Seventh FRAMEWORK PROGRAMME
FP7-ICT-2009-5 - ICT-2009-1.6
Future Internet Research and Experimentation (FIRE)

SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

Deliverable D2.1
"**Routing System Architecture**"

Project description
Project acronym: **EULER** Project full title: Experimental UpdateLess Evolutive Routing Grant Agreement no.: **258307**
Document properties
Number: FP7-ICT-2009-5-1.6-258307-D2.1 **Title**: Routing System Architecture **Responsible**: Dimitri Papadimitriou (ALB) **Editor(s)**: Dimitri Papadimitriou (ALB) **Contributor(s)**: Miguel Camelo, Davide Careglio, Jean-Charles Delvenne, Lluis Fabrega, Christian Glacet, Nicolas Hanusse, Clemence Magnien, Dimitri Papadimitriou, Sahel Sahhaf, Bernard Sales, Fabien Tarissan, Wouter Tavernier, Pere Vila. **Dissemination level**: Public (PU) **Date of preparation:** *Date* **Version: 1.1**

D2.1 - Routing System Architecture

List of Authors

Affiliation	Author
Alcatel-Lucent Bell (ALB)	Dimitri Papadimitriou, Bernard Sales
IBBT	Sahel Sahhaf, Wouter Tavernier
INRIA	Nicolas Hanusse, Christian Glacet
UCL	Jean-Charles Delvenne
UPC/CAT	Davide Careglio, Pere Vila, Lluis Fabrega, Miguel Camelo
UPMC	Fabien Tarissan, Clemence Magnien

List of Acronyms

DRI	Discover Routing Information
DRP	Determine Routing Path
DTI	Discover Topology Information
EFFBD	Enhanced Functional Flow Block Diagram
FT	Forwarding Table
FTE	Forwarding Table Entry
PRFT	Produce Routing and Forwarding Tables
HIP	Host Identity Protocol
IIB	ILB Information Base
IIU	Identifier Information Unit
ILB	Identifier-Locator Binding
LIU	Locator Information Unit
PDAI	Pre-process Discovered and Associated Information
RIB	Routing Information Base
RIU	Routing Information Unit
RNLC	Resolve Name to Locator / Coordinate
RT	Routing Table
RTE	Routing Table Entry
TIB	Topology Information Base
TIU	Topology Information Unit

Table of Contents

1. Introduction

This section outlines the objectives and the motivations for initiating architecture work for the Internet routing system by means of a systematic functional and information modeling approach. In other terms, this section provides a first level answer to the questions "Why do we need a routing system architecture" and "What are the motivations to follow a systematic model-driven approach to specify this architecture?" These questions are important not only from a theoretical perspective but also in the context of the experimentally-driven multi-disciplinary research approach promoted by the FIRE initiative.

1.1 Objectives

In distributed routing, each node part of the routing system implements a routing function that computes for any reachable destination name a loop-free routing path so that incoming messages directed to a given destination can reach it. Routing is an essential function of the Internet at networking but also applicative layers. In the former case, it is referred to as packet, datagram or data traffic routing and in the latter information routing.

As already foreseen and reported twenty years ago (see for instance RFC 1287), the most fundamental issues faced by the Internet are related to its (inter-domain) routing system (which comprises about 350k nodes/routers and 35k autonomous systems), in particular in terms of its scalability, convergence, and stability properties. These issues result from the Internet topology evolution with respect to the design of the addressing system design and the addressing space usage together with the intrinsic limitations of the inter-domain routing protocol architecture and properties:

o Addressing design and routing scaling: originally, host IP addresses were assigned based on network topological location. Adoption in the mid 90's of dedicated mechanisms to perform address aggregation (called CIDR in the IETF jargon) was felt sufficient to handle address scaling. However, conditions to achieve efficient address aggregation and thus relatively small routing tables are not met anymore. This situation is exacerbated by the current Regional Internet Registry (RIR) policy that allocates Provider-Independent (PI) addresses to sites whereas PI addresses are not topologically aggregatable; thus, making CIDR ineffective to handle address scaling. The result is that the increase of routing table sizes worsens over time as these prefixes are allocated without taking into account effects on the global routing system. Indeed, routing on PI address prefixes requires additional routing entries in the Internet routing system whereas the "costs" incurred by these additional prefixes, in terms of routing table entries and associated processing overhead, are supported by the global routing system as a whole, rather than directly charged to the users of the PI space. Coupled to the increase of the number of routes —resulting from topology independent address prefix allocation that impedes prefix aggregation—, site multi-homing (~25% of sites), ISP multi-homing, and inter-domain traffic-engineering exacerbate the limitations of the Internet routing system. Nowadays, the latter must not only scale with increasing network size and growth of the Internet but also with a growing set of constraints and functionalities: bottom line, routers shall cope with growing routing table size even if the network itself would not be growing.

o Border Gateway Protocol (BGP) protocol architecture: BGP, the inter-domain routing protocol of the Internet has been designed to compute and maintain Internet routes between administrative domains. The algorithm for route computation and maintenance selected for BGP is subject to Path Exploration phenomenon: BGP routers may announce as valid, routes that are affected by a failure and that will be withdrawn shortly later during subsequent routing updates. This phenomenon is (one of) the main reasons for the large number of routing update messages received by inter-domain routers. In turn, path exploration exacerbates inter-domain routing system instability and processing overhead. Both result in delaying BGP convergence time upon topology change/failure or policy change. Several mitigation mechanisms exist but experience has shown that the cure is worse than the disease. Indeed, the reactive approach often at the origin of the design of these mechanisms does not allow studying their impact on the global routing system.

Solving these problems requires to address multiple challenges altogether

1. The routing table size scaling (in terms of memory space consumption or memory complexity) required to sustain the increasing number of routing table entries resulting from the growing number of nodes/routers, networks, and autonomous systems. It is important to underline here that even without network size growth in terms of number of nodes/routers, networks, and autonomous systems, that the increasing number of peering links observed on the Internet topology, induces an increase in memory space consumption to store routing information. Indeed, according to the best available data (see, e.g., Caida) as the Internet topology grows in terms of links, the number of links connecting nodes of similar degrees (tangential links, i.e., the so-called peering links between AS) increases faster than the number of links connecting low-degree to high-degree nodes (radial links, i.e., the so-called customer-to-provider links). In turn, this excess of tangential links is responsible for the increase of the number of BGP routing paths per destination -even if the network size itself does not increase.
2. The communication complexity (measured as the rate of exchange between nodes of routing messages in order for the routing function to properly operate) as the dynamics of the routing information exchanges between routers increasingly impacts the routing system convergence, and stability properties.
3. The increasing architectural complexity (measure of the complexity of a given architecture proportionally to its number of components and the number of interactions among components) of the Border Gateway Protocol (BGP), underlying the Internet routing system.

Prominent research efforts have been conducted over last decade to overcome the limitations of BGP including trials to 1) Shorten routing update interval (to accelerate convergence), 2) Route flap damping (to prevent instabilities), 3) Inclusion of AS-Path dependency and failure root cause/location information (to mitigate path exploration effects), or 4) Multiple AS-Path per destination (to improve fault-tolerance). Over time, some of these ad-hoc improvements have permitted to limit the performance degradation of the Internet routing system but none of them improves the intrinsic limitations of the path-vector routing algorithm impacting the scalability (stretch-1 routing paths) and the convergence (due to path exploration) properties of the routing system.

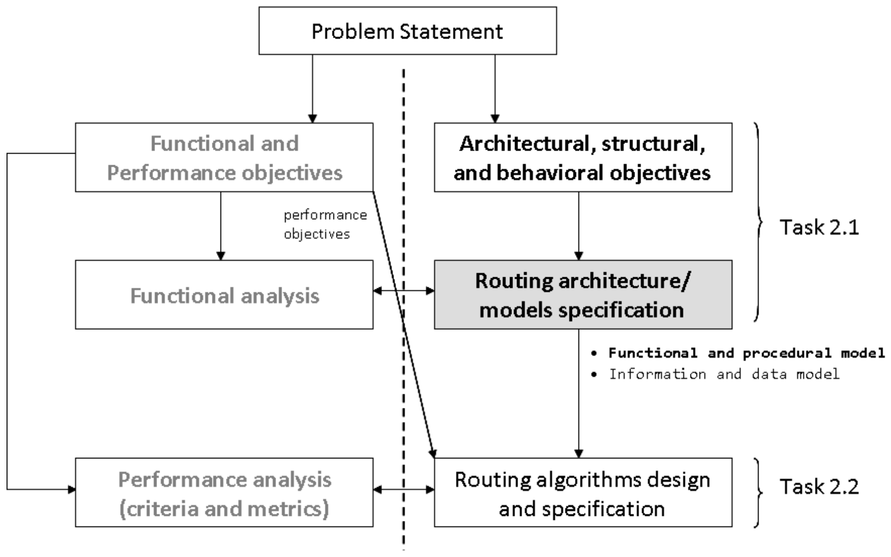
Consequently, there is a growing consensus among the research community that the current practice of incrementally improving the routing protocol of the Internet will not be able to sustain its continuous growth at an acceptable cost and speed. This observation led research on distributed routing algorithmic to look at disruptive paradigms to address the above mentioned challenges altogether.

1.2 Approach

One of the main root causes resides also in the lack of architectural modeling of the global routing system. Indeed, the main driver for a routing protocol design is the selection of the algorithm and associated mechanisms for the route computation. But as the routing system is not properly modeled, the impacts of these design choices on the global routing system are almost impossible to evaluate (hence, the cure is sometimes worse than the disease).

Therefore, in order to address these challenges altogether, we model in the context of the WP2/Task 2.1 the routing system by identifying its functional (architectural) components, their relationships, and their spatio-temporal distribution (functional model) together with the information properties, operations and relationships (information model).

In the context of WP2/Task 2.2, the next step will then be dedicated to i) the specification of the routing algorithms together with their validation, ii) their functional analysis together with their impact on the global routing system architecture, and iii) their performance evaluation and analysis in the context of WP3.


Problem Statement
Functional and Performance objectives
Architectural, structural, and behavioral objectives
performance objectives
Task 2.1
Functional analysis
Routing architecture/ models specification
• Functional and procedural model
• Information and data model
Performance analysis (criteria and metrics)
Routing algorithms design and specification
Task 2.2

1.3 Motivations

The main motivations for taking a systematic approach in specifying the architecture of the routing system can be summarized as follows:

- To determine a common architectural baseline and answer whether a common architecture can cover all/part of the routing models/schemes initially considered in the context of this project, e.g., compact routing, greedy routing, etc.? and where do we need to introduce per-routing scheme customization and/or specialization and at which level of the routing system specification?
- To identify open routing research areas by determining which routing (sub-)functions are currently under-specified or even badly specified but also which routing (sub-)functions can be replaced, added or removed;
- To define a common "language"; thus, prevents misinterpretations among different dimensions and actors involved in the design of the routing system;
- To provide a structuring and cohesive role (pending on the commonality level between the different routing model components);
- To detail the model views up to the level appropriate for further routing systems engineering (establish routing system engineering controls over the allocations and interfaces);
- To provide executable models of the interaction. Note that in distributed or multi-agent systems like the Internet routing, the choice/decision is locally performed by each agent independently of the others using the exchanged information (i.e., discovered information) but individual agents' choice/decision affects other agent's choice/decision);
- To enable concurrent design of routing protocols (and associated operations), control systems, and components and to facilitate comparison between the different routing schemes that will be designed in context of Task 2.2.

Additionally, by following such systematic modeling approach, we expect that:

- In the computer communication context, this approach leads to modular software development (and prevents duplicates);
- By starting from a top-level view down to the design of the routing protocol (and their components), it results into a holistic approach to the problem space that is complementary to experimental/bottom-up approach;
- Past experience shows that without well defined system architecture, adding or removing functionality leads to further complexity (see IP control plane design today); in practice, finding the suitable tradeoff between evolutivity, flexibility/adaptivity, and performance is critical to ensure longevity of the architecture.

2. Terminology

The terms defined in this section are structured in accordance to the sequence of the WP2 (sub-)tasks. As its title indicates, this document focuses on the architecture of the routing system. Hence, defining the architecture is the starting point of this section.

Architecture: Many definitions of (system) architecture have been formulated over time. We borrow the terms of our definition from D.E.Perry and A.L.Wolf (in "Foundations for the Study of Software Architecture", ACM SIGSOFT Software Engineering Notes, Vol.17, No.4, October 1992), Garlan and D.E.Perry (in their Editorial to the IEEE Transactions on Software Engineering, April 1995) and Booch (from his presentation at the Software Developers Conference in 1999). We define as "system architecture" a set of functions, states, and objects/information (referred to as "elements") together with their behavior, structure (relationships and interactions), composition and spatio-temporal distribution. The specification of these elements is referred to as the functional model architecture, the information/object model architecture and the state model architecture, respectively. Structure and behavior define two different level of the architecture: structural description and behavioral description.

2.1 Functional and Procedural Model

Function: characteristic action (verb) a system performs to transform available inputs to the desired outputs.

Functional model: determines a systematic decomposition of the (routing) system by defining the routing system functional design, its inputs/outputs, and its various interfaces. This modeling technique (or methodology) enables thus to systematically describe the automated processing that a complex system must perform to transform available inputs to the desired outputs. The fundamental underlying idea is the following: the system is viewed as a distributed computing function (or, more generally, as solving an information processing problem). The processing performed by the system can be explained by iteratively decomposing the more complex top-level functions or functional areas into a set of simpler functions (sub-functions). Each sub-function is computed by an organized sub-system. This decomposition is performed up to the level of atomic functions, i.e., as their name indicates atomic functions are functions that can't by definition be further decomposed. The overall objective of performing such functional decomposition stems from the assumption that each sub-function taken individually will be simpler than the original functions.

Procedure: method to perform a given task/action specified as a sequence of discrete steps (each step being a finite instruction or operation, finite meaning represented by a finite number of symbols) which have to be executed in a regular definite order in order to always obtain the same result under the same conditions. Note that in the routing system modeling context, procedures are devised as computational models of sequences of operations that the routing function is conjectured to perform as it processes routing information.

Procedural model: a procedural model refers to the formal description of the procedures. In the context of this document, a **routing model** refers to the procedural model together with its associated data model, state machine model, and the data communication model which characterizes the interactions and interfaces (i.e. messages, calls, events, etc.) between the elements of the model (i.e. procedures, data structures, state machines).

Algorithm: consists of sequences of steps (operations, instructions, statements) for transforming inputs (preconditions of the algorithm: description of the inputs, including their types as well as any relationships or properties that they must satisfy before execution) to outputs (post-conditions of the algorithm: description of the outputs, including all relationships and properties that must be satisfied after execution). It provides the description of the effective computational procedures composing the procedural model. Note that an algorithmic scheme provides the description of class of effective computational procedures (that are based on the same "principles") composing the procedural model. Termination of the algorithm refers to the situation such that whenever the preconditions are satisfied, the algorithm will stop running after a finite number of steps (e.g., no infinite loops). Correctness of the algorithm refers to the situation such that whenever the pre-condition are satisfied before the algorithm executes, the post-conditions will be satisfied after it executes. Note: correctness without termination is called partial correctness whereas total correctness refers to partial correctness with termination.

2.2 Information and Data Model

Information/object model: representation of the concepts, relationships, properties, constraints, rules, and operations to specify the information semantics for a given application domain/area. Hence, such model is also referred to as semantic data model or conceptual data model. There are different methods for developing an information model. Among them, the **entity-relationship (E-R) model** expresses in terms of entities, the relationships (or associations) among those entities, and the attributes (properties) of both the entities and their relationships. The ultimate goal of applying such model is to capture as much of the meaning of the data (semantic) as possible so as to obtain a better design that is scalable and easier to maintain. The E-R model in its original form does not support specialization/generalization abstractions (also termed hierarchies). Hence, the E-R model has been extended to include all modeling concepts of basic E-R with additional concepts: set-subset relationships (sub-classes/super-classes), specialization/generalization, categories, and attribute inheritance. The resulting model is called the Enhanced E-R or Extended ER (E2R or EER) model. It is used to model applications more completely and accurately if needed.

