updated.
Available Member: The number of members that this cluster head node can accept.
Current Members: The number of current members at this cluster head.
Location[8]: The geographical location of the cluster head node using a specified coordinate system. The coordinate system is specified in the configuration file.
Rate: The bandwidth rate that this cluster head offers to each member in kilobits per second (kbps).
Metric: The value for the application-defined metric for this cluster head.

7. Timers

HeartbeatTime

[7] "The difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC" from Sun Java API Specifications.
[8] Two floating-point numbers in the single-precision floating-point format of IEEE 754-1985 specification

Default: 1000 ms
XPath: /Public/Node/CT/HeartbeatTime
Description: The time period between two consecutive Hello Messages, HeadDiscovery Messages, or HeadOffer Messages

MemberTimeout
Default: 3000 ms
XPath: /Public/Node/CT/MemberTimeout
Description: A cluster member entry will be removed if it has not been updated for this specified timeout value

HeadTimeout
Default: 3000 ms
XPath: /Public/Node/CT/HeadTimeout
Description: a cluster member will try to request an id from other head if a head entry has not been updated for this specified timeout value

This timer is reset if a member receives a *Hello* message from a head. If it does not receive any *Hello* message before the timer expires, a member assumes that a head has failed and tries to connect to another head.

MemberReferralInterval
Default: 5000 ms
XPath: /Public/Node/CT/MemberReferralInterval
Description: the time period between two consecutive HeadReferral Messages that a head send to its current members

HeadCacheReferralInterval
Default: 1000 ms
XPath: /Public/Node/CT/HeadCacheReferralInterval
Description: the time period between two consecutive HeadReferral Messages that a head send to its current heads in the head list

CacheEntryTimeout
Default: 10000 ms
XPath: /Public/Node/CT/CacheEntryTimeout
Description: an entry in a cache will be removed if it has not been updated for this specified timeout value.

OfferCollisionWindow
Default: 500 ms
Path: /Public/Node/CT/OfferCollisionWindow
Description: a Hybrid (Head) will ignore all HeadOffer messages from other Hybrids during this specified time period after it sends a Head Offer message.

8. Example

Comment: Let us discuss an example.....

9. Statistics

Comment: Let us discuss ....

The Clustering Protocol supports statistics. A list of supported statistics is ...

Required by M&C

- LogicalAddress (R)
- PhysicalAddress (R)
- NumOfNeighbors (R)
- NeighborTable (R)

Time

- NodeStartTime (R)
- NodeStopTime (R)
- HeartbeatTime (RW)

Contents of Tables : HeadCacheInfo

- Head (R)
  - PhysicalAddress (R)
  - Option (R)
    - HeadType (R)
      - Head
      - Hybrid(Head)
      - Hybrid(Member)
    - Timestamp (R)
    - Criteria (R)
      - AvailableMember (R)
      - Location (R)
        - Latitude (R)
        - Longitude (R)
      - OfferBandwidth (R)
      - OfferAvailability (R)
    - Result (R)

Status

- Head (R) – current head
- State (R)
  - Stopped
  - Member Candidate Without Head
  - Member Candidate With Head
  - Member
  - Head Without Member
  - Head With Member
- Mode (R)
  - Head
  - Member
  - Hybrid

Roots of its components statistics

- Adapter (R)

(R) stands for Read-Only
(RW) stands for Read&Write

Comment: Add detail for each statistic & think about any useful statistics

Comment: All statistics are implemented.