**1. Discovery Algorithm**

**Version: v0.4**

**Last update: 2016-03-12**

This section describes the algorithm used to discover the topology of a network nodes.

**1.1 General Informations**

Each node of the network has an 8 bit ID and 2 of 4 full duplex connections (AXIOM 15 will have 2 full duplex connections, AXIOM 35 will have 4 full duplex connections).

The implemented algorithm to automatically discovers the topology and it provides for the presence of a master node which has an initially assigned ID.

At the end of the algoritmh execution, the master node knows the entrire topology of the network.

The nodes use 8-bytes “small-neighbours” messages in order to communicate with their neighbours during the discovery algorithm execution; the format of the small messages used for the discovery protocol is below specified:

typedef struct axiom\_small\_msg {

union {

axiom\_small\_tx\_hdr\_t tx;

axiom\_small\_rx\_hdr\_t rx;

} header; /\* Message header \*/

uint32\_t payload; /\* Data to be sent \*/

} axiom\_small\_msg\_t;

where the header fields are the following:

typedef struct axiom\_small\_tx\_hdr {

struct {

uint8\_t reserved : 2;

uint8\_t port : 3;

uint8\_t flag : 3;

} port\_flag; /\* port and flag fields \*/

uint8\_t dst; /\* destination (for tx) identificator \*/

uint8\_t spare[2];

} axiom\_small\_tx\_hdr\_t;

typedef struct axiom\_small\_rx\_hdr {

struct {

uint8\_t reserved : 2;

uint8\_t port : 3;

uint8\_t flag : 3;

} port\_flag; /\* port and flag fields \*/

uint8\_t src; /\* source (for rx) identificator \*/

uint8\_t spare[2];

} axiom\_small\_rx\_hdr\_t;

and the payload for discovery is the following:

typedef struct axiom\_discovery\_payload {

uint8\_t command; /\* Command of discovery messages \*/

uint8\_t src\_node; /\* Source node id \*/

uint8\_t dst\_node; /\* Destination node id \*/

uint8\_t src\_dst\_if; /\* Source interface | Dest Interface \*/

} axiom\_discovery\_payload\_t;

**1.2 Pseudo-code**

**/\***

**\* @brief Discover algorithm pseudo-code.**

**\* @param Topology: Matrix memorizing the topolgy of the network**

**\* @param NextId: Actual Node Identification value**

**\*/**

void discover\_phase(axiom\_dev\_t \*NextId, axiom\_dev\_t Topology[ [NUMBER\_OF\_INTERFACES])

{

< Read my node ID (MyNodeId) >

< Get the number of the connected interfaces >

< For Each 'i' active interface >

< Say over interface 'i':I am node MyNodeId give me your node id >

axiom\_send\_small\_discovery(dev, i, AXIOM\_DSCV\_CMD\_REQ\_ID, my\_node\_id, 0, 0, 0);

< Wait for the neighbourg response >

axiom\_recv\_small\_discovery(dev, &msg\_cmd, &src\_node\_id, &dst\_node\_id,

&src\_interface, &data\_src\_if, &data\_dst\_if);

if (msg\_cmd == AXIOM\_DSCV\_CMD\_RSP\_ID) < The neighbour has an ID >

< Update the topology data structure: my 'i' interface is connected the neighbour node

The neighbour node is connected to me on its 'srcInterface'>

if (msg\_cmd == AXIOM\_DSCV\_CMD\_RSP\_NOID) < The neighbour has no ID >

< Say over interface 'i': you are node NextId >

axiom\_send\_small\_discovery(dev, i, AXIOM\_DSCV\_CMD\_SETID, my\_node\_id, \*next\_id, 0, 0);

< Immediately update my routing table: NextId node is connceted to my 'i' interface >

axiom\_set\_routing(\*NextId, i);

< Update the topology data structure: my 'i' interface is connected the new NextId node NextId node is connected to me on its 'srcInterface'>

