H2020 i**P** **O**ver Ic**N**- the bet**T**er IP (POINT)

Design Description

THE Management API (MAPI)

Author: Mays AL-Naday

1. Objective

New solutions of the POINT platform, such as: the Monitoring framework, path management and TE introduced the need for applications (ICN or otherwise) to have a management communication with the ICN core, either to: retrieve up to date core information, current stats or to set a configuration parameter(s). For this purposes the core platform have been extended with a dedicated IPC channel for management communication as well as a set of predefined management information with their handling functions. Management requests will be communicated to the core through a dedicated interface, namely the Management Application Programming Interface (MAPI).

MAPI will be used by Management Applications (MApps) to retrieve management information from the core node to assist the application in fulfilling further functionality; or respond to network variations appropriately. Information retrieval will be facilitated through a set of function calls that either allow the application to ask for well-defined information; setup a configuration parameter or to listen for updates/notifications from the core node.
MAPI coexists with other APIs such as the ICN API and MOLY, as shown in Figure 1. it uses IPC (netlink sockets) to support the bidirectional communication on a different port number.

MAPI follows a GET/SET model for retrieving management information in a synchronous communication from the core node and updating the node settings/configuration based on network changes. An asynchronous *callback or notification* interest can be registered by applications that also would like to retrieve future updates of the management information.

2. Management information

This includes node-specific information that would assist an application in managing the node and fulfilling further functionality, including (but not limited to):

information	Type	description
nodeID	String, unsigned int	Node identifier in either one of two formants
RVTMFID	Bitvector	FID to reach RV/TM
connection_status	Unsigned char	Connection to the network
Up_time (hours)	double	Time since start of the node
forwarded_bytes	Long int	Traffic forwarded

sinked_bytes	Long int	Received traffic
active_pubs	Long int	Active publications
acitve_sub	Long int	Active subscriptions
acitve_iSubs	Unsigned int	Active implicit subscribers

3. Management Application (MApp)

A MApp is any application that requires management information from the core node to perform part or all of its functionalities. Accordingly some applications are only assuming the role of a MApp for when they obtain this information; whereas others are dedicated MApps. Example MApps include:

- Monitoring Agent (MA): ask for the nodeID
- NAP: ask for its nodeID (if needed)
- LinkState Monitor (LSM): ask for nodeID, RVTMFID, set the node connected/disconnected status

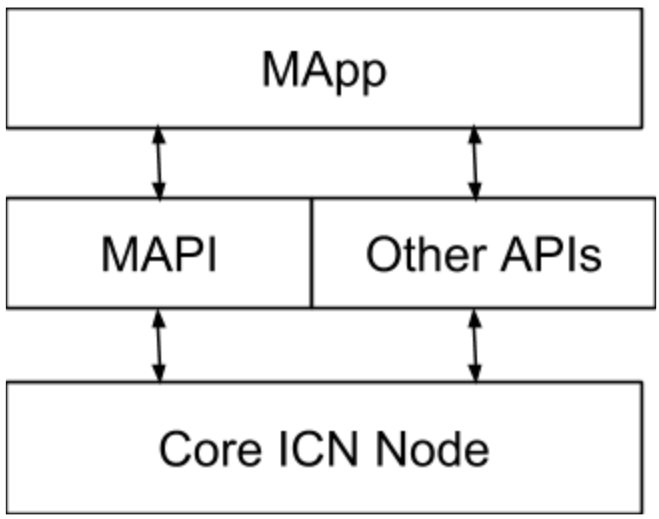

Figure 1: Functional Block Diagram showing MAPI placement within a POINT node

4. Management separate from ICN

MAPI communicates with the core platform over a dedicated netlink socket for management, that is separate from the socket used for ICN communication. A dedicated Process ID (PID) has been allocated for the communication, namely `PID_MAPI`. Figure 2 illustrates the separation within the internal functional elements within the core platform.