Data model: formal description of data structures (organization) and their relationships together with data operators (manipulation) applied to data structures.

2.3 Other terms commonly used in this document

Scheme (also referred to as pattern) refers to a set/class of models that are based on the same "principles" and thereby sharing the same essential and global characteristics as well as structuring/cohesive elements.

Routing Scheme: refers to a set/class of routing models that are based on the same "principles" and thereby sharing the same essential and global characteristics as well as structuring/cohesive elements.

Identifier: identifiers are unambiguously assigned from a value space to nodes. Identifiers can either follow the topology, thus be topology-dependent (in this case, they are referred to as locators and can be used directly by the routing and the forwarding function without requiring resolution) or not, thus be topology-independent (in this case, they are referred to as names or simply identifiers).

Locator: identifies a location in an internetwork. Nodes and endpoints are assigned locators. A node is assigned only one locator. Locators identify nodes by specifying “where” the node is positioned in the network. Locators do not specify a path to the node. An endpoint can be assigned more than one locator so that a locator might appear in more than one location of an internetwork. A locator can take the form of a topology dependent label (flat and unstructured), a topologically assigned address (structured), or a coordinate (structure determined by the geometric space).

3. Functional Model

In the early design phase, functional modeling refers to a methodology (part of the architectural specification process) aiming to systematically identifying, describing, and relating the functions a system must perform (thus the functions that need to be included in the system) in order to meet its objectives. Functional modeling does not address how these functions will be performed but instead deals with i) ***identification***: the specification of the top-level functions that need to be performed by the system and their decomposition into sub-functions together with the definition of their inputs/outputs, and their various interfaces, ii) ***spatial distribution***: where theses functions need to be performed (space); iii) ***temporal distribution***: how often they need to be performed (time); iv) under what operational context and environmental conditions.

The basic idea underlying functional modeling is the following: the system is viewed as computing a function or, more generally, as solving an information processing problem. For what concerns the decomposition, functional modeling proceeds by systematically describing the automated processing that a complex system must perform to transform available inputs to the desired outputs. Indeed, functional modeling assumes that such processing can be explained by iteratively decomposing the corresponding complex function into a set of simpler functions (sub-functions) that are computed by an organized sub-system up to the level of atomic functions. The expectation is that when this type of decomposition is performed, the sub-functions that are defined will be simpler than the original functions.

There are four elements to be addressed by functional modeling approach: i) ***hierarchical decomposition*** of the functions in which there is a top-level function (or top-level functional area) for the system. The top-level function is partitioned into a set of sub-functions that use the same inputs and produce the same outputs as the top-level function. Each of these sub-functions can then be partitioned further, with the decomposition process continuing as often as it is useful, ii) ***functional block diagrams*** can represent the information flow among the functions within any portion of the functional decomposition. As the first and subsequent functional decompositions are examined, it is common for one function to produce outputs that are not useful outside the boundaries of the system. These outputs are needed by other functions in order to produce the needed and expected external outputs, iii) ***processing instructions*** that contain the needed information for the functions to transform the inputs to the outputs, and iv) ***control flow*** that sequences the termination and activation of the functions so that the process is both efficient and effective.

3.1 Hierarchical Decomposition of the Routing Functional Area

Following the definition provided by [Buede00], a function is a transformation process that changes inputs into outputs, the **Routing** or **Route functional area** is defined as the local process of determining (computing) and deciding (selecting) for any destination node, the next node along a loop-free routing path such that the traffic directed to each destination will follow to reach their respective destination. From this definition, the routing function of a system can be modeled by a single, top-level area or top-level function that can be decomposed into a hierarchy of sub-areas or sub-functions. The following sub-sections provide an outline of the routing functional sub-functions description following the hierarchical decomposition of the routing functional area (see Fig.1).

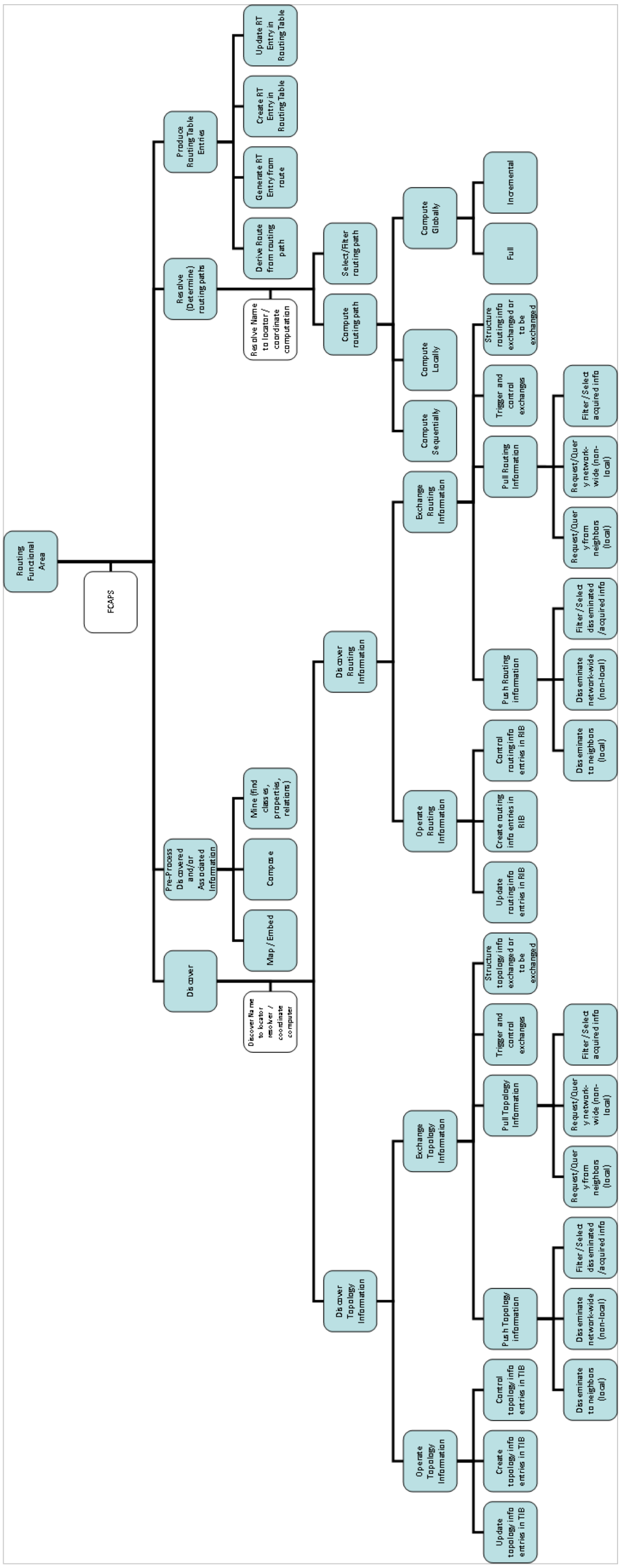

Routing Functional Area
FCAPS
Produce Routing Table Entries
Update RT Entry in Routing Table
Create RT Entry in Routing Table
Generate RT Entry from route
Derive Route from routing path
Resolve (Determine) routing paths
Resolve Name to locator / coordinate computation
Select/Filter routing path
Compute routing path
Compute Globally
Incremental
Full
Compute Locally
Compute Sequentially
Structure routing info exchanged or to be exchanged
Trigger and control exchanges
Pull Routing Information
Filter/Select acquired info
Request/Query network-wide (non-local)
Request/Query from neighbors (local)
Exchange Routing Information
Discover Routing Information
Push Routing Information
Filter/Select disseminated /acquired info
Disseminate network-wide (non-local)
Disseminate to neighbors (local)
Operate Routing Information
Control routing info entries in RIB
Create routing info entries in RIB
Update routing info entries in RIB
Pre-Process Discovered and/or Associated Information
Mine (find classes, properties, relations)
Compose
Map / Embed
Discover
Discover Name to locator resolver / coordinate computer
Discover Topology Information
Exchange Topology Information
Structure topology info exchanged or to be exchanged
Trigger and control exchanges
Pull Topology Information
Filter/Select acquired info
Request/Query network-wide (non-local)
Request/Query from neighbors (local)
Push Topology information
Filter/Select disseminated /acquired info
Disseminate network-wide (non-local)
Disseminate to neighbors (local)
Operate Topology Information
Control topology info entries in TIB
Create topology info entries in TIB
Update topology info entries in TIB

Fig.1 Hierarchical decomposition of the routing functional area

3.1.1 Discover Function

A common approach for decomposing the ''Discover'' (D) function (also referred to as knowledge or information acquisition function) is to segment this function with respect to specialization following the processing and exchange of two main class of information, i.e., topology and routing information:

- o **Routing information discovery** where routing information refers to (non-local) distances (or information to derive these distances) and/or routes (or route segments) together with their attributes. Note that routing information can also be derived from spatio-temporal properties of traffic flows.

- o **Topology information discovery** where topology information refers to all information related to local and remote interfaces (and their properties), incident links (and their properties), nodes adjacent to incident links (and their properties) as well as non-local environment including remote links and (abstract) nodes.

Following this decomposition, one could then decompose both the Discover Routing Information and the Discovery Topology Information into Operate Discovered Information and Exchanged Discovered Information.

3.1.1.1 Discover Routing Information Function

The Discover Routing Information function is then structured as follows

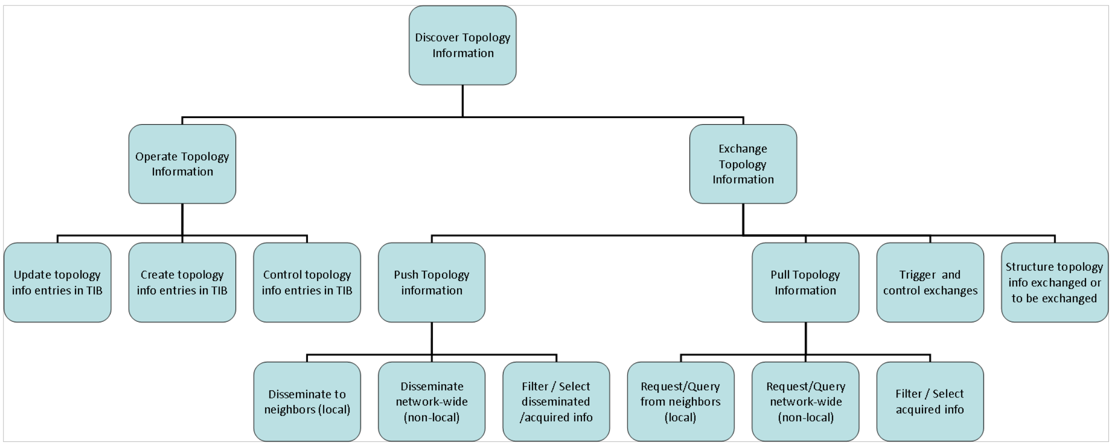

Discover Topology Information
Operate Topology Information
Exchange Topology Information
Update topology info entries in TIB
Create topology info entries in TIB
Control topology info entries in TIB
Push Topology information
Pull Topology Information
Trigger and control exchanges
Structure topology info exchanged or to be exchanged
Disseminate to neighbors (local)
Disseminate network-wide (non-local)
Filter / Select disseminated /acquired info
Request/Query from neighbors (local)
Request/Query network-wide (non-local)
Filter / Select acquired info

Fig.2: Discover Routing Information functional decomposition

i) Operate routing information sub-function sub-divides into:

- o **Create routing information entries** in Routing Information Base (RIB)
- o **Update routing information entries** in Routing Information Base (RIB)
- o **Control routing information entries** of Routing Information Base (RIB)

ii) Exchange routing information sub-function sub-divides into:

- o **Push routing information** which includes disseminate routing information to neighbors (local), disseminate routing information network-wide (non-local) but also filtering/selection of the disseminated/acquired routing information
- o **Pull routing information** which includes query/request from neighbors (local), query/request from network (non-local) but also filtering/selection of the pulled/acquired information
- o **Trigger and control exchanges** (push/pull) of routing information
- o **Structure routing information** exchanged or to be exchanged (syntax function)

Note that the distinction between local (or neighbor) and network (or non-local) discovery is commonly performed and applies to routing information:

- o **Local discovery** (also referred to as neighbor discovery): functionality enabling the acquisition/dissemination of knowledge about the local environment (neighborhood) to local entities including local and remote interfaces (and their properties), incident links (and their properties), and nodes adjacent to incident links (and their properties).
- o **Remote/non-local discovery** (also referred to as network discovery): functionality enabling the acquisition/dissemination of knowledge about the non-local environment from/to remote entities including remote links/nodes, paths and/or distances to reachable destinations.

3.1.1.2 Discover Topology Information Function

The Discover Topology Information function is then structured as follows

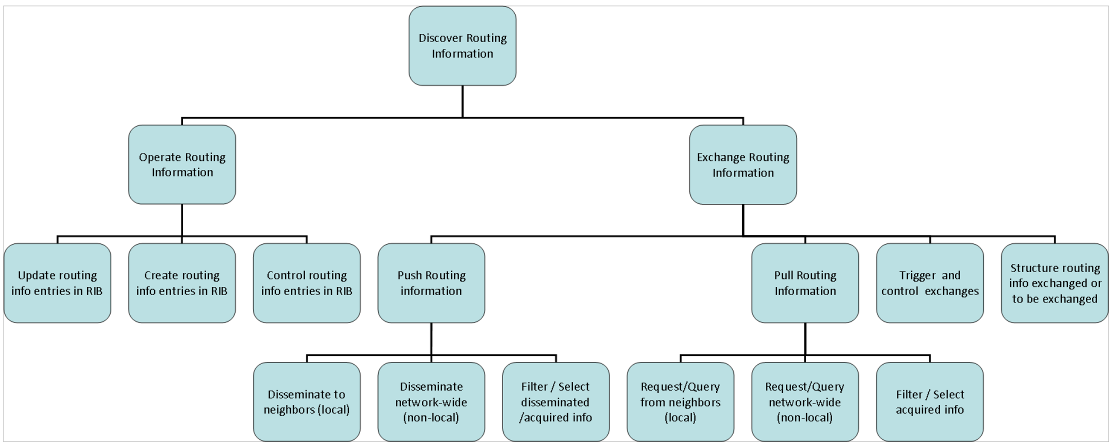

Discover Routing Information
Operate Routing Information
Exchange Routing Information
Update routing info entries in RIB
Create routing info entries in RIB
Control routing info entries in RIB
Push Routing information
Pull Routing Information
Trigger and control exchanges
Structure routing info exchanged or to be exchanged
Disseminate to neighbors (local)
Disseminate network-wide (non-local)
Filter / Select disseminated /acquired info
Request/Query from neighbors (local)
Request/Query network-wide (non-local)
Filter / Select acquired info

Fig.3 Discover Topology Information functional decomposition

i) Operate topology information sub-function sub-divides into:

- o **Create topology information entries** in Topology Information Base (TIB)
- o **Update topology information entries** in Topology Information Base (TIB)
- o **Control topology information entries** of Topology Information Base (TIB)

ii) Exchange topology information sub-function sub-divides into:

- o **Push topology information** which includes disseminate topology information to neighbors (local), disseminate topology information network-wide (non-local) but also filtering/ selection of the disseminated/acquired topology information
- o **Pull topology information** which includes query/request from neighbors (local), query/request from network (non-local) but also filtering/selection of the pulled/acquired topology information
- o **Trigger and control exchanges** (push/pull) of topology information
- o **Structure topology information** exchanged or to be exchanged (syntax function)

As for the Discover Routing Information, the distinction between local (or neighbor) and network (or non-local) discovery is commonly performed and applies also to topology information:

- o **Local discovery** (also referred to as neighbor discovery): functionality enabling the acquisition/dissemination of knowledge about the local environment (neighborhood) to local entities including local and remote interfaces (and their properties), incident links (and their properties), and nodes adjacent to incident links (and their properties).
- o **Remote discovery** (also referred to as network discovery): functionality enabling the acquisition/dissemination of knowledge about the non-local environment from/to remote entities including remote links/nodes, paths and/or distances to reachable destinations.