< Say over interface 'i': start discovery protocol, nextid >

axiom\_send\_small\_discovery(dev, i, AXIOM\_DSCV\_CMD\_START, my\_node\_id, \*next\_id, 0, 0)

< wait for the messages with the NextId topology data structure >

while (msg\_cmd != AXIOM\_DSCV\_CMD\_END\_TOPOLOGY)

< Receive a message with:

- 'i' interface connected Neighbour node topology

- the new updated nextid

- A Neighbour ID request on e non-'i' interface

>

axiom\_recv\_small\_discovery(dev, &msg\_cmd, &src\_node\_id, &dst\_node\_id,

&src\_interface, &data\_src\_if, &data\_dst\_if);

< request for my id from a node which is executing its discovery algorithm >

if (msg\_cmd == AXIOM\_DSCV\_CMD\_REQ\_ID)

< Reply: I am node 'MyNodeId', I am on interface 'srcInterface' >

axiom\_send\_small\_discovery(dev, src\_interface, AXIOM\_DSCV\_CMD\_RSP\_ID,

my\_node\_id, src\_node\_id, src\_interface, 0);

< topology info from the 'i' interface connected Neighbour>

else if (msg\_cmd == AXIOM\_DSCV\_CMD\_TOPOLOGY)

< Update the topology data structure with the received information >

< Update my routing table: from the sender node to the NextId node they are node connected on my actual receiving interface >

< End of neighbour discovery protocol, it sends me the new updated nextid >

else if (msg\_type == AXIOM\_DSCV\_CMD\_END\_TOPOLOGY)

\*NextId = data;

< end While >

< End For Each 'i' active interface >

}

**/\***

**\* @brief Master node Discovery Algorithm pseudo-code.**

**\* @param Topology: Matrix memorizing the topolgy of the network**

**\* (a row for each node, a column for each interface)**

**\* Topology[id\_node1][id\_iface1] = (id\_node2, id\_iface) means that**

**id\_node1 node, on its id\_iface\_1 is connected with**

**id\_node12node, on its id\_iface\_2**

**\*/**

void master\_node\_code(axiom\_dev\_t Topology[][NUMBER\_OF\_INTERFACES])

{

axiom\_dev\_t NextId = NODE\_MASTER\_ID;

/\* Initializes the Topology matrix: no node is connected \*/

init\_topology\_structure(Topology);

/\* Start the discovery phase\*/

discover\_phase(&NextId ,Topology);

}

**/\***

**\* @brief Slave node Discovery Algorithm pseudo-code.**

**\* @param Topology: Matrix memorizing the topolgy of the network**

**\* (a row for each node, a column for each interface)**

**\* Topology[id\_node1][id\_iface1] = (id\_node2, id\_iface) means that**

**id\_node1 node, on its id\_iface\_1 is connected with**

**id\_node12node, on its id\_iface\_2**

**\*/**

void slave\_node\_code(axiom\_dev\_t Topology[][NUMBER\_OF\_INTERFACES])

{

< Read My Node Id (MyNodeId) >

< Wait for the neighbour AXIOM\_DSCV\_CMD\_REQ\_ID type message >

while (msg\_CMD != AXIOM\_DSCV\_CMD\_REQ\_ID)

axiom\_recv\_small\_discovery(dev, &msg\_cmd, &src\_node\_id, &dst\_node\_id,

&src\_interface, &data\_src\_if, &data\_dst\_if);

< Imediatly Update my routing table: srcNodeId is connectet to me >

< If I already have a node Id >

if (MyNodeId != 0)

< Reply 'I am node 'MyNodeId', I'm on interface 'srcInterface' >

axiom\_send\_small\_discovery(dev, src\_interface, AXIOM\_DSCV\_CMD\_RSP\_ID, \*my\_node\_id, src\_node\_id, src\_interface, 0);

else

< Reply 'I don't have an ID', I'm on interface 'srcInterface' >

axiom\_send\_small\_discovery(dev, src\_interface, AXIOM\_DSCV\_CMD\_RSP\_NOID, \*my\_node\_id, src\_node\_id, src\_interface, 0);