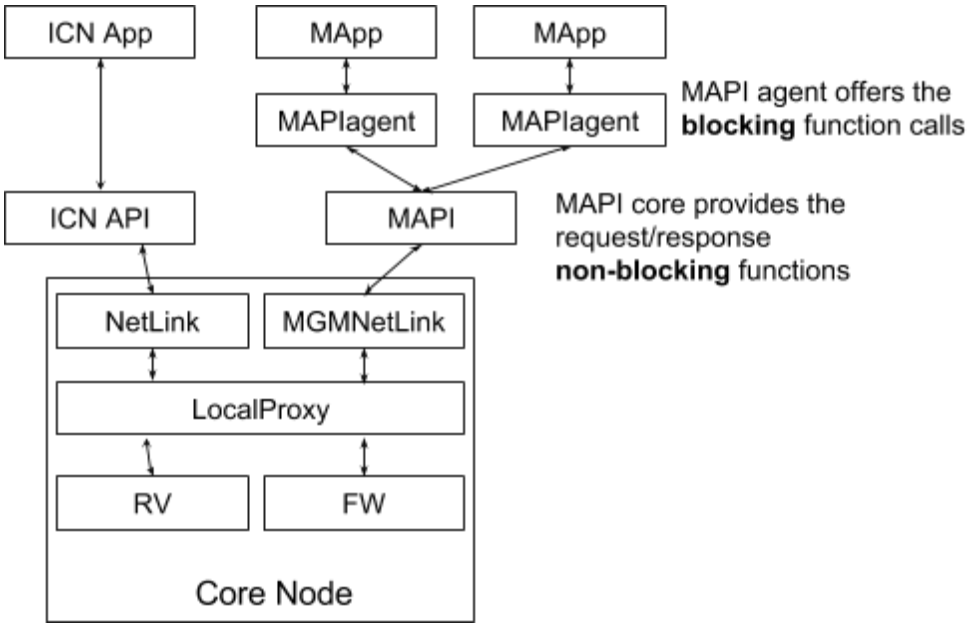

Figure 2: Extended Core node to support dedicated management communication

5. Basic Primitives

5.1 GET_NODEID

MApps use this primitive to request the current nodeID from the core platform.

5.1.1 Function call

```
Bool mapi::get_nodeID(string &nodeID )
Input: reference to the variable that should return the nodeID
Return: Boolean indicate success/failure of the call

Bool mapi::get_nodeID(unsigned int &nodeID )
Input: reference to the variable that should return the nodeID
Return: Boolean indicate success/failure of the call
```

5.2 GET_RVTMFID

MApps use this primitive to request the current RVTMFID(s) from the core platform. Also to be updated if the nodeID change at any point in time.

5.2.1 Function call

```
Bool mapi::get_RVTMFID(Bitvector &FID)
Input: reference to the variable that should return the RVTMFID
Return: Boolean indicate success/failure of the call
```

5.3 SET_CONNECTION_STATUS

The connection status of the node to either CONNECTED/DISCONNECTED, indicating whether or not the node is connected to the ICN network.

5.3.1 Function call

```
Bool mapi::set_connection_status(unsigned char status)
Input: status
Type: unsigned char
Return: Boolean indicate success/failure of the call
```

6. Extended Primitives (Reg. for notifications)

These primitives are used to register interest in receiving future updates about one of the information listed in Section 2.

6.1 GET_NODEID_UPDATE

This primitive should be used to register interest in receiving updates of the information name nodeID. This call will result in registering a state of <*nodeID, processID*> in the core node, which allows the latter to track MApps that are interested in the nodeID and notify them should an update occur. Notice that processID represent the identifier of the requesting MApp.

6.1.1 Function call

```
Bool mapi::get_nodeid_update()
Input: none
Return: Boolean indicate success/failure of sending the message
```

6.2 GET_RVTMFID_UPDATE

This primitive should be used to register interest in receiving updates of the information name *RVTMFID.* This call will result in registering a state of <*RVTMFID, processID*>, which allows the core node to track MApps that are interested in the RVTMFID and notify them when the RVTM FID is updated

6.2.1 Function call

```
Bool mapi::get_rvtmfid_update()
Input: none
Return: Boolean indicate success/failure of sending the message
```

7. Functional Model

The basic communication model of MAPI features a synchronous and blocking Request/Response message exchange. Requests/responses are in the format of *Type-Length-Value.* MAPI can also support an asynchronous, event-based, notification mechanism that allow the core node (aka blackadder) to notify the applications of update to node information.

7.1 Asynchronous notification mechanism

An event-based notification mechanism can also be supported through MAPI, whereby blackadder can send local notifications of well-identified types to MApps that have already registered interest for one or more of the management information listed in Section 2. Returned events also follow a TLV structure, including but not limited to:

Type	Length (in bits)	Value
UPDATE_NODEID	nodeid.length()	nodeID
UPDATED_RVTMFID	FID_LEN * 8	RVTMFID