3.1.2 Pre-Process Function

Pre-processing consists in structuring and/or analyzing the topology information and/or routing information using a combination of the following operations:

- o **Embedding/mapping**: given metric spaces (X,d_X) and (Y,d_Y), where X and Y are spaces, and d_X and d_Y are distance functions, a mapping function $f: X \rightarrow Y, x \rightarrow y=f(x)$ is called an embedding. An embedding is called distance-preserving if $\forall x, y \in X, d_X(x,y) = d_Y(f(x),f(y))$

- o **Composition**: this function combines topology and/or routing information so as to build more complex topology and/or routing information (called structures).

- o **Mining**: includes all functions enabling to find i) (hidden) relationships between routing information, between topology information as well as between routing and topology information, ii) features/properties characterizing routing and topology information, and iii) classes in this information.

3.1.3 Resolve Routing Path function

The Resolve Routing Path function determines (hence it is also referred to as Determine Routing Path function) the method to actually obtain the routing path by means of:

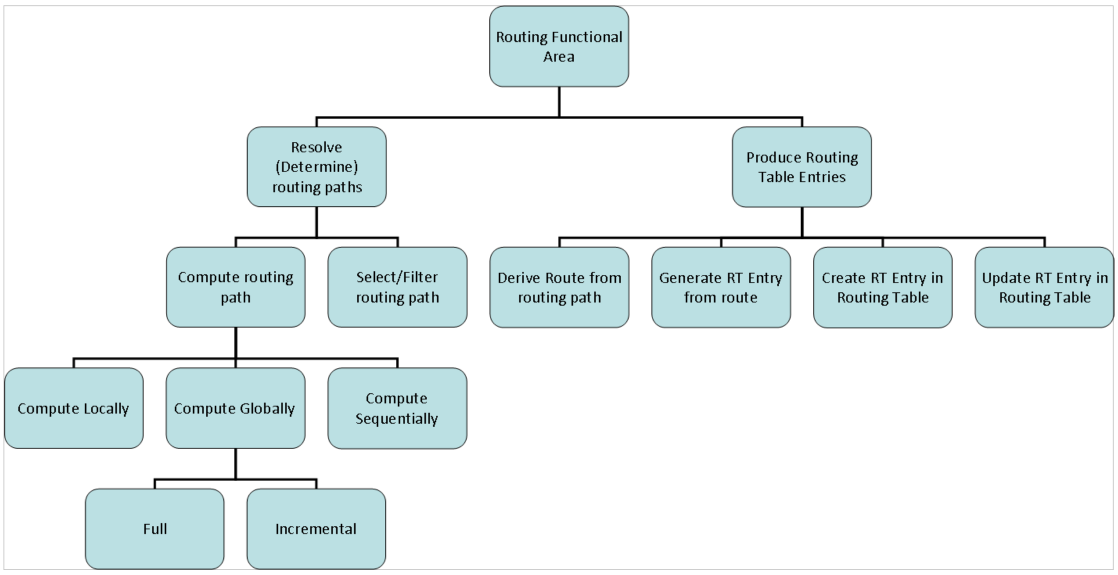

Routing Functional Area
Resolve (Determine) routing paths
Produce Routing Table Entries
Compute routing path
Select/Filter routing path
Derive Route from routing path
Generate RT Entry from route
Create RT Entry in Routing Table
Update RT Entry in Routing Table
Compute Locally
Compute Globally
Compute Sequentially
Full
Incremental

Fig.4: Resolve Routing Paths and Produce Routing Table Entries functional decomposition

- o **Computation**: function applied to (structured) routing information units and/or (structured) topology information units and that produces routing paths from which routing table entries can be derived. Computation can be seen as the operation of finding the routing path that minimizes/maximizes a (multi-)constrained (multi-)objective function. The computation function can be further sub-divided as follows:
  - **Compute Global-Full** function: computation is based on global knowledge of the graph; upon routing information update the full routing table is recomputed.
  - **Compute Global-Incremental** function: computation is based on global knowledge of the graph; upon routing information update only the affected routing table entries are recomputed.
  - **Compute Local** function: computation is based on local knowledge (neighbors related information) possibly complemented by a partial knowledge of remote/non-local parts of the network graph properties.
  - **Compute Sequential** function: computation is based on hop-by-hop/serial information propagated by local neighbors along either a given spatial trajectory.

- o **Selection**: either by enforcing selection rules, by applying filters, and/or by multi-criteria decision on a set of routing information units (typically routing paths with associated attributes). By means of this processing, a limited number of routing paths is selected from which routing table entries can be derived.

- **Select/Filter Paths per destination** function: routing paths are selected per destination, e.g., path-vector based routing protocols.
- **Select/Filter (logical) Ports per destination** function: (logical) ports are selected per destination, e.g., spanning-tree based routing protocols.

3.1.4 Produce Routing Table Entry Function

The Produce Routing Table Entry function derives a route from the computed and/or selected routing paths and generates the corresponding routing table entry from the (selected) route, together with the creation of a new entry or the update of an existing routing table entry in the Routing Table (RT). Each forwarding table entry is then subsequently derived from a sub-set of one or more routing table entries.

3.1.5 Associated Functions

The set of associated functions include i) all FCAPS associated to the routing functionality (system configuration, administration, protection, etc.), ii) the transfer of routing table entries: a mechanism allowing to export the routing table entries towards the forwarding engine component and iii) trigger or poll for renewal/update control/routing behavior of a node based on external or internal events.

Moreover, "addressing and reachability" information can be derived by means of distribution/discovery function part of the "routing" itself or by means of an associated resolution system dictionary (white boxes). Indeed, we assume that the routing functional area operates on locators (see definition in Section 2). Henceforth, a resolution function is associated to it that can operate on reachability information only (e.g., Host Identity Protocol where IP addresses function as pure locators; the applications use Host Identifiers to name peer hosts instead of using IP addresses) or on the topology/routing information itself (name-independent compact routing is an example of locator/label-based routing augmented with a identifier-to-locator function).

- **Identification function**: assigns identifiers to nodes. These names can be either topology-dependent (locators) or topology-independent (names); a locator can take the form of a label, a topology-dependent address or a coordinate.

- **Resolution function**: performs translation, conversion, or mapping from the name of the destination to its associated locator.

- **Location function**: the functionality allowing destinations to be located by means of the resolution function.

3.2 Functional Block Diagrams

This section makes use of the (Enhanced) Functional Flow Block Diagram (FFBD) to specify the routing functional model of the routing system architecture. For this purpose, we consider that each network node implements the routing functionality, which we call the Route function. Following the hierarchical decomposition of the Route functional area depicted in Fig.1, this function is decomposed iteratively in a set of (sub-)functions, which have a sequential relationship between them. In the following sections we provide a definition of each of these functions, and then we describe its sequential relationship using EFFBD diagrams.

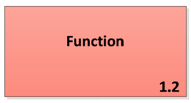
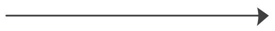
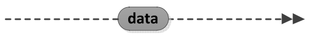
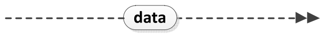
3.2.1 FFBD and EFFBD Overview

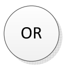
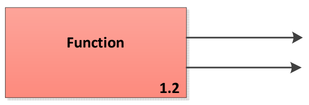
A Functional Flow Block Diagram (FFBD) provides a multi-level decomposition of the system's function with a control structure that dictates the order in which the (sub-)function can be executed at each level of the decomposition. The FFBD presents thus the logical sequencing of the same (sub-)functions as those identified through functional decomposition by displaying them in their logical, sequential relationship. The decomposition of the function into the sequence of activities is carried out by asking the question "WHAT" needs to be done to perform the particular function (but does not assume a particular on how the function shall be performed). FFBD is a commonly used tool in functional modeling to define the functional level and the sequences of activities. The purpose of the FFBD is thus to show the (sub-)functions that a system is to perform, the order in which they are to be performed (logical sequence), and their relationships. The order of execution is specified from the set of available control constructs. The basic FFBD syntax includes four types of control structure: series, concurrent/parallel (AND), selection (OR), and multi-exit function together with the completion criterion (IF-THEN-ELSE). Note that in order to include iteration (IT symbol), loop (LP symbol), and replication (RP symbol), the extended version of FFDB shall be used because basic FFDB syntax only allows to depict 4 types of control structures being series, concurrency, selection and multi-exit). The FFBD does not contain any semantics relating to the flow of information between functions, and therefore does not distinguish between triggering and non-triggering inputs.

Enhanced FFBD (EFFBD) displays the control dimension of the functional model in an FFBD format with an information flow overlay to effectively capture information dependencies. Thus, Enhanced FFBD represents: (1) functions, (2) control flows, and (3) data flows. The logic constructs allow indicating the control structure and sequencing relationships of all functions accomplished by the system being analyzed and specified. When displaying the information flow as an overlay on the control flow, the EFFBD graphically distinguishes between triggering and non-triggering inputs. Triggering input is required before a function can begin execution. Therefore, triggers are actually information with control implications. The Enhanced FFBD specification of a system is complete enough that it is executable as a discrete event model, providing the capability of dynamic, as well as static, validation. A fundamental rule in the interpretation of an EFFBD specification is that a function must be enabled (by completion of the function(s) preceding it in the control construct) and triggered (if any information input to it is identified as a trigger) before it can execute. This provides maximum flexibility to use either control constructs or data triggers (or a combination of both) to specify execution conditions for individual system functions. Moreover, by augmenting the EFFBD with function duration estimates/budgets and resource constraints and utilization, trade studies can be performed using dynamic simulation outputs.

Table 1 summarizes the main symbols used in EFFBD (throughout this document). Annex 1 provides more details on functional FFBD and EFFBD diagramic modeling techniques. It is important to underline that the lack of formal function definition makes specification errors to remain possibly undetected. Formalization can be used to detect and correct subtle system errors during the specification phase.

Table 1: EFFBD symbols and description

Name	Symbol	Description
Function		A function is a unit of actions or activities to be performed. The symbol contains the function's name (an action verb plus a noun phrase) and the function's number (a unique decimal number). Separate from the symbol, a function has also associated a text (the function's definition) describing what the function does.
Control (Enabling) Line		A control (enabling) line connecting two functions indicates that the function at the right is enabled by control by the function at the left, i.e., only when the function at the left has completed the execution, the function at the right will be allowed to begin the execution.
Data Triggering Line *		A data triggering line connecting two functions indicates that the function at the right is triggered by a data produced by the function at the left, i.e., only when the function at the left has produced the data, the function at the right will be allowed to begin the execution. Note that the data triggering line has to be used together with the control (enabling) line, i.e., there is a function at the left (the same or another) that enables by control the function at the right. The symbol contains the name of the triggering data item.
Data (Non-Triggering) Line		A data (non-triggering) line connecting two functions indicates that the function at the right uses a data produced by the

*		function at the left but without triggering it, i.e., the function at the left produces a data before the function at the right be allowed to begin the execution. Note that the data (non-triggering) line has to be used together with the control (enabling) line, i.e., there is a function at the left (the same or another) that enables by control the function at the right. The symbol contains the name of the data item.
AND		The "AND" constructor is a condition in which all preceding or succeeding paths are required: it may contain one input with multiple outputs (i.e., the input enables in parallel all the outputs) or multiple inputs with one output (i.e., the output is enabled when all inputs enable it). It cannot contain multiple inputs and outputs combined.
Exclusive OR		The "Exclusive OR" constructor is a condition in which one of multiple preceding or succeeding paths is required, but not all: it may contain one input with multiple outputs (i.e., the input enables only a single output) or multiple inputs with one output (i.e., the output is enabled when one of the inputs enable it). It cannot contain multiple inputs and outputs combined.
Multi-Exit Constructor		A multi-exit constructor indicates that a function may enable multiple (one, two, etc.) functions according to some exit conditions.

(*Note that for practical reasons we are using a slightly different symbols than in traditional EFFBD)

3.2.2 Route (R) function

We consider that each network node implements the routing functionality, which we call the Route function. The Route (R) function (applied at each node belonging to the routing system) is defined as the process of determining (computing) and deciding (selecting) for any destination node, the next node along a routing path that the traffic directed to each destination will follow to reach their destination. For this purpose, routing includes the discovery of information about the network topology, the distance, and/or the routing paths as received from other nodes through queries (pull) or dissemination (push), the processing of this information, the computation and/or the selection of routing paths as well as the derivation of the next-hop node. The results are stored in the Routing Table (RT).

Following the hierarchical decomposition depicted in Fig.1, the Route (R) function comprises the following sub-functions:

Discover (D): this function that plays a fundamental role in distributed routing comprises:

- o **Discover Topology Information (DTI)** function (box_1), where topology information includes all information related to local and remote interfaces (and their properties), incident links (and their properties), nodes adjacent to incident links (and their properties) as well as non-local environment including remote links and (abstract) nodes. The topology information exchanged is structured in units, referred to as topology information units (TIU).
- o **Discover Routing Information (DRI)** function (box_2), where routing information includes non-local distances (or information to derive these distances) and/or paths (segments) together with their attributes. The routing information exchanged is structured in units, referred to as routing information units (RIU).

Pre-process Discovered and Associated Information (PDAI) (box_3): this function structures and analyzes the discovered information about topology and routing (e.g., distance to destination, paths to destination) by means of advanced techniques that permit: to obtain patterns, features, properties and hidden relationships in the information; to derive complex information from simple information; to transform a coordinate space to another space with new properties.

Determine Routing Path (DRP) also referred to as "Resolve Routing Path" function (box_4): computes routing paths from topology and/or routing information (possibly pre-processed by means of the pre-processing function) or selects routing paths based on some rules, filters, and/or selection criteria applied on the received routing information or selects routing paths based on some criteria applied on a sub-set of computed routing paths.

Produce Routing Table Entry (PRTE) (box_5): generates or derives the Routing Table Entries (RTEs), i.e., routes from the computed and/or selected routing paths. Each Forwarding Table Entry (FTE) is in turn generated or derived from a sub-set of one or more RTEs.

Resolve Name to Locator / Coordinate (RNLC) (box_6): performs association of node locators to node identifiers (also referred to as names) and the resolution of node identifiers to its associated locators. The information about node identifiers is structured in Identifier Information Units (IIUs), the information about node locators is structured in Locator Information Units (LIUs), and the information about identifier-locator(s) bindings is structured in Identifier-Locator Bindings (ILBs). This function is mainly seen as “external” to the routing functional area performing at each node since the association of locators to identifiers and the maintenance of the identifier-locator(s) binding information is performed by a dedicated resolving entity. At each node, this function is responsible for issuing the resolution requests with an external resolving entity and/or with an internal database where the identifier-locator(s) binding information is stored (“cached”), processing the responses and storing the identifier-to-locator binding information. The ILB Information Base (IIB) is the database where ILBs are maintained.

The sequential relationship between these functions using an EFFBD diagram is shown in Fig. 5.