< Wait for the neighbour AXIOM\_DSCV\_CMD\_START type message >

while (msg\_cmd != AXIOM\_DSCV\_CMD\_START)

axiom\_recv\_raw\_discovery(dev, &msg\_cmd, &src\_node\_id, &dst\_node\_id,

&src\_interface, &data\_src\_if, &data\_dst\_if);

NextId = dstNodeId; < that is my id, previously yet recevied into data field>

< Start the dicovery algorithm >

discover\_phase(&NextId, Topology);

< Send topology (Node1, if1, Node2, If2) list to the node which sent me the start message >

axiom\_send\_small\_discovery(dev, dstInterface , AXIOM\_DSCV\_CMD\_TOPOLOGY,

Node1, if1, Node2, if2);

< Says: <<Finished sending the topology structure, I send back actual NextId>> >

axiom\_send\_small\_discovery(dev, dstInterface, AXIOM\_DSCV\_CMD\_END\_TOPOLOGY, NextId ,0, 0, 0);

}

**1.3 Messages exanched during the discovery phase**

1.3.1 Type: **AXIOM\_DSCV\_CMD\_REQ\_ID**

Message for node ID request.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_DISCOVERY

message.header.tx.port\_flag.flag= AXIOM\_SMALL\_FLAG\_NEIGHBOUR

message.header.tx.dst = src\_interface; /\* sender node interface \*/

/\* Payload \*/

message.payoload.command = AXIOM\_DSCV\_CMD\_REQ\_ID;

message.payoload.src\_node = src\_node\_id; /\* sender node id \*/

message.payoload.dst\_node = not used

message.payoload.src\_dst\_if = not used

1.3.2 Type: **AXIOM\_DSCV\_CMD\_RSP\_ID**

Message sent by a node that already has an ID.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_DISCOVERY

message.header.tx.port\_flag.flag= AXIOM\_SMALL\_FLAG\_NEIGHBOUR

message.header. tx.dst = src\_interface; /\* sender node interface \*/

/\* Payload \*/

message.payoload.type = AXIOM\_DSCV\_CMD\_RSP\_ID;

message.payoload..src\_node = src\_node\_id; /\* sender node id \*/

message.payoload.dst\_node = dst\_node\_id; /\* receiver node id \*/

message.payoload.src\_dst\_if = src\_interface; /\* sender node interface \*/

1.3.3 Type: **AXIOM\_DSCV\_CMD\_RSP\_NOID**

Message sent by a node that not yet has an ID.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_DISCOVERY

message.header.tx.port\_flag.flag= AXIOM\_SMALL\_FLAG\_NEIGHBOUR

message.header. tx.dst = src\_interface; /\* sender node interface \*/

/\* Payload \*/

message.payoload.type = AXIOM\_DSCV\_CMD\_RSP\_NOID;

message.payoload.src\_node = not\_used

message.payoload.dst\_node = dst\_node\_id; /\* receiver node id \*/

message.payoload.src\_dst\_if = src\_interface; /\* sender node interface \*/

1.3.4 Type:  **AXIOM\_DSCV\_CMD\_SETID**

Message for sending the node ID.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_DISCOVERY

message.header.tx.port\_flag.flag= AXIOM\_SMALL\_FLAG\_NEIGHBOUR

message.header. tx.dst = src\_interface; /\* sender node interface \*/

/\* Payload \*/

message.data.type = AXIOM\_DSCV\_CMD\_SETID;

message.data.src\_node = src\_node\_id; /\* sender node id \*/

message.data.dst\_node = id\_to\_set; /\* id assigned to the receiver node \*/

message.data.src\_dst\_if = not\_used

1.3.5 Type:  **AXIOM\_DSCV\_CMD\_START**

Message for sending the request of starting the discovery protocol.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_DISCOVERY

