0.1 Introduction

The fast hand-over problem consists in *speeding up* the classic hand-over procedure (Section **??**) without dropping any already established connection and guaranteeing an acceptable continuity to real-time services.

The actual B.A.T.M.A.N. behaviour in case of hand-over simply consists in attaching the *new client* information (called HNA) to the next OGM sent out by the new Access Point so that every node in the network can learn the new route to the client as soon as each node receives the message (the other direction of the connection will obviously work because the client will use the routes already known by the new mesh node, Figure 1). This behaviour introduces a delay caused by the travel time of the OGM which is stricly connected to the B.A.T.M.A.N. convergence time. Usually this time is no more than a few seconds but if we imagine a scenario in which the clients are using VoIP or video streaming applications this could obviously lead to a services degradation.

Now that we know the general problem, let's analyze two different scenarios: a network with a single gateway and a network with multiple gateways.

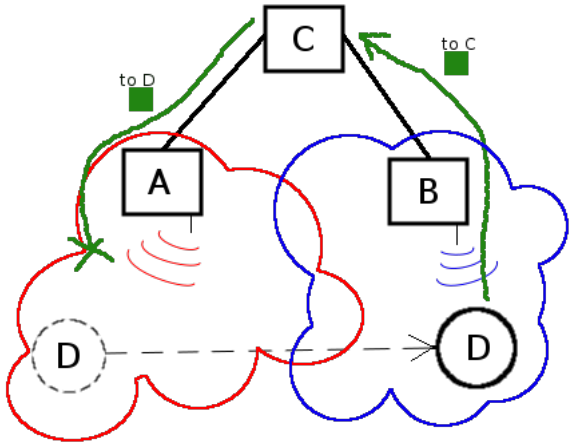

Figure 1: *Hand-over with B.A.T.M.A.N. The moving Client 'D' will immediatly send the packets through the correct route, while Node 'C' will still use the wrong route until it will receive the new OGM from B*

0.2 Into the solution

The new idea tries to take into account both problems and to solve them at the same time.

0.2.1 Announcing HNAs

Regarding the maximum HNA size problem, we decided to completely avoid to announce the whole local translation table in each OGM. Instead, each node will have a HNA_VERSION_NUM that starts from zero on node bootup and that will be increased on each HNA change (adding/removing a client). This value will be spread around within each OGM (in a new field). In this way, every node in the network will be able keep track of the changes of the node tables simply storing an array of values, one per node (*HOW TO EFFICIENTLY MANAGE THIS?*), other than the whole global translation table.

In case a node received an OGM with a HNA_VERSION_NUM greater than the value it stored for the source of the message (OGM.SRC from now on), a HNA_FAULT event will

occur and a TT_QUERY_REQUEST will be issued and sent in unicast to OGM.SRC to request an HNA update.

The TT_QUERY_REQUEST message will contain the following fields:

1. **SRC**: the source of the request
2. **DST**: the destination of the request
3. **OBSOLETE_HNA_VERSION_NUM**: the version number known by the requester
4. **NEW_HNA_VERSION_NUM**: the version number seen in the OGM by the requester for which the node is issueing the request
5. **TABLE_START**: the HNA entry from which the response have to start. Usefull for TT_QUERY_RESPONSE request for retransmission

The node receiving a TT_QUERY_REQUEST will reply with a TT_QUERY_RESPONSE, which will contain the last changes made in case that the OBSOLETE_HNA_VERSION_NUM is at most MAX_KEEP_HISTORY less than the actual value, while the whole local translation table otherwise.

The nodes in the path of a TT_QUERY_REQUEST have to inspect the message and check whether they already know such NEW_HNA_VERSION_NUM for the destination of the request.

The TT_QUERY_RESPONSE message will contain the following fields:

1. **SRC**: the source of the response
2. **DST**: the destination of the response
3. **HNA_VERSION_NUM**: the fresher HNA_VERSION_NUM for the source of the response
4. **TABLE_SIZE**: the size of the translation table being transmitted in number of address. If 0, the response contains only changes
5. if (TABLE_SIZE ¿ 0)
   (a) **TABLE_START**: the identification number of the first address in the translation table (usefull in case of TT_QUERY_RESPONSE fragmentation to check for missing packets)
   (b) **FRAG**: if 1, this is not the last TT_QUERY_RESPONSE message that composes this reply
   (c) **DATA**: the whole local translation table containing TABLE_SIZE addresses

   else

   (a) **DATA**: an array of *struct tt_change* of length at most MAX_KEEP_HISTORY

The *struct tt_change* is simply composed as follows:

```
struct tt_change {
    char oper; //1 if the address has been added, 0 if removed
    char addr[6]; //the address
}
```

The nodes in the path from the source to the destination of the TT_QUERY_RESPONSE have to inspect the message and possibly gain information from it.

In the event of a lost TT_QUERY_RESPONSE, that can be detected using the FRAG and the TABLE_START fields of the received fragments, a TT_QUERY_REQUEST can be sent specifing in the TABLE_START field the starting HNA from which we want to start the recovering. (*how to indicate the end?*)

0.2.2 Handling the roaming phase

Actually there is not a real handling procedure for the romaing phase in B.A.T.M.A.N. With this new HNAs announcing mechanism we also try to improve this scenario.

In case of a new client connecting to a mesh node, on the first packet generated by the client (so that the mesh node can detect it) a ROAMING_ADV packet will be sent in broadcast by the mesh node.

The ROAMING_ADV message will contain:

1. **SRC**: the new mesh node to which the client is connected
2. **CLIENT_ADDR**: the client address
3. **HNA_VERSION_NUM**: the source translation table version number
4. **OLD_NODE**: the old node to which the client was connected (in case of roaming)

In this way all the node that will receive the message will update their gobal translation table with this new information.

To improve the mechanism, the new node could first send a ROAMING_ADV in unicast to the old mesh node (if any) so that all the packets that are still flowing to it will be immediatly redirected to the new mesh node.

The broadcast ROAMING_ADV message will be spread over the network with the same mechanism of a normal OGM message.

0.2.3 Forwarding a packet

A mesh node sending a unicast packet to a client node has to include the destination mesh node's HNA_VERSION_NUM within the BATMAN_HEADER. In this way, every mesh node in the network that will have to forward the packet can first check whether it has an higher destination mesh node's HNA_VERSION_NUM and check if the client node is

still connected to it. In case of negative response, the forwarding mesh node will search for the new mesh node to which the client is connected and will modify the destination and HNA_VERSION_NUM fields of the packet before forwarding it.

0.2.4 Notes and Issues

- Some resending mechanisms must be taken into account to manage packet loss in the various scenario.
- The fragmentation of the TT_QUERY_RESPONSE must be analized deeper
- I assumed that a node increase its HNA_VERSION_NUM by one per one operation. So every increment can be described with one *struct tt_change* emelent.
- In case of TT_QUERY_REQUEST with TABLE_START greater than 0 (for retransmission request), how can we indicate the "end"?