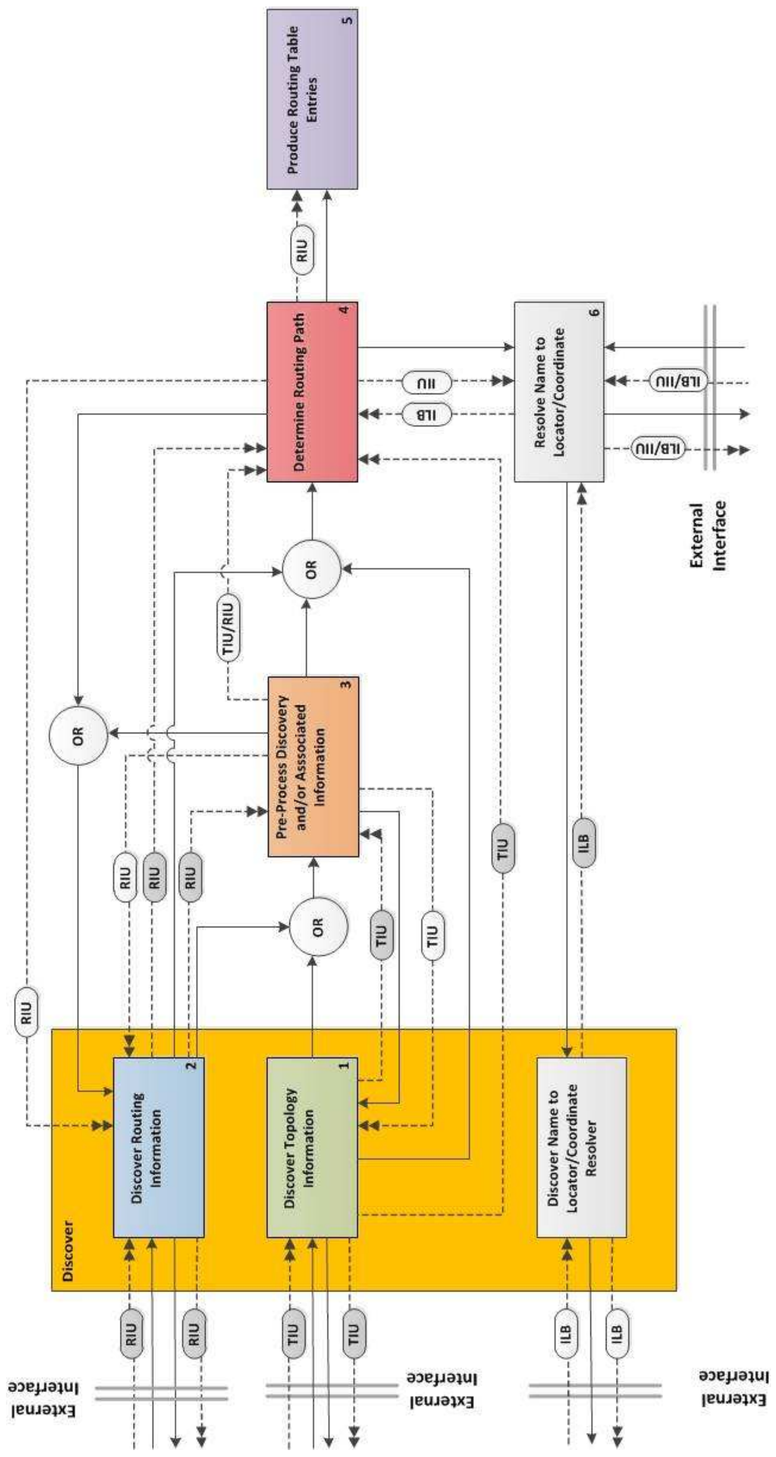

Produce Routing Table Entries 5
Determine Routing Path 4
Resolve Name to Locator/Coordinate 6
Pre-Process Discovery and/or Asssociated Information 3
Discover Routing Information 2
Discover Topology Information 1
Discover Name to Locator/Coordinate Resolver
Discover
OR
OR
OR
RIU
TIU/RIU
TIU
ILB
IIU
ILB/IIU
External Interface

Fig.5: Sequential relationship between sub-functions of the routing (R) function

3.2.3 Discover (D) function

Following the definition provided in Section 3.2.2, the Discover (D) function can be decomposed into:

- **Discover Topology Information (DTI)** (box_1): the acquisition, dissemination and maintenance of information about the topology, i.e., states and properties of interfaces, links and nodes. The information about topology, structured in Topology Information Units (TIUs), is obtained through the exchange of information with other (neighbor and remote) nodes, after possible filtering/selection of the information to be disseminated or the information being acquired. The Topology Information Base (TIB) is the database where TIUs are maintained.

- **Discover Routing Information (DRI)** (box_2): the acquisition, dissemination and maintenance of information about the routing paths and/or distances to reachable destinations. The information about routing, structured in Routing Information Units (RIUs), is obtained through the exchange of information with other (neighbor and remote) nodes, after possible filtering/selection of the information to be disseminated or the information being acquired. The Routing Information Base (RIB) is the database where RIUs are maintained.

3.2.3.1 Discover Topology Information (DTI) function

The **Discover Topology Information** (DTI) function is decomposed into the following functions (Fig.1):

- **Exchange Topology Information** function (box_1.1): Activation (trigger) and control of the operations for the acquisition, dissemination and maintenance of discovered Topology Information Units (TIUs), i.e., this function initiates the sending of TIUs to other nodes, the processing of received TIUs and related operations in the Topology Information Base (TIB), for the processing of TIUs in order to structure (syntax) the topology information. This function can be activated by changes in the TIB, by internal timers (e.g., aging of TIB entries) or others. The Exchange Topology Information function is further decomposed as follows:

  - **Push Topology information** function (box_1.1.1): includes the dissemination to neighbors (local), the network-wide (non-local) dissemination but also filtering/selection of the disseminated/acquired information. The Push Topology Information function comprises a communication function and a communication control function. The communication function comprises Send and Receive sub-functions. Sending and receiving messages comprising the Topology Information Units (TUIs) to/from other nodes through the external interfaces. This function also includes the operations related to message scheduling and management of input and output queues (storage, sending priorities, discarding rules, etc.) as well as the classification of the received messages.

  - **Pull topology information** function (box_1.1.2): includes query/request from neighbors (local), query/request from network (non-local) but also filtering/selection of the pulled/acquired information. The Push Topology Information comprises a communication function and a communication control function. The communication function comprises Send and Receive sub-functions. Sending and reception of messages comprising the Topology Information Units (TIUs) to/from other nodes through the external interfaces. This function also includes the operations related to message scheduling and management of input and output queues (storage, sending priorities, discarding rules, etc.) as well as the classification of the received messages.

  - **Trigger and control exchanges** (push/pull) function (box_1.1.3): the purpose of this function is to initiate and/or activate the exchange (push/pull) of topology information units. The exchange control regulates the rate of exchanges of topology information units and the associated rate adaptation.

- **Structure topology information** function (box_1.1.4): the role of this syntax function is to structure the received topology information units (acquisition) or the topology information units to be exchanged (dissemination). For instance, include multiple topology information units as part of single message, and segment a given topology information unit into multiple messages, etc.

o **Operate Topology Information Base (TIB)** function (box 1.2): creation, maintenance (update) and use of the topology information entries stored in the Topology Information Base (TIB). This function also includes the control to access the data entries, the enforcement of data integrity, the management of concurrency, the recovery and restoration of the database after failures, and the maintenance of the database security. The Operate TIB function is further sub-divided into the following sub-functions:
  - **Create Topology Information Entry** in TIB (box 1.2.1): this function creates entry in the TIB when no prior entry in the database exists for the corresponding topology information.
  - **Update Topology Information Entry** in TIB (box 1.2.2): this function updates entry of the TIB (resulting from a change of information stored as part of an existing entry.
  - **Control Topology Information Entry** of TIB (box 1.2.3): this function controls access to the database entries, the enforcement of the integrity of the database entries, the management of concurrent access to database entries, the recovery and restoration of the database entries after database failures, and the maintenance of the database security.

The sequential relationship between these functions using an EFFBD diagram is shown in Fig. 5.1, 5.2 and 5.3.

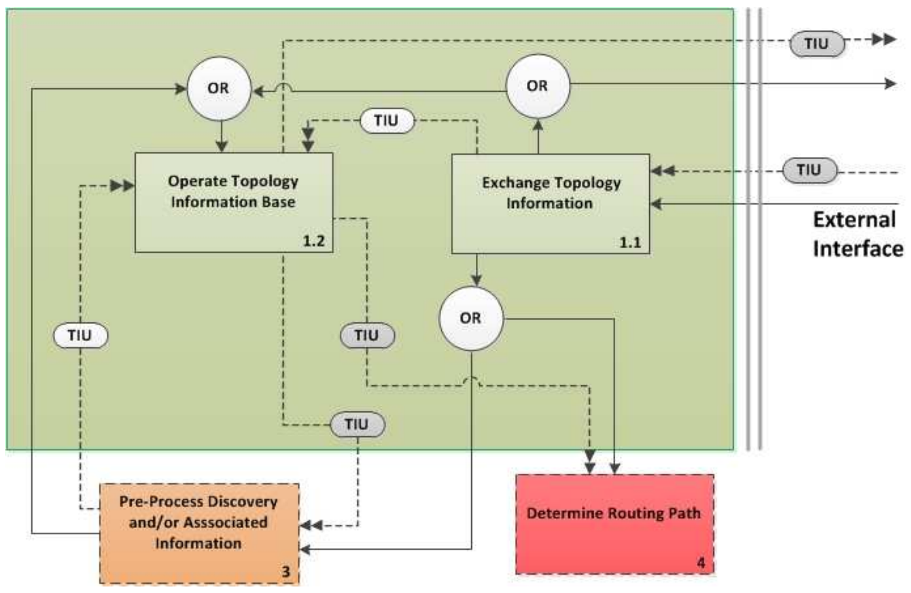

TIU
OR
OR
TIU
TIU
Operate Topology Information Base
1.2
Exchange Topology Information
1.1
External Interface
OR
TIU
TIU
TIU
Pre-Process Discovery and/or Associated Information
3
Determine Routing Path
4

Fig. 5.1: EFFBD of the Discover Topology Information function.

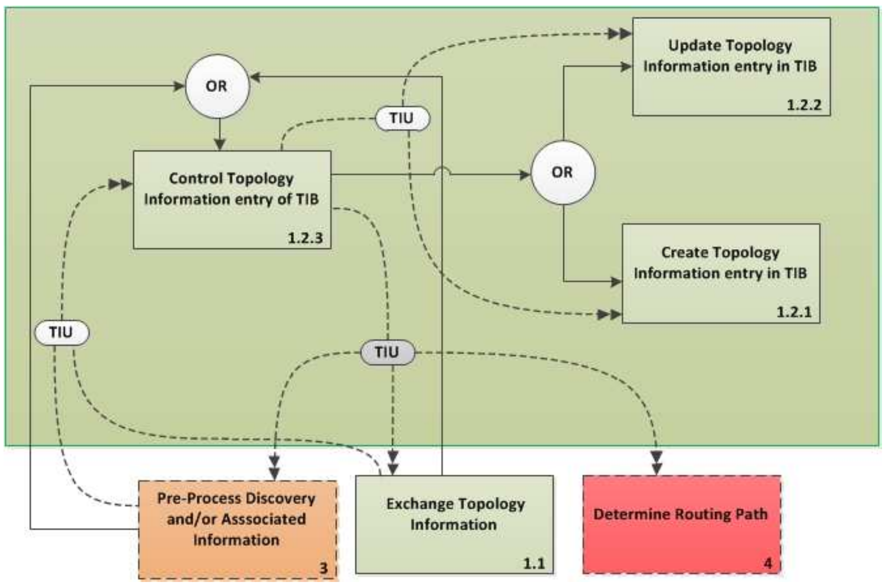

OR
Update Topology Information entry in TIB 1.2.2
TIU
Control Topology Information entry of TIB 1.2.3
OR
Create Topology Information entry in TIB 1.2.1
TIU
TIU
Pre-Process Discovery and/or Assosciated Information 3
Exchange Topology Information 1.1
Determine Routing Path 4

Fig. 5.2: EFFBD of the Operate Topology Information Base Function

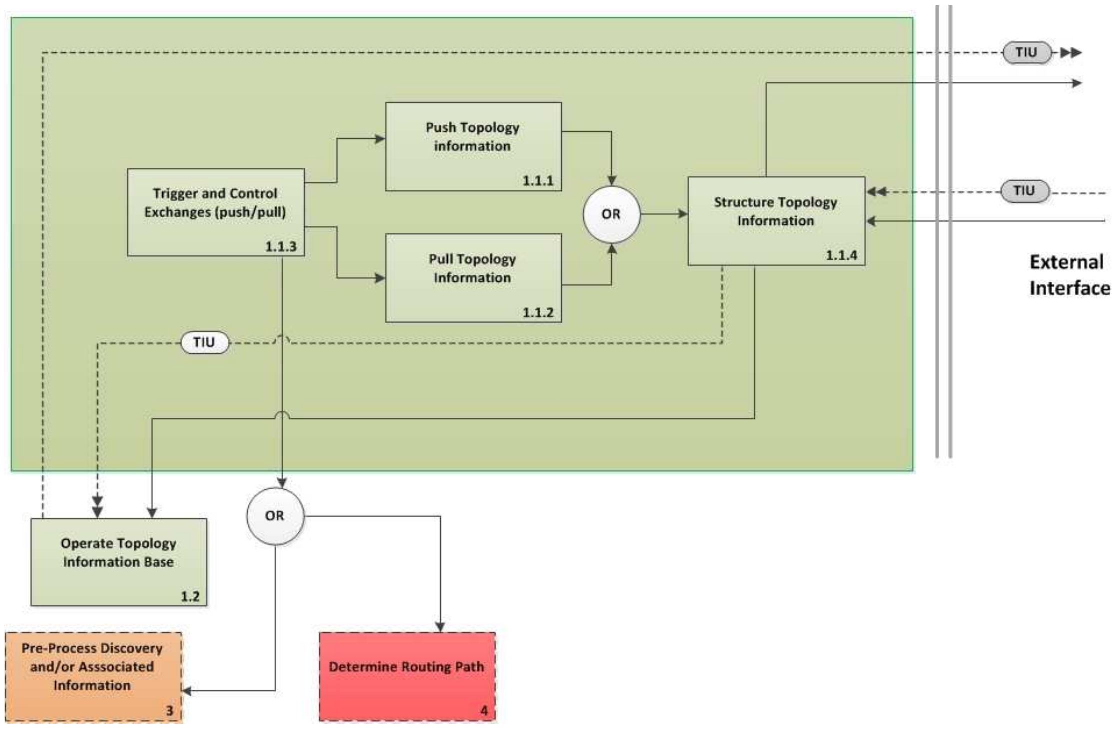

TIU
Push Topology information 1.1.1
Trigger and Control Exchanges (push/pull) 1.1.3
OR
Structure Topology Information 1.1.4
TIU
Pull Topology Information 1.1.2
External Interface
TIU
OR
Operate Topology Information Base 1.2
Pre-Process Discovery and/or Asssociated Information 3
Determine Routing Path 4

Fig.5.3: EFFBD of the Exchange Topology Information Function

The DTI function is activated/triggered by the following functions (Fig. 5 and 5.1):

- o The DTI function of other nodes, when topology information units are received from other nodes through the external interfaces.
- o By the PDAI function, when the result of its processing (composition) is be stored in the TIB, or when this function requests processing of topology information stored in the TIB.
- o By certain changes in the TIB, internal timers (aging of TIB entries) or other maintenance operations applied to the TIB entries.

The DTI function activates/triggers the following functions (Fig. 5 and 5.1):

- o The DTI function of other nodes, when topology information units are sent to other nodes through the external interfaces.

- The PDAI function in order to process routing information units for structuring and analyzing topology information.
- The DRP function in order to compute and/or select routing paths using the topology information stored in the TIB.