message.header.tx.port\_flag.flag= AXIOM\_SMALL\_FLAG\_NEIGHBOUR

message.header. tx.dst = src\_interface; /\* sender node interface \*/

/\* Payload \*/

message.payload.command = AXIOM\_DSCV\_CMD\_START;

message.payload src\_node = src\_node\_id; /\* sender node id \*/

message.payload dst\_node = actual\_max\_id; /\* actual maximum node id used \*/

message.payload src\_dst\_if = not\_used

1.3.6 Type:  **AXIOM\_DSCV\_CMD\_TOPOLOGY**

Message for sending routing information: 'node1\_id' is connected on its 'node\_if1' interface with 'node2\_id' on its 'node\_if2' interface.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_DISCOVERY

message.header.tx.port\_flag.flag= AXIOM\_SMALL\_FLAG\_NEIGHBOUR

message.header. tx.dst = src\_interface; /\* sender node interface \*/

/\* Payload \*/

message.payload.type = AXIOM\_DSCV\_CMD\_TOPOLOGY;

/\* The folowing fields memorize this information:

\* (node1\_id, node1\_ if1) is connected to (node2\_id, node2\_if1) \*/

message.payload.src\_node = node1\_id;

message.payload.dst\_node = node2\_id;

message.payload.src\_dst\_if = node1\_if | node2\_if

1.3.7 Type:  **AXIOM\_DSCV\_CMD\_END\_TOPOLOGY**

Message for declaring the end of the topology matrix already sent and for sending back the actual maximum node ID used.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_DISCOVERY

message.header.tx.port\_flag.flag= AXIOM\_SMALL\_FLAG\_NEIGHBOUR

message.header. tx.dst = src\_interface; /\* sender node interface \*/

/\* Payload \*/

message.payload.type = AXIOM\_DSCV\_CMD\_END\_TOPOLOGY;

message.payload.src\_node = actual\_max\_id; /\* actual maximum node id used \* inside the network \*/

message.payload.dst\_node = not\_used

message.payload.src\_dst\_if = not\_used

**2. Routing tables computation**

This section describes the modality used for calculating the routing tables of each node.

**2.1 General Informations**

At the end of the discovery algorithm, the Master node (which knows the entire network topology) performs a similar-Dijkstra algorithm in order to compute the routing tables of all the nodes of the network.

This algorithm assumes that each link between two nodes has an unitary cost (the shortes path has to be computed) and it starts from Master neighbours e than to their neighbours, and so on.

In computing Y node routing table, each time an Y neighbour node or an Y neighbours connected node is found, the Master immediately add this X node into the Y routing table; that is the first time the Master found an Y connected node it adds the node into the routing table.

Since the nodes are added into the routing table by Master analyzing increasing neighborhood levels, the shortest path from Y to each network node is computed.

***This algorithm computes only a single path between two given nodes.***

**2.2 Pseudo-code**

< for each node Y into network >

< Master node analyzes the network topology matrix and it memorizes Y node neighbours into Y routing table >

<do>

< for each X neighbour of Y >

< Master node analyzes the network topology matrix and it memorizes the X node neighbours into Y routing table >

< end for >

< while (not all nodes are into Y routing table) >

< end for >

**3. Routing tables delivery**

This section describes the modality used for distributing the routing tables of each node from the Master node to the proper receiver.

**2.1 General Informations**

Master node sends the routing table with node-ID increasing-order and it uses 8-bytes “small” messages to communicate with the routing table receiver node.

During this phase, messages can be exchanged between nodes thanks to the each node partial routing table, set during the discovery algorithm execution.

When Master node has delivered all the routing tables, it sends to its neighbours an 8-bytes “small-neighbours” message indicating to set the previously received routing table.

A node receiving this small message has to update its routing table and then to send to its neighbours the recevied message.

