
H2020 iP Over ICN- the better IP (POINT)

Design-ICN-SDN-Application

POINT ICN-SDN Application Design



List of Authors: George Petropoulos, Konstantinos Katsaros

[1.Introduction](#)

[2.Implementation](#)

1.Introduction

The ICN-SDN application implements the server side of the TM-SDN interface and supporting functionalities. It implements relevant methods and functionalities, including:

- The definition of the TM-SDN protocol format using Google Protocol Buffers, the generated header and source files, as well as the supporting methods to read and write Google Protobuf objects to streams.
- The server side of the ICN-SDN interface, which handles all the cases of received messages and coordinates next functions.
- The resource management functions, which provide unique node, link and internal link identifiers, and update the state of the topology.
- Supporting configuration methods, dealing with reading of network interfaces and initialization and update of Click files (used for bootstrapping purposes).

The ICN-SDN is assumed to be running on the same machine as Topology Manager does. This will facilitate the topology update in case of new or removed links, the resilience handling in case of link failures and the traffic engineering functions for path calculation. In the future, the ICN-SDN will be integrated into the Topology Manager, in order to make the resource assignment process part of the whole topology management functions.

2.Implementation

In terms of implementation, the ICN-SDN application is implemented as a boost:asio server application, and consists of the following source files. The key methods are also presented briefly, while more detailed documentation is exported at doc/ folder of the source repository, generated by Doxygen.

- `server.cpp`: The server class, implementing the server side of the TM-SDN interface, and also listening for bootstrapping protocol messages.
 - `void start_accept()`: The method which listens for incoming TM-SDN messages and acts accordingly. In case of ResourceRequest messages it calls the `resource_manager` module and returns the assigned identifiers included in ResourceOffer message. In case of TrafficMonitoring and LinkStateMonitoring messages, it prints the received information.
 - `int main(int argc, char* argv[])`: The main method initializing the required threads: the server process and bootstrapping message listener.
 - `resource_manager.cpp`: The resource manager class. It will generate unique node, link and internal link identifiers, and also keep track of assigned ones.
 - `string generate_node_id(string node_information)`: The method which generates a new node identifier, given node's information. It is used to
-

generate unique node identifier for requests performed by the SDN controller. New nodes are identified by their Openflow identifier.

- `void generate_link_id(TmSdnMessage::ResourceRequestMessage::ResourceRequest request, TmSdnMessage::ResourceOfferMessage::ResourceOffer *offer):` The method which creates a resource offer for a received resource request. It is used to generate unique link identifier for resource requests via the TM-SDN interface. It is the main method which orchestrates the rest of functions for the generation of unique resources for SDN nodes. It first checks whether the node has not unique identifiers and then creates, otherwise returns the already provided ones. All are then included in the resource offer to be sent back to the SDN controller.
 - `string generate_link_id(string node_information, string attached_node_information):` The method which generates a new link identifier, given node's information. It is used to generate unique link identifier for requests performed by the SDN controller. New links are identified by the SDN controller-internal identifiers of the 2 nodes being connected.
 - `string generate_internal_link_id(string src_node_id, bool flag):` The method which generates a new link internal identifier. It is used to generate unique internal link identifier for a new node.
 - `void addNode(string src, bool tmrv):` The method which adds a new node to the topology graph.
 - `void addNodeConnector(string src_node, string dst_node, string lid_bits, string ilid_bits):` The method which adds a new node connector to the topology graph.
 - `void updateGraph():` The method which updates the topology graph.
 - `configuration.cpp:` The class for creating and updating click configuration.
 - `void initialize_click():` The method which initializes click.
 - `void update_click(string new_node_id, string new_rv_fid, string new_tm_fid, string new_internal_lid, string new_link_id):` The method which updates the click configuration with the new parameters.
 - `void read_interfaces():` The method which reads the available network interfaces.
-