3.2.3.2 Discover Routing Information (DRI) function

The **Discover Routing Information (DRI)** function (box 2) defined in Section 1.3.1 is decomposed into the following functions (Fig.1):

- **Exchange Routing Information** (box_2.1): performs activation (trigger) and control of the operations for the acquisition, dissemination and maintenance of Routing Information Units (RIUs), i.e., this function initiates the sending of RIUs to other nodes, the processing of received RIUs and the related operations in the Routing Information Base (RIB), for the processing of RIUs in order to structure (syntax) routing information. This function can be activated by changes in the RIB, by internal timers (e.g., aging of RIB entries) or others. The Exchange Routing Information function is further decomposed as follows:

  - **Push Routing Information** function (box_2.1.1): includes the dissemination to neighbors (local), the network-wide (non-local) dissemination but also filtering/selection of the disseminated/acquired routing information. The Push Topology Information comprises a communication function and a communication control function. The communication function comprises Send and Receive sub-functions. Sending and receiving messages comprising the Routing Information Units (RUIs) to/from other nodes through the external interfaces. This function also includes the operations related to message scheduling and management of input and output queues (storage, sending priorities, discarding rules, etc.) as well as the classification of the received messages.

  - **Pull Routing Information** function (box_2.1.2): includes query/request from neighbors (local), query/request from network (non-local) but also filtering/selection of the pulled/acquired information. The Push Topology Information comprises a communication function and a communication control function. The communication function comprises Send and Receive sub-functions. Sending and receiving messages comprising the Routing Information Units (RIUs) to/from other nodes through the external interfaces. This function also includes the operations related to message scheduling and management of input and output queues (storage, sending priorities, discarding rules, etc.) as well as the classification of the received messages.

  - **Trigger and control exchanges** (push/pull) function (box_2.1.3): the purpose of this function is to initiate and/or activate the exchange (push/pull) of routing information units. The exchange control regulates the rate of exchanges of routing information units and the associated rate adaptation.

  - **Structure routing information** function (box_2.1.4): the role of this syntax function is to structure the received routing information units (acquisition) or the routing information units to be exchanged (dissemination). For instance, include multiple routing information units as part of single message, and segment a given routing information unit into multiple messages, etc.

- **Operate Routing Information Base** (RIB) function (box_2.2): creation, maintenance (update) and use of the routing information entries stored in the Routing Information Base (RIB). This function also includes the control to access the data entries, the enforcement of data integrity, the management of concurrency, the recovery and restoration of the database after failures, and the maintenance of the database security. The Operate RIB function is further sub-divided into the following sub-functions:

- **Create Routing Information entry** in RIB (box_2.2.1): this function creates entry in the RIB when no prior entry in the RIB exists for the corresponding topology information.
- **Update Routing Information entry** in RIB (box_2.2.2): this function updates entry of the RIB (resulting from a change of information stored as part of an existing entry.
- **Control Routing Information entry** of RIB (box_2.2.3): this function controls access to the RIB entries, the enforcement of the integrity of the database entries, the management of concurrent access to database entries, the recovery and restoration of the database entries after database failures, and the maintenance of the database security.

The sequential relationship between these functions using an EFFBD diagram is shown in Fig.6.1, 6.2 and 6.3.

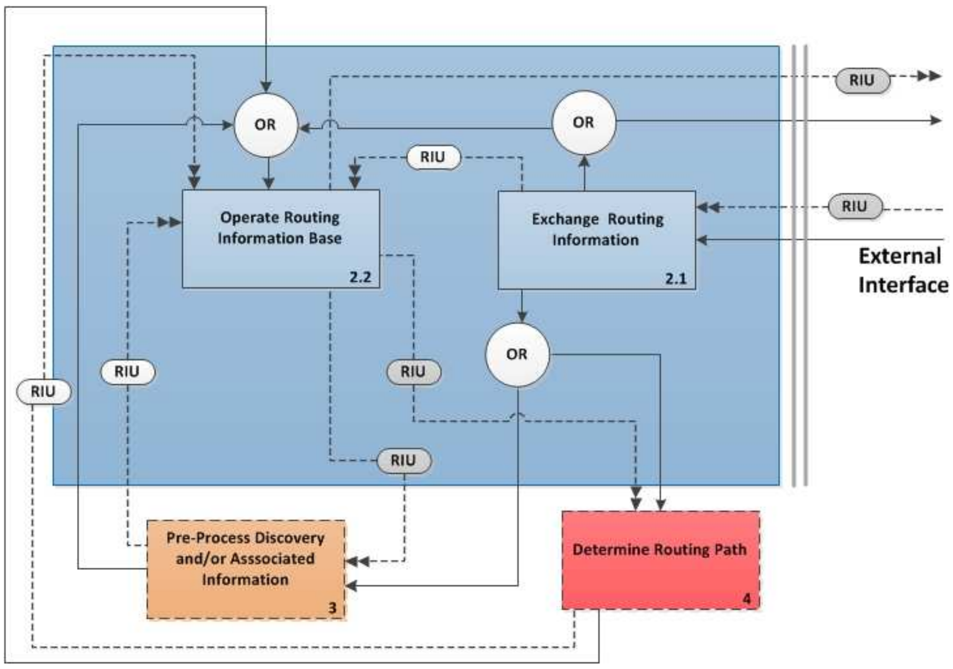

RIU
OR
OR
RIU
Operate Routing Information Base
2.2
Exchange Routing Information
2.1
RIU
External Interface
OR
RIU
RIU
RIU
RIU
Pre-Process Discovery and/or Asssociated Information
3
Determine Routing Path
4

Fig. 6.1: EFFBD of the Discover Routing Information (DRI) function.

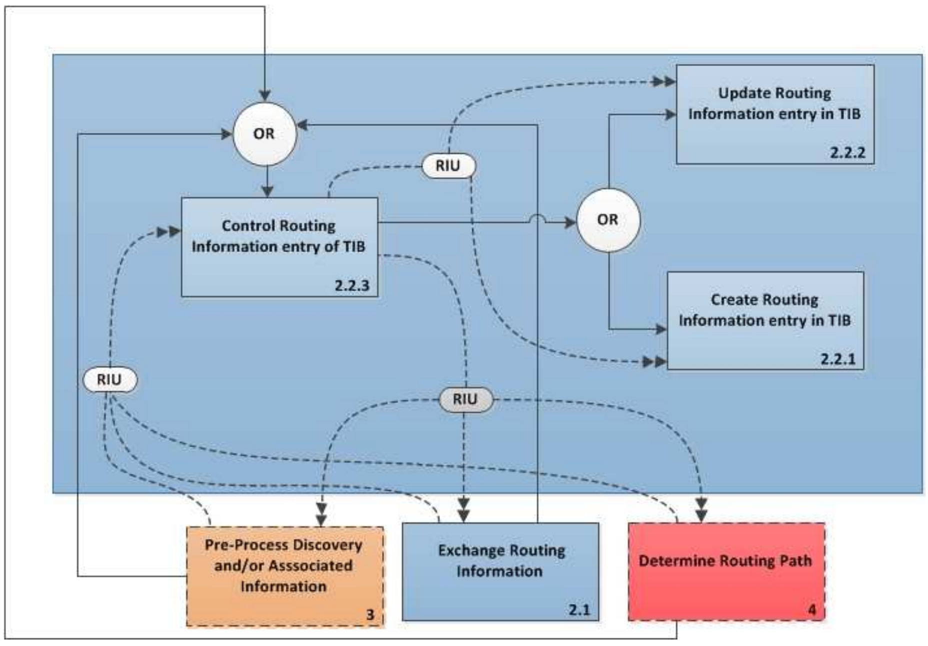

OR
Update Routing Information entry in TIB
2.2.2
RIU
Control Routing Information entry of TIB
2.2.3
OR
Create Routing Information entry in TIB
2.2.1
RIU
RIU
Pre-Process Discovery and/or Asssociated Information
3
Exchange Routing Information
2.1
Determine Routing Path
4

Fig. 6.2: EFFBD of the Operate Routing Information Base Function

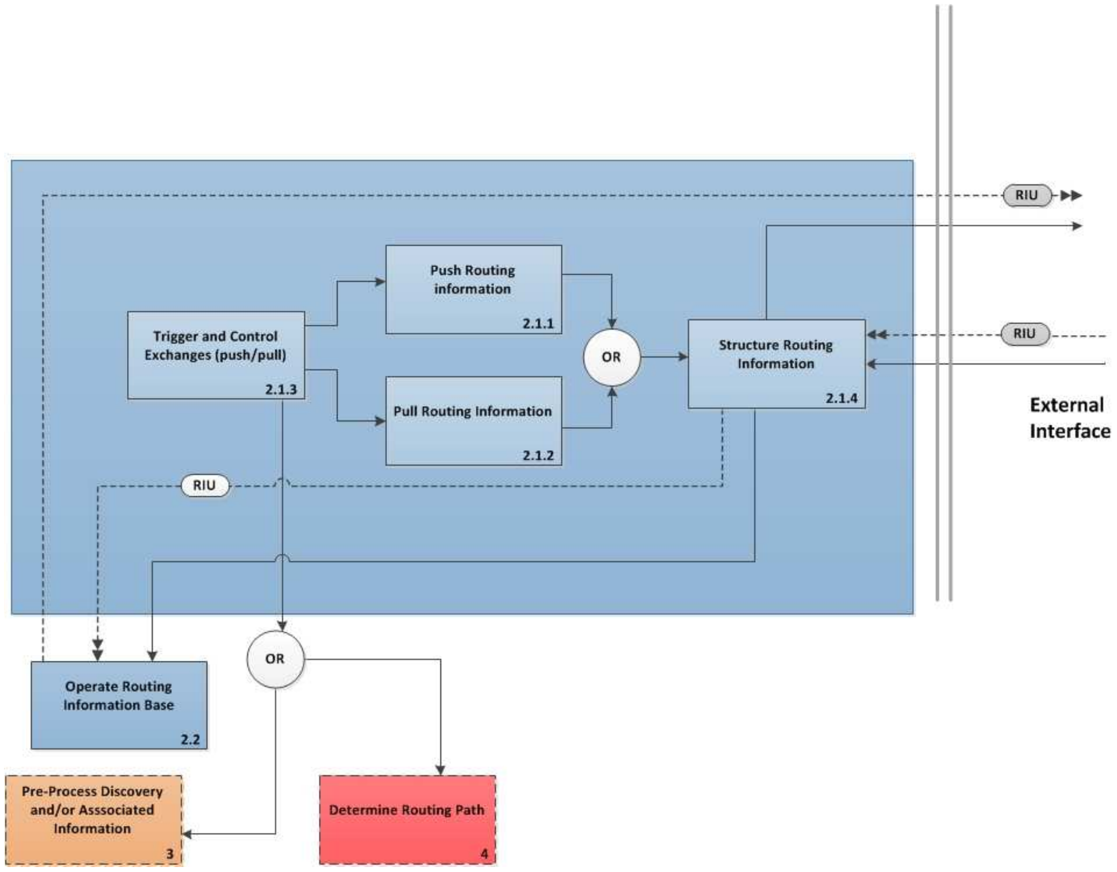

RIU
Push Routing information
2.1.1
Trigger and Control Exchanges (push/pull)
2.1.3
OR
Structure Routing Information
2.1.4
RIU
Pull Routing Information
2.1.2
External Interface
RIU
OR
Operate Routing Information Base
2.2
Pre-Process Discovery and/or Asssociated Information
3
Determine Routing Path
4

Fig.6.3: EFFBD of the Exchange Routing Information Function

The DRI function is activated/triggered by the following functions (Fig. 5 and 6.1):

- o The DRI function of other nodes, when routing information units are received from other nodes through the external interfaces.
- o By the PDAI function, when the result of its processing (composition) is be stored in the RIB, or when this function requests processing of routing information stored in the RIB.
- o By the DRP function, in order to store in the RIB the route(s) derived from the "computed" and "selected" routing path(s).
- o By certain changes in the RIB, internal timers or others.

The DRI function activates/triggers the following functions (Fig. 5 and 6.1):

- o The DRI function of other nodes, when routing information units are sent to other nodes through the external interfaces.
- o The PDAI function in order to process routing information units for structuring and analyzing routing information.
- o The DRP function in order to compute and/or select routing paths.

3.2.4 Pre-process Discovered and Associated Information (PDAI) function

Following the definition provided in Section 3.2.2, the **Pre-process Discovered and Associated Information (PDAI)** function (box_3) is in charge of structuring and analyzing the discovered information about topology and routing (e.g., distance to destination, paths to destination) by means of combination the following operations: embedding, composition, and mining. The operations enable to obtain patterns, features/properties and hidden relationships between discovered information; to derive/build complex structures from simple information; to transform a coordinate space to another space with new properties.

The PDAI function is decomposed into the following functions (see Fig.1):

- **Start Pre-Processing** (box_3.1): the purpose of this function is to retrieve the topology information from the TIB and the routing information from the RIB together with the activation and control of the operations for processing this information. This function can be activated by timers (time-driven) and/or occurrence of certain events (event-driven).
- **Mine** (box_3.2): given a set of topology and routing information units, find (hidden) relationships, features/properties, patterns and/or classes between them by applying learning and mining methods on this set of information.
- **Compose** (box_3.3): given a set of topology and routing information units, this function combines this information so as to build more complex topology and/or routing information units, called "structured" TIUs or RIUs.
- **Embed** (box_3.4): transformation of a given metric space (combination of nodal and distance information) into another metric space with new properties, which allow computing better paths. The transformation is given by a mapping function.

The sequential relationship between these functions using an EFFBD diagram is shown in Fig.7.

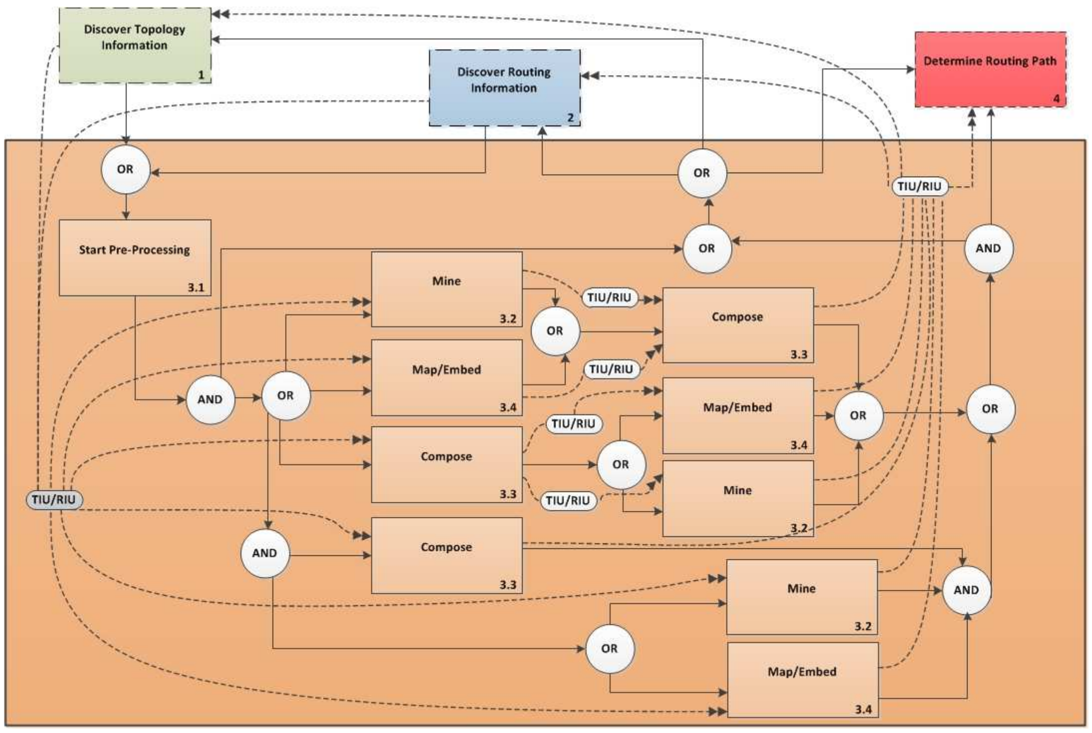


Fig.7: EFFBD of the Pre-process Discovered and Associated Information (PDAI) function

The PDAI function is enabled/triggered by the following functions (Fig.5 and Fig.7):

- By means of internal timers and others in order to process information.
- By the DTI function, when this function requires processing TIUs for structuring and analyzing topology information.
- By the DRI function, when this function requires processing RIUs for structuring and analyzing routing path information.