The format of the raw messages used for routing tables delivery is below specified:

typedef struct axiom\_small\_msg {

union {

axiom\_small\_tx\_hdr\_t tx;

axiom\_small\_rx\_hdr\_t rx;

} header; /\* Message header \*/

uint32\_t payload; /\* Data to be sent \*/

} axiom\_small\_msg\_t;

where the header fields are the following:

typedef struct axiom\_small\_tx\_hdr {

struct {

uint8\_t reserved : 2;

uint8\_t port : 3;

uint8\_t flag : 3;

} port\_flag; /\* port and flag fields \*/

uint8\_t dst; /\* destination (for tx) identificator \*/

uint8\_t spare[2];

} axiom\_small\_tx\_hdr\_t;

typedef struct axiom\_small\_rx\_hdr {

struct {

uint8\_t reserved : 2;

uint8\_t port : 3;

uint8\_t flag : 3;

} port\_flag; /\* port and flag fields \*/

uint8\_t src; /\* source (for rx) identificator \*/

uint8\_t spare[2];

} axiom\_small\_rx\_hdr\_t;

and the payload for discovery is the following:

typedef struct axiom\_routing\_payload {

uint8\_t command; /\* Command of routing messages \*/

uint8\_t node\_id; /\* Node ID to set into the routing table \*/

uint8\_t if\_mask; /\* Mask of the interface \*/

uint8\_t spare;

} axiom\_routing\_payload\_t;

**2.2 Messages exanched during the routing tables delivery**

2.2.1 Type: **AXIOM\_RT\_CMD\_INFO**

Message for sendig a (node\_id\_set,if\_set) pair of the' dst\_node\_id' node .

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_ROUTING

message.header.tx.port\_flag.flag= 0

message.header.tx.dst = dst\_node\_id; /\* receiver node id \*/

/\* Payload \*/

message.payload.command = AXIOM\_RT\_CMD\_INFO;

message.payload.node\_id = node\_id\_set /\* node to set into dest\_node\_id routing table \*/

message.payload.if\_mask = if\_set /\* mask intarface to set into dest\_node\_id

\* routing table to reach node\_id node \*/

message.payload.spare = not used

2.2.2 Type: **AXIOM\_RT\_CMD\_END\_INFO**

Message used to communicate the end of the routing table delivery to 'dst\_node' node.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_ROUTING

message.header.tx.port\_flag.flag= 0

message.header.tx.dst = dst\_node\_id; /\* receiver node id \*/

/\* Payload \*/

message.payload.command = AXIOM\_RT\_CMD\_END\_INFO;

message.payload.node\_id = not used

message.payload.if\_mask = not used

message.payload.spare = not used

2.2.3 Type: **AXIOM\_RT\_CMD\_RT\_REPLY**

Message used by each slave node to communicate to master to have received the routing table.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_ROUTING

message.header.tx.port\_flag.flag= 0

message.header.tx.dst = master\_node\_id; /\* id of node master \*/

/\* Payload \*/

message.payload.command = AXIOM\_RT\_CMD\_RT\_REPLY;

message.payload.node\_id = sende\_node\_id /\* id of node which sent the reply to master \*/

message.payload.if\_mask = not used

message.payload.spare = not used

**2.3 Messages for routing table setting**

2.3.1 Type: **AXIOM\_RT\_CMD\_SET\_ROUTING**

Message to inform all neighbours to start using the routing table previously sent.

axiom\_small\_msg\_t message;

/\* Header message \*/

message.header.tx.port\_flag.port = AXIOM\_SMALL\_PORT\_ROUTING

message.header.tx.port\_flag.flag= AXIOM\_SMALL\_FLAG\_NEIGHBOUR

message.header.tx.dst = src\_interface; /\* sender node interface \*/

/\* Payload \*/

message.payload.command = AXIOM\_RT\_CMD\_SET\_ROUTING;

message.payload.node\_id = not used

message.payload.if\_mask = not used

message.payload.spare = not used