The PDAI function enables/triggers the following functions (Fig.5 and Fig.7):

- The DTI function for requesting TIUs from the TIB that are to be processed or for storing TIUs in the TIB that have been processed.
- The DRI function for requesting RIUs from the RIB that are to be processed or for storing RIUs in the RIB that have been processed.
- The DRP function in order to process "structured" RIUs or TIUs for obtaining new ("computed") routing paths.

3.2.5 Determine Routing Path (DRP) function

The "Determine Routing Path" (DRP) function (box_4) also referred to as "Resolve Routing Path" function is in charge of i) the computation of routing paths (called "computed" RIU) from the discovered and/or pre-processed topology information and/or routing information, and ii) the selection of routing paths (referred to as "selected" RIUs) based on some criteria applied on the received routing paths and its associated attributes (referred to as "received" RIUs) or the selection based on some criteria applied on a sub-set of computed routing paths (referred to as "computed" RIUs).

Note that these functions refer to a distributed/decentralized computation and to an adaptive routing, i.e., the routing function is capable to compute and/or select routes for any sequence of information updates arriving at given rate (in contrast to not-adaptive/static routing where routing paths are computed at provisioning time).

The task of routing path calculation can be classified from the point of view of time, space and processing. From the time perspective, the calculation can be either incremental (given an update, re-computing of only the affected routing table entries) or full/complete (all the paths are recomputed). From the spatial perspective, the calculation can be either local (paths are computed without a global network knowledge, using only information from neighbors) or global (paths are calculated using a global network knowledge). From the processing perspective, the calculation can be performed sequentially (paths are calculated one after the other) or in parallel. Remember also that the computation or selection/filtering is performed per "destination" (where destination here is to be considered as the "identifier" or the "locator" carried as part of the information unit).

The DRP function is decomposed into the following functions (see Fig.1):

- o **Computation** (box_4.1): function applied to (structured) routing information units and/or (structure) topology information units so as to produce (new) routing paths from which routing table entries can be derived. Computation can be seen as the operation of finding the routing path that minimizes/maximizes a (multi-)constrained (multi-)objective function. This function can be further sub-divided as follows:
  - **Compute Global-Full** function (box_4.1.1): computation is based on global knowledge of the graph; upon routing information update the full routing table is recomputed.
  - **Compute Global-Incremental** function (box_4.1.2): computation is based on global knowledge of the graph; upon routing information update only the affected routing table entries are recomputed.
  - **Compute Local** function (box_4.1.3): computation is based on local knowledge (neighbors related information) possibly complemented by a partial knowledge of remote/non-local parts of the network graph properties.
  - **Compute Sequential** function (not represented): computation is based on hop-by-hop/serial information propagated by local neighbors along either a given spatial trajectory.

- o **Selection** (box_4.2): this function performs either by enforcing selection rules, by applying filters, and/or by multi-criteria decision on a set of routing information units (typically routing paths with associated attributes). By means of this processing, a limited number of routing paths is selected from which routing table entries can be derived.
  - **Select/Filter Paths per Destination** function (box_4.2.1): routing paths are selected per destination, e.g., path-vector based routing protocols.
  - **Select/Filter (logical) Ports per Destination** function (box_4.2.2): (logical) ports are selected per destination, e.g., spanning-tree based routing protocols.

The sequential relationship between these functions using an EFFBD diagram is shown in Fig. 8.1 and 8.2.

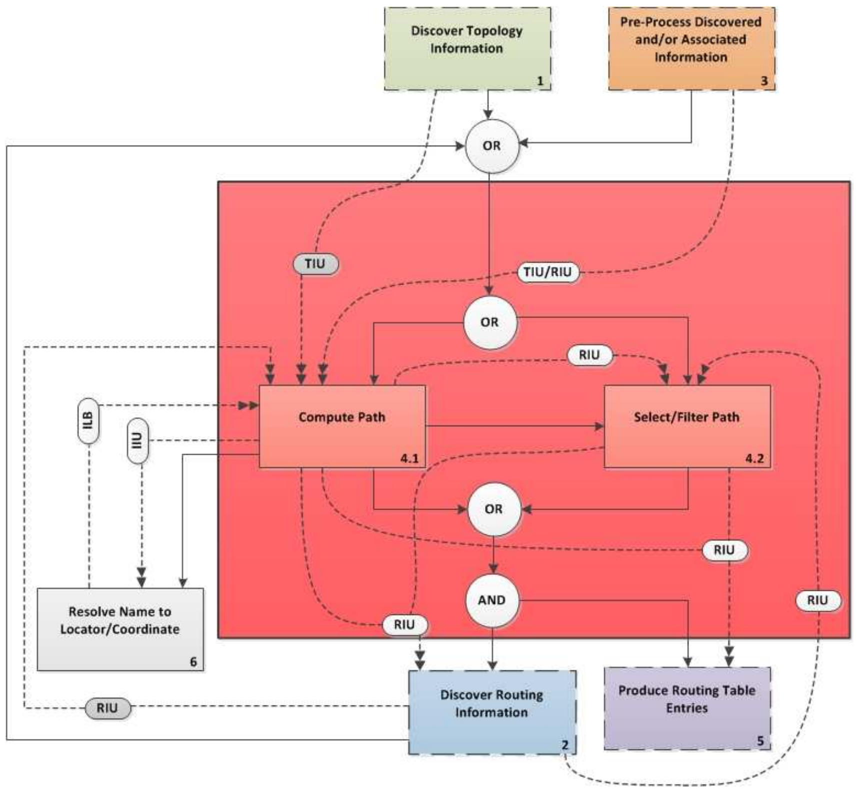

Discover Topology Information
1
Pre-Process Discovered and/or Associated Information
3
OR
TIU
TIU/RIU
OR
RIU
Compute Path
4.1
Select/Filter Path
4.2
ILB
IIU
OR
RIU
AND
RIU
RIU
Resolve Name to Locator/Coordinate
6
Discover Routing Information
2
Produce Routing Table Entries
5
RIU

Fig.8.1: EFFBD of the Determine Routing Path (DRP) function

The DRP function is activated/triggered by the following functions (Fig.5 and Fig.8.1):

- The DTI function in order to process topology information units and to subsequently obtain new (“computed”) routing paths.
- The DRI function in order to compute and/or select routing paths.
- The PDAI function in order to process “structured” topology and/or routing information units and to subsequently obtain new (“computed”) routing paths.

The DRP function activates/triggers the following functions (Fig.5 and Fig.8.1):

- The DRI function in order to store “computed” and “selected” routing paths in the RIB.
- The RNLC function in order to resolve a node identifier to its associated locator
- The PRTE in order to generate routing and forwarding tables entries based on stored “computed” and “selected” routing paths.

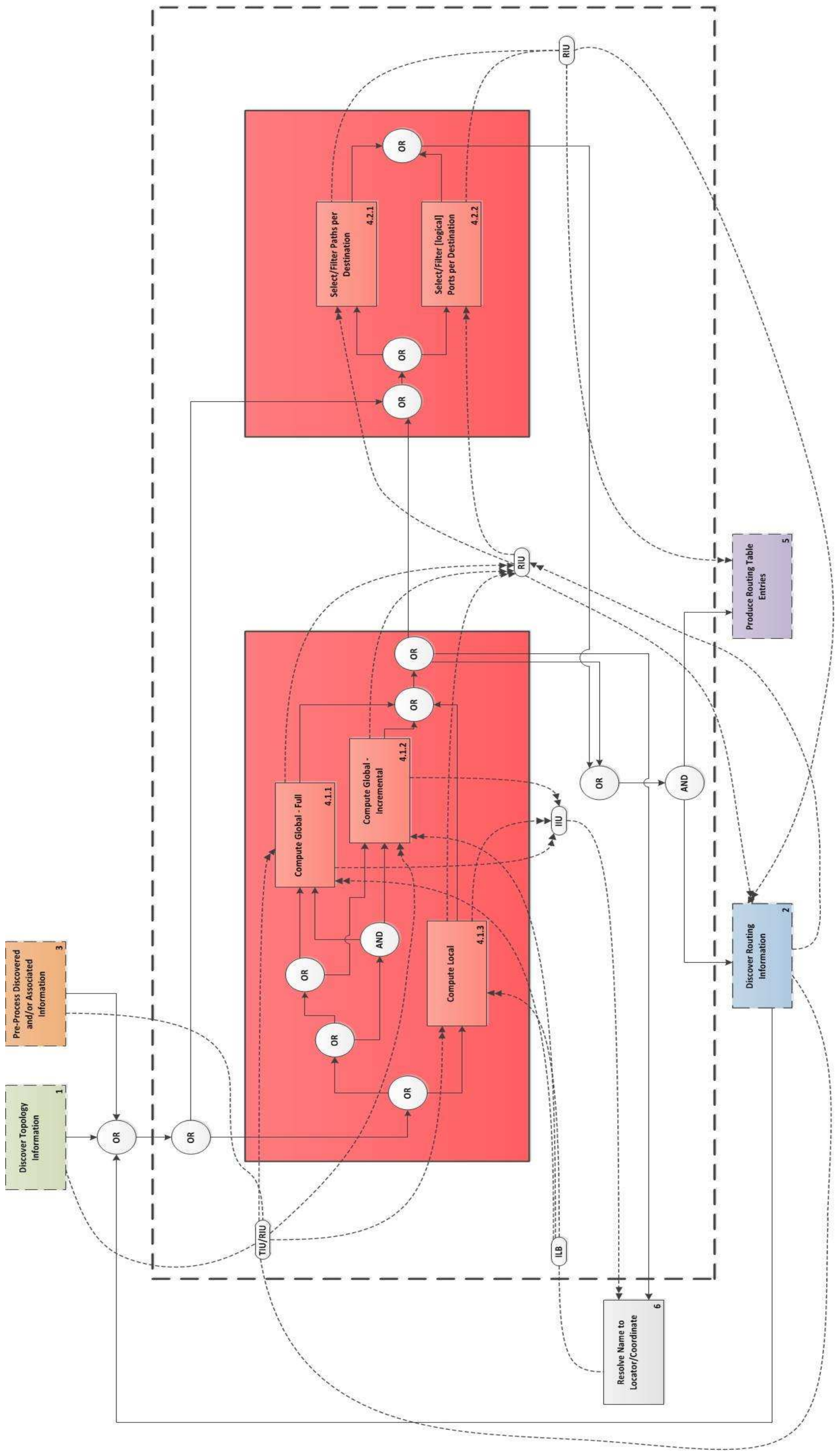

Discover Topology Information 1
Pre-Process Discovered and/or Associated Information 3
Compute Global - Full 4.1.1
Compute Global - Incremental 4.1.2
Compute Local 4.1.3
Select/Filter Paths per Destination 4.2.1
Select/Filter [logical] Ports per Destination 4.2.2
Produce Routing Table Entries 5
Discover Routing Information 2
Resolve Name to Locator/Coordinate 6
OR
AND
RIU
IIU
ILB
TIU/RIU

Fig.8.2: EFFBD of the Compute Routing Path and the Select/Filter Path Functions

3.2.6 Produce Routing Table Entry (PRTE) function

The "Produce Routing Table Entry (PRTE)" function (box_5) is in charge of the derivation of a route from the computed and/or selected routing paths and the generation of the Routing Table Entries (RTEs) from the (selected) route, together with the creation of an RTE or the update of an existing RTE in the Routing Table (RT). Each Forwarding Table Entry (FTE) is subsequently generated or derived from a sub-set of one or more RTEs.

The PRTE function can thus be decomposed into the following functions (Fig.1):

- **Derive Route** (DE) (box_5.1): Derivation of a route from the computed and/or selected routing path(s) and selection of the route (if multiple routes are produced). Note that selected routes are also transferred in the Routing Information Base (RIB) for subsequent exchange as part of the discovery process.
- **Generate the Routing Table Entry** (RTE) from the computed and/or selected route. Note that this function is often combined with the Derive route function.
- **Create Routing Table Entry** (RTE) (box_5.2): create routing table entries in the routing table (when no prior entry exists in the Routing Table)
- **Update Routing Table Entry** (RTE) (box_5.3): update of an existing routing table entry of the routing table (resulting from a change in routing path or its attributes)
- **Transfer to Forwarding Table** (box 5.4): derive a forwarding entry from a set of/one routing table entries and the store the resulting entry in the forwarding table.

The sequential relationship between these functions using an EFFBD diagram is shown in Fig. 9.

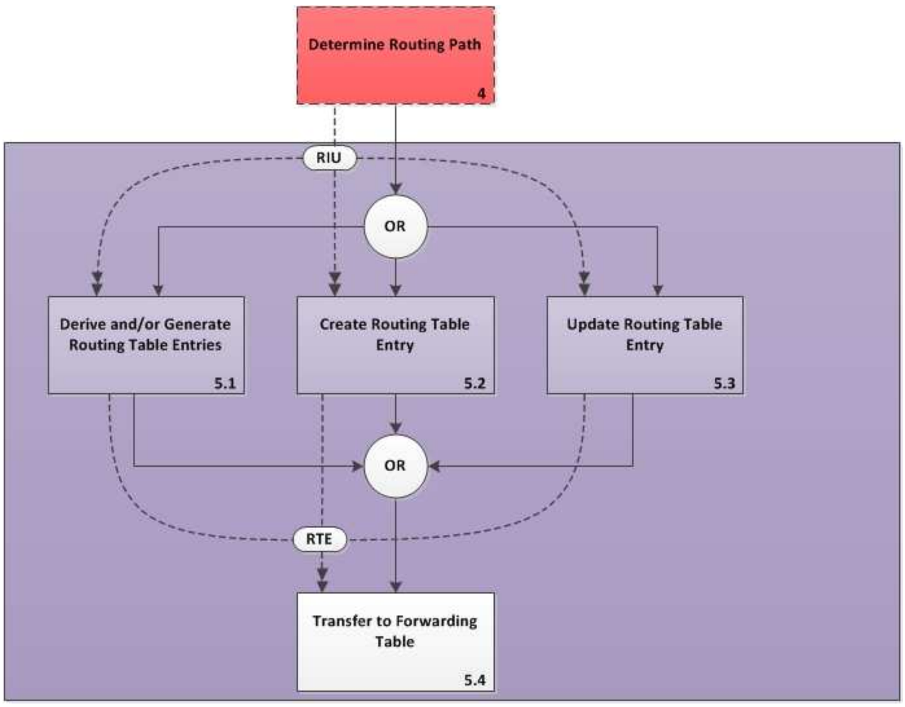

Determine Routing Path
4
RIU
OR
Derive and/or Generate Routing Table Entries
5.1
Create Routing Table Entry
5.2
Update Routing Table Entry
5.3
OR
RTE
Transfer to Forwarding Table
5.4

Fig.9: EFFBD of the Produce Routing Table Entry (PRTE) function

The PTRE is enabled/triggered by the DRP function (see Fig. 5 and 9): in order to generate the Routing and forwarding table entries based on the “computed” and “selected” routing paths..

3.2.7 Resolve Name to Locator / Coordinate (RNLC) function

It is important in the context of this document, and more generally in the context of the specification of a routing system architecture to make a clear distinction between identifier and locator. Identifiers are unambiguously assigned from a value space to nodes. Identifiers can either follow the topology, thus be topology-dependent (in this case, they are referred to as locators and can be used directly by the routing and the forwarding function without requiring resolution) or not, thus be topology-independent (in this case, they are referred to as names or simply identifiers). On the other hand, a locator identifies a location in an internetwork. Nodes and endpoints are assigned locators. A node is assigned only one locator. Locators identify nodes by specifying "where" the node is positioned in the network. Locators do not specify a path to the node. An endpoint can be assigned more than one locator so that a locator might appear in more than one location of an internetwork. A locator can take the form of a topology dependent label (flat and unstructured), a topologically assigned address (structured), or a coordinate (structure determined by the geometric space).

Stricto sensu, routing and forwarding functions act on locators, i.e., identifiers (or names) are not used in making routing or forwarding decisions. In this respect, "name-independent" compact routing (i.e., the schemes that work with –topology-independent– identifiers) can be seen as name-dependent compact routing (using locators) operating underneath an identifier-to-locator resolution function. This resolution function can operate "locally" (sometimes even by locally equating the identifier to its associated locator) and is often customized on per routing scheme basis, thus sometimes strongly coupled to the routing functionality itself. A couple of representative examples extracted from existing routing schemes will make the semantic and role distinction between identifier and locator clearer. Current IP networks do not differentiate between IP addresses used as identifiers (Provider Independent addresses) or locators (Provider Allocated addresses). Indeed, current routing or forwarding engines (which by definition acts on IP addresses) do not discriminate whether these addresses are associated to a PI space or a PA space. In the Host Identity Protocol (HIP) IP addresses function as pure locators, and applications use Host Identifiers to name peer hosts (instead of IP addresses). In geometric routing, coordinates function as pure locators, and applications use Host Identifiers to name peer hosts. In name-independent compact routing (assuming identifiers are IP addresses), being either intermediate nodes (edge-to-edge) or hosts (end-to-end) schemes, there is an identifier-to-locator resolution function distributed among nodes performing on top of name-dependent compact routing using locators.

The "**Resolve Name to Locator / Coordinate**" (RNLC) function (box_6) is in charge of the association of node locators to node identifiers (also referred to as names) and the resolution of node identifiers to its associated locators. The information about node identifiers is structured in Identifier Information Units (IIUs), the information about node locators is structured in Locator Information Units (LIUs), and the information about identifier-locator(s) bindings is structured in Identifier-Locator Bindings (ILBs). This function is mainly seen as "external" to the routing functional area of the nodes since the association of locators to identifiers and the maintenance of the identifier-locator(s) binding information is done by an external resolving entity. Internally, in the nodes, this function is responsible of issuing the resolution requests with this external resolving entity and/or with an internal database where the identifier-locator(s) binding information is stored ("cached"), processing the responses and storing the identifier-to-locator binding information. The ILB Information Base (IIB) is the database where ILBs are maintained.

The RNLC function (see the definition in section 1) is decomposed into the following functions (Fig.1):

- **Resolve** (box 6.1): translation, conversion or mapping from a node identifier to its associated locator(s) from the corresponding identifier-locator(s) binding information. Resolution relies on a resolving entity that is in charge of the association of node locator(s) to a node identifier which results in identifier-locator(s) binding information. The tasks performed by this entity are the following 1) determine the location of a node (given the node identifier), bind the node identifier to a set of one or more locators, and stores this information in its IIB; 2) upon receiving a resolution request from a node, send back the corresponding identifier-locator(s) binding information to the requesting node (then the receiving node stores this binding information in its IIB). In case the resolving entity does not have the requested identifier-locator(s) binding information, it shall further determine which other resolving entity can store this information (or query an intermediate entity that would perform this action on behalf of it). Internally, in the nodes, this function is responsible of issuing the resolution requests with the resolving entity and/or with the local ILB where the identifier-locator(s) binding information is stored ("cached"), processing the responses and storing the identifier-to-locator binding information. The resolving function is an associated (local) routing function, i.e., stricto sensu this function is not part of the routing functional area. This function may be co-located with the other routing functions or not. When collocated with the routing function, an internal interface enables exchanges with the resolution function (also referred to as local resolver); when non-collocated an external interface enables these exchanges. It is important to notice that even when "external" resolution is performed the request/responses are passed via the local resolver also local resolution agent (if that function isn't actually present locally a "void" function takes place). Send and Receive resolution requests and responses to/from a resolving entity through dedicated interfaces (to a local resolving agent or directly to a non-collocated resolving entity). This includes the operations related to packet scheduling and management of input and output queues (storage, sending priorities, discarding rules, etc.), and the classification of the received ILBs.

- o **Operate ILB Information Base** (box 6.2): creation, maintenance and use of the ILB database. This includes the control to access the data, the enforcement of data integrity, the management of concurrency, the recovery and restoration of the database after failures, and the maintenance of the database security.

The sequential relationship between these functions using an EFFBD diagram is shown in Fig. 10.

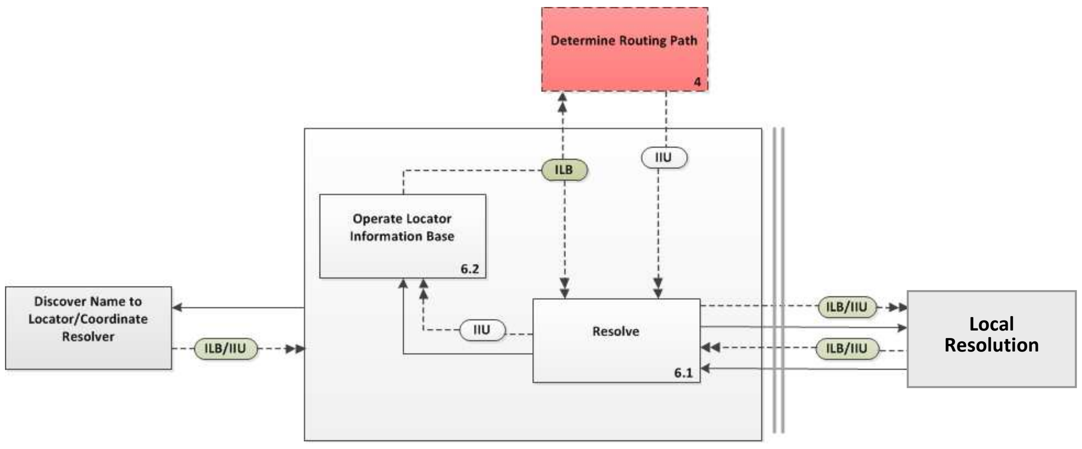


Fig.10: Resolve Name to Locator / Coordinate" (RNLC) function

The RNLC function is enabled/triggered by the following functions (Fig. 2 and 8):

- o By the DRP and PDAI function in order to resolve a node identifier to its associated locator
- o By the external and/or internal resolving entity, when resolution information is received from this entity through the external and/or internal interfaces.

The RNLC function enables/triggers the external and/or internal resolving entity, when resolution requests are sent to it through the external and/or internal interfaces.

3.3 Control Flow (and Control Interfaces)

This part of the functional model specification includes: i) activation/termination of (sub-)functions so as to specify the temporal relationship between them, their duration, and their sequencing/ordering over time and under which internal/external conditions (conditional control), ii) control loops: Feedback loops, adaptive loops and model reference loops, iii) and error control. Note that in order to include iteration (IT symbol), loop (LP symbol), and replication (RP symbol), the extended version of FFDB shall be used because base FFDB only allows to depict 4 types of control structures being series, concurrency, selection and multi-exit.

3.3.1 Error Control

The following table contains a list of the error situations that may occur in the routing system architecture, in terms of the functions that are involved in the error, a description of the error, and the mechanisms that could be implemented in the functions to solve the error.

Table 2: Functional Error Description and Potential mitigation

FUNCTION: SUBFUNCTION	ERROR DESCRIPTION	POTENTIAL MITIGATION
DTI, DRI, RNLC: Communicate	Corruption of information units exchanged with other nodes that occur during their transmission through the network	Error detection codes and retransmission mechanisms
	Loss of information units exchanged with other nodes that occur during their transmission through the network	Sequence numbers or similar identifiers, timeout and retransmission mechanisms
	Duplication of information units exchanged with other nodes that occur during their transmission through the network	Sequence numbers or similar identifiers, and discarding
	Malformation of messages	Check the message structure
	Security attacks, against the confidentiality (interception of the information)	Encryption mechanisms
	Security attacks, against the identity (overriding, superseding, repudiation)	Checksums and encryption mechanisms
	Security attacks, against the information integrity (alteration)	Checksums
DTI: Exchange Topology Information, DRI: Exchange Routing Information, RNLC: Resolve	Inconsistency of information units exchanged with other nodes (unexpected or incoherent information) caused by software or hardware errors	Mechanisms to detect the inconsistent behavior of other nodes and that react according to it
	Bad (too high) update messages rate	Setting and checking of negotiated and/or configured rate parameters
DTI: Operate Topology Information Base, DRI: Operate Routing Information Base, RNLC: Operation ILB Information Base	Data base unauthorized access	Access control mechanisms
	Simultaneous writing operations in data bases over related information and data base entry corruption	Concurrency control and verification of data base consistence
	Data base failure	Data base recovery mechanisms
	Aged / out-of-date data	Setting and checking TTL information for the data. Remove out-of-date data.
All: All	Mis-interpretation of internal timers and/or input information units that initiates a sequence of functions	Detection of bad sequences of functions and/or loops (through counters and others)

4. Applications of Generic Functional Model

This section provides several examples of application of the generic functional model detailed in Section 3. For this purpose, we refer to the following routing schemes: BGP path-vector routing protocol, greedy routing (on greedy embedding of the observed Internet topology and on a constructed Internet topology), and compact routing (both unicast and multicast). These examples are provided to demonstrate applicability of the proposed generic functional model (not to delimit a subset of routing schemes and algorithms that will be developed in the context of the EULER project).

4.1 Path-vector routing

Specified in RFC 4271, Border Gateway Protocol (BGP) is a routing protocol relying on the path-vector routing algorithm. This routing protocol is used to exchange network reachability information between autonomous systems (AS); hence BGP relies on an AS path vector routing algorithm. An AS is defined as "a set of routers under the control of a single technical administration entity or unit that presents a consistent picture of what destinations are reachable through it." Each AS is identified by its AS Number (ASN). Each AS has one or more border routers that connect to routers in neighboring AS, and possibly a number of internal BGP routers. The main function of BGP is to exchange network reachability information with peering (neighboring) BGP routers. Reachability information includes an AS path that lists the sequence of AS numbers traversed by the BGP route advertisement comprising reachability information from the originating AS. This information is used by BGP routers for constructing AS connectivity graph for this reachability, and so as to detect and avoid routing loops. BGP connections between routers belonging to neighboring AS are called eBGP (external BGP), while those between routers in the same AS are called iBGP (internal BGP). Note that adjacent AS may have more than one eBGP connection.

4.1.1 BGP Operations

In BGP, a route is defined as a unit of information that pairs a set of destinations with the attributes of a path to those destinations. These routes are advertised between BGP routers in UPDATE messages. The set of destinations are systems whose IP addresses are contained in one IP address prefix that is carried in the Network Layer Reachability Information (NLRI) field of an UPDATE message. The actual path to this set of destinations is the information reported in the AS_PATH attribute field of the same UPDATE message. The AS_PATH attribute enumerates the sequence of AS numbers a route in the UPDATE message has traversed.

The AS topology of the routing system is described as a graph G = (V,E), where the vertices (nodes) set V, |V| = n, represents the AS, and the edges set E, |E| = m, represents the links between AS. At each node $u \in V$, a route r per destination d ($d \in D$) is selected and stored as an entry in the local routing table (RT) whose total number of entries is denoted by N, i.e., |RT| = N. At node u, a route r_i to destination d is defined by $r_i = \{d, (a_k=u, a_{k-1},\ldots,a_0=v), A\}$ with $k > 0 \mid \forall j, k \geq j > 0, \{a_j, a_{j-1}\} \in E$ and $i \in [1,N]$, where $(a_k=u, a_{k-1},\ldots,a_0=v)$ represents the AS_PATH attribute of the route r_i, a_{k-1} the next hop of v along this path from node u to v, and A its attribute set. As part of the set of mandatory attributes, we can mention the ORIGIN (generated by the BGP speaker that originates the associated routing information), the NEXT_HOP (defines the IP address of the router that should be used as the next hop to the destinations listed the UPDATE message), and the LOCAL_PREF (used by an iBGP speaker to inform its peers within the same AS of the advertising speaker's degree of preference for an advertised route). The MULTI_EXIT_DISC (MED) is an optional non-transitive attribute whose value may be used by a BGP speaker on external (inter-AS) links to discriminate among multiple exit or entry points to the same neighboring AS. In addition to the AS_PATH, the LOCAL_PREF and MED attribute are also used in the BGP Route Selection process (see below).

BGP routers (or speakers) advertise network reachability information about destinations by sending to their neighbors UPDATE messages containing set of destination address prefix announcements (feasible routes) or withdrawals (unfeasible routes) together with attributes associated to a path to these destinations.

- An announcement informs neighboring BGP routers of a path to a given destination. When a local BGP router propagates a route learned from the UPDATE message sent by one of its peering BGP routers, it modifies the route's AS_PATH attribute based on the location of the BGP router to which the UPDATE message containing that route will be sent. Formally, let $P_{(u,v),d}$ denote the set of paths from node u to v towards destination d where each path p(u,v) is of the form $\{(a_k=u, a_{k-1},\ldots,a_0=v), A\}$. A routing UPDATE message leads to a change of the AS_PATH attribute $(a_k, a_{k-1},\ldots,a_0)$ or an element of its attribute set A.
- A withdrawal is an update indicating that a previously advertised destination is no longer reachable. Route withdrawals only contain the destination and implicitly tell the receiver to invalidate (or remove) the route previously announced by the sender. Formally, let $P_{(u,v),d}$ denote the set of paths from node u to v towards destination d where each path p(u,v) is of the form $\{(a_k=u, a_{k-1},\ldots,a_0=v), A\}$. A withdrawal is denoted by an empty AS_PATH attribute (ε) and $A = \emptyset$: $\{d,\varepsilon,\emptyset\}$. According to the above definition, if there is more than one path per destination d, each path will be associated to a distinct route.

When a local BGP router propagates a route learned from the UPDATE message sent by one of its peering BGP routers, it modifies the route's AS_PATH attribute based on the location of the BGP router to which the UPDATE message containing that route will be sent. In contrast, route withdrawals only contain the destination and implicitly tell the receiver to invalidate (or remove) the route previously announced by the sender.

A BGP router receives UPDATE messages from its BGP peering neighbors following a time varying interval bound by a minimum threshold. As detailed in RFC 4271, there is a minimum amount of time (MRAI) between two BGP UPDATE messages sent towards the same BGP router. Thus, a given BGP router receives one BGP UPDATE message per MRAI time interval per neighbor (and sometimes per destination prefixes d). The output (i.e., the class of the BGP UPDATE message) of the learning process is used by the selection process of the local BGP router. This output is not distributed to the router's neighbors or other nodes in the system. However, the router's output (i.e., the BGP update messages that are forwarded) will influence the route selection of the BGP router's neighbor; as depicted in Fig.7.3, the crossed circles represents selectors acting at the input and output of the "BGP route selection" process).

4.1.2 BGP Routing Information Bases (RIB)

BGP routes are stored in the Routing Information Bases (RIBs). At each BGP speaker, the RIB consists of three distinct parts:

- Adj-RIB-In: stores routing information (routes) learned from inbound UPDATE messages received from other BGP speakers. These routes are available as input to the Decision Process after applying Import Policy rules (import filter).
- Loc-RIB: contains the local routing information the BGP speaker selects by applying its local policies to the routing information (routes) contained in its Adj-RIB-In. These are the routes that will be used by the local BGP speaker.
- Adj-RIB-Out: stores routing information (routes) that the local BGP speaker has selected for advertisement to its peers. This routing information will be carried in the local BGP speaker's UPDATE messages and advertised to its peers by means of the local speaker's UPDATE messages after applying Export Policy rules (export filter).

In other terms, the Adj-RIB-In contains unprocessed routing information that has been advertised to the local BGP speaker by its peers; the Loc-RIB contains the routes that have been selected by the local BGP speaker's Decision Process; and the Adj-RIB-Out organizes the routes for advertisement to specific peers (by means of the local speaker's UPDATE messages).

4.1.3 BGP Route Selection Process

When a router receives a route advertisement, it first applies inbound filtering process (using some import policies) to the received routing information. If accepted, the route is stored in the Adj-RIB-In. The collection of routes received from all neighbors (external and internal) and stored in the Adj-RIB-

In defines the set of candidate routes (for that destination). Subsequently, the BGP router invokes a route selection process - guided by locally defined policies - to select from this set a single best route for each destination. After this selection is performed, the selected best route is stored in the Loc-RIB and is subjected to some outbound filtering process and then announced to all the router's neighbors. Importantly, prior to being announced to an external neighbor, but not to an internal neighbor in the same AS, the AS path carried in the announcement is prepended with the ASN of the local AS.

4.1.4 Hierarchical Decomposition of BGP

The BGP routing functionality can be decomposed into three main functional blocks: i) the discovery (push) of routing information, i.e., BGP routes advertized by BGP peers, together with the optional inbound filtering of the received BGP routes, ii) the per-node selection of the "best" route by means of the so-called BGP route selection process, and iii) the push of the selected route together with optional outbound filtering of the selected BGP routes to downstream neighbors/BGP speakers.

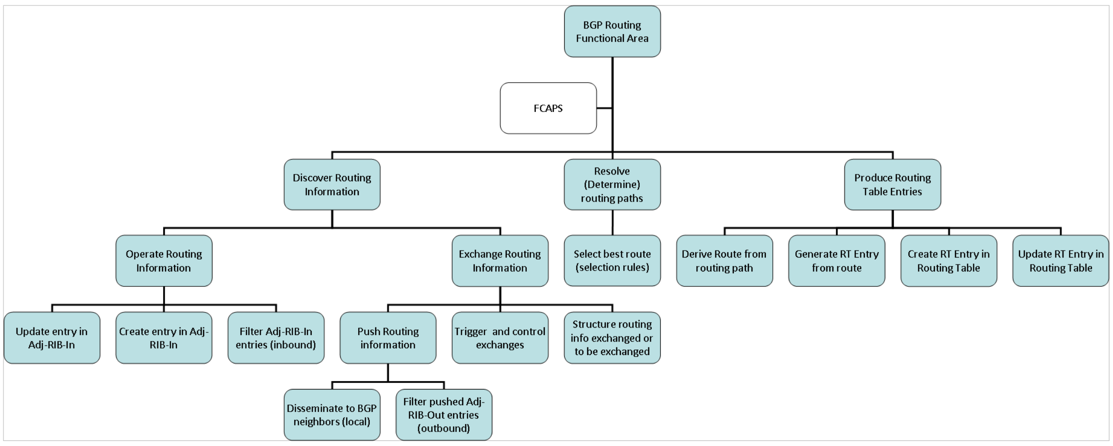

BGP Routing Functional Area
FCAPS
Discover Routing Information
Resolve (Determine) routing paths
Produce Routing Table Entries
Operate Routing Information
Exchange Routing Information
Select best route (selection rules)
Derive Route from routing path
Generate RT Entry from route
Create RT Entry in Routing Table
Update RT Entry in Routing Table
Update entry in Adj-RIB-In
Create entry in Adj-RIB-In
Filter Adj-RIB-In entries (inbound)
Push Routing information
Trigger and control exchanges
Structure routing info exchanged or to be exchanged
Disseminate to BGP neighbors (local)
Filter pushed Adj-RIB-Out entries (outbound)

Fig.11: Hierarchical decomposition of BGP routing function

4.1.5 BGP Flow Block Diagrams

The BGP routing protocol makes use of the following functional blocks discussed in Section 1.3.1: the Discover Routing Information (DRI) function, the Determine Routing Path (DRP) function, and the Produce Routing Table Entry (PRTE) function.

4.1.5.1 Discover Routing Information

The **Discover Routing Information (DRI)** function enables the discovery of routing information. This information comprises the set of destination IP address prefix that is carried in the Network Layer Reachability Information (NLRI) field of an UPDATE message, the actual path to this set of destinations in the AS_PATH attribute field of the same UPDATE message together with mandatory and optional attributes associated to this path. Optionally, the incoming routing information is filtered by means of inbound or import filters before being stored in the Adj-RIB-In.

4.1.5.2 Determine Routing Path

The **Determine Routing Path (DRP)** function performs by means of route **selection** function which applies either by enforcing selection rules on a set of (inbound filtered) routes stored in the Adj-RIB-In. By means of this selection process, a single route per destination is selected, and stored in the Loc-RIB.

4.1.5.3 Produce Routing Table Entry

The **Produce Routing Table Entry (PRTE)** function then derives a route from the selected route stored in the Loc-RIB and generates a Routing Table Entry (RTEs) from this route, together with the creation of an RTE or the update of an existing RTE in the Routing Table (RT).

4.2 Compact Unicast Routing

Compact unicast routing aims to find the best tradeoff between the memory-space required to store the routing table (RT) entries at each node and the stretch factor increase on the routing paths it produces. Such routing schemes have been extensively studied following the model developed in [Peleg89]. Since then, in accordance to the distinction between labeled (nodes are named by polylogarithmic size labels encoding topological information) and name-independent (node names are named by topologically independent) compact routing schemes have been designed, notably in the AGMNT name-independent compact routing scheme [Abraham08]. In the context of this document (and more generally in the context of this project), It is important to note that the AGMNT name-independent compact routing scheme can not be considered in the framework of the current project because this compact routing scheme is centralized and static (instead of distributed and dynamic). Such scheme is thus by definition inapplicable for the Internet. The present section corresponds to an attempt of an adaptation of the AGMNT compact routing scheme into a distributed and dynamic setting. The question remains open if starting from another or even a new foundational scheme would provide a more suited answer to the fundamental tradeoff between routing path stretch - memory space consumption - computation complexity and adaptation/communication cost.

The key idea of Compact Routing Scheme is to consider shortest paths routing tables for nodes in the close neighborhood and a path built with two shortest paths for nodes at a larger scale of distance. In the last case, the intermediate node is either called a landmark or corresponds to a node whose color is equal to the hash value of the destination (there exists such a node in the close neighborhood for every node). Each node has a color chosen among the set {1..k} and is able to use a balanced hash function h that assign a color for each node identifier. Note that the color of a node does not correspond in general to the color defined by the hash function. In order to built the tables dedicated to routing, each node of color X has to know its close neighborhood, the set of landmarks and nodes for which the hash value is X. The knowledge of shortest paths between these 3 categories of nodes is enough to get a routing with a stretch less than 3.

4.2.1 Notation

Throughout this section the following notation and terms are used:

- o Colors: we suppose as input a colored graph with k colors. Each node u has i) a color, in the set [1,k], denoted by c(u) and ii) a hash value, in the set [1,k], h(u) calculated from its identifier. Note that there is no correlation between c(u) and h(u).

- o Landmarks: we consider a random set of nodes called landmarks L, which has size O(n/k) . One easy way to choose landmarks is to consider the set $L = \{u \in V \mid c(u) = 1\}$. Each node u has a closest landmark $L(u) \in L$, the set of nodes that have chosen a node l as landmark is denoted $L^{-1}(l) = \{ u \in V \mid L(u) = l \}$.

- o Manager: we say that a node u such that $R(u) = \{v \in V \mid h(v) = c(u)\}$ ''manage'' nodes of R(u) .
- o Vicinity Ball: Is the set of nodes B(u) close from u such that every k color is represented in B(u) at least once. Note that a single node could belongs to several Vicinity Balls $B^{-1}(u) = \{v \in V \mid u \in B(v)\}$.

4.2.2 Hierarchical Decomposition

The routing functional block can be decomposed in the following way. Each node maintains two tables (Topology Information Base (TIB) and Routing Information Base (RIB) of information concerning its environment. This two tables are maintained by discovery functions (discover routing information and discover topology information). Moreover, this tables are used to take routing decisions/selections (resolve, determine routing paths).
The last functional block, pre-process, consists in functions that perform computations/associations of paths (i.e., converts paths to routes). This last function is used after discovering routing information to

fill routing tables. The next figures represent the hierarchical decomposition of the functional blocks composing a compact routing functional area. Each functional block is described more precisely in the following paragraphs and diagrams.

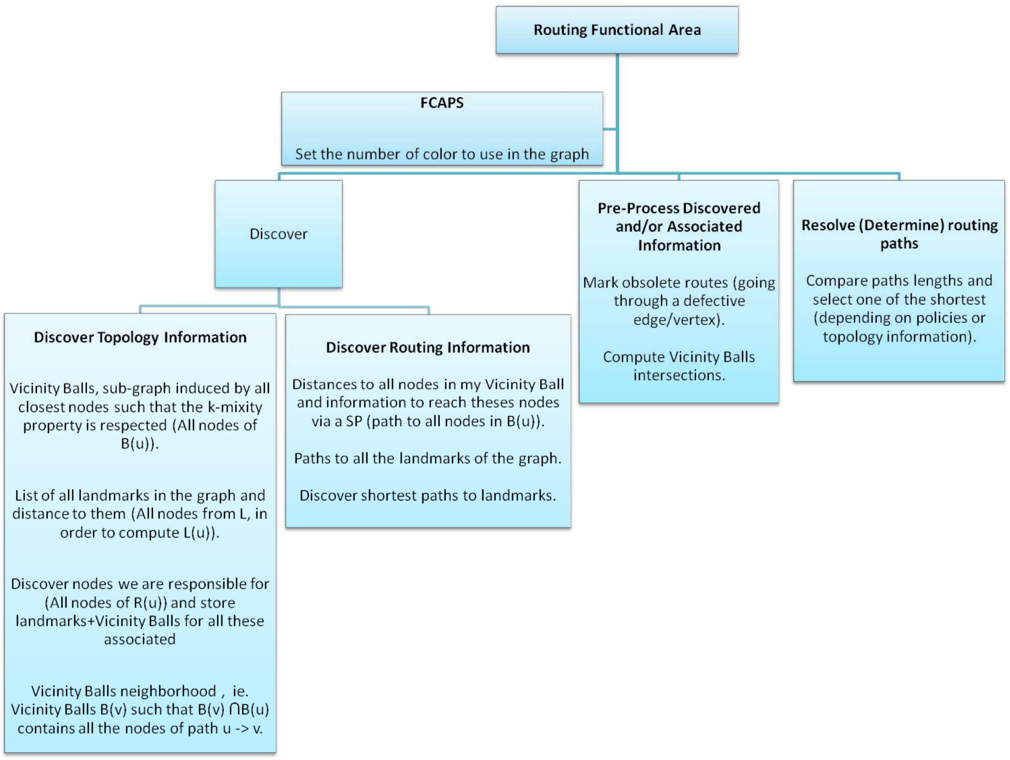

Routing Functional Area
FCAPS
Set the number of color to use in the graph
Discover
Pre-Process Discovered and/or Associated Information
Mark obsolete routes (going through a defective edge/vertex).
Compute Vicinity Balls intersections.
Resolve (Determine) routing paths
Compare paths lengths and select one of the shortest (depending on policies or topology information).
Discover Topology Information
Vicinity Balls, sub-graph induced by all closest nodes such that the k-mixity property is respected (All nodes of B(u)).
List of all landmarks in the graph and distance to them (All nodes from L, in order to compute L(u)).
Discover nodes we are responsible for (All nodes of R(u)) and store landmarks+Vicinity Balls for all these associated
Vicinity Balls neighborhood , ie. Vicinity Balls B(v) such that B(v) ∩B(u) contains all the nodes of path u -> v.
Discover Routing Information
Distances to all nodes in my Vicinity Ball and information to reach theses nodes via a SP (path to all nodes in B(u)).
Paths to all the landmarks of the graph.
Discover shortest paths to landmarks.

Fig.13: Hierarchical decomposition of compact routing

i) Discover Topology Information (DTI) functional decomposition (see Fig.13.1):

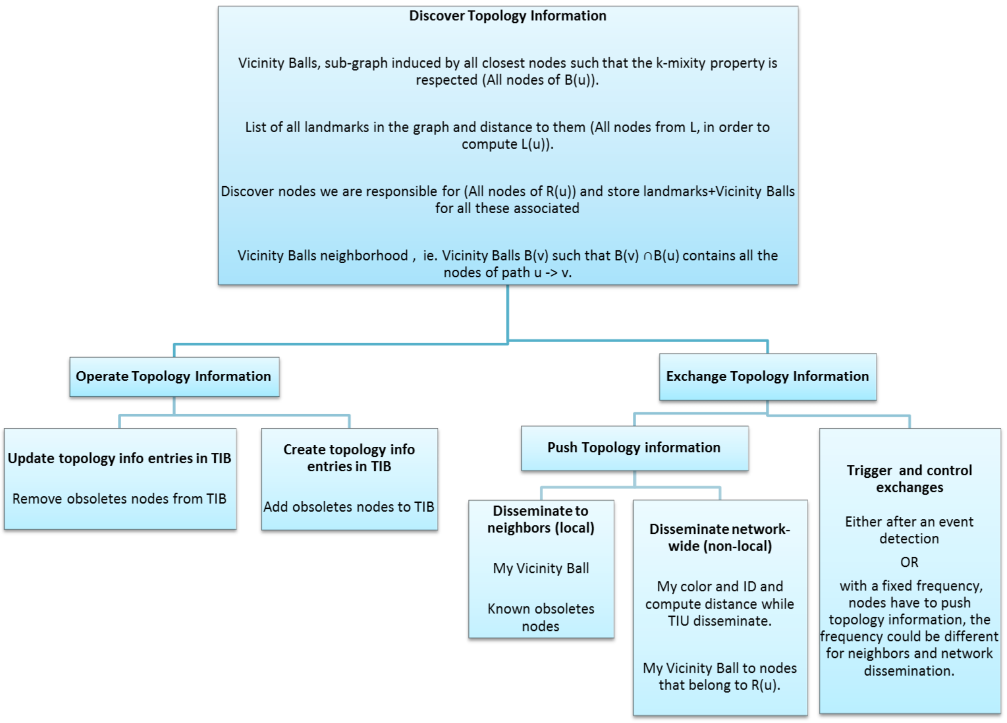

Discover Topology Information
Vicinity Balls, sub-graph induced by all closest nodes such that the k-mixity property is respected (All nodes of B(u)).
List of all landmarks in the graph and distance to them (All nodes from L, in order to compute L(u)).
Discover nodes we are responsible for (All nodes of R(u)) and store landmarks+Vicinity Balls for all these associated
Vicinity Balls neighborhood , ie. Vicinity Balls B(v) such that B(v) ∩B(u) contains all the nodes of path u -> v.
Operate Topology Information
Exchange Topology Information
Update topology info entries in TIB
Remove obsoletes nodes from TIB
Create topology info entries in TIB
Add obsoletes nodes to TIB
Push Topology information
Disseminate to neighbors (local)
My Vicinity Ball
Known obsoletes nodes
Disseminate network-wide (non-local)
My color and ID and compute distance while TIU disseminate.
My Vicinity Ball to nodes that belong to R(u).
Trigger and control exchanges
Either after an event detection
OR
with a fixed frequency, nodes have to push topology information, the frequency could be different for neighbors and network dissemination.

Fig.13.1: Discover Topology Information (DTI) functional decomposition

ii) Discover Routing Information (DRI) functional decomposition (see Fig.13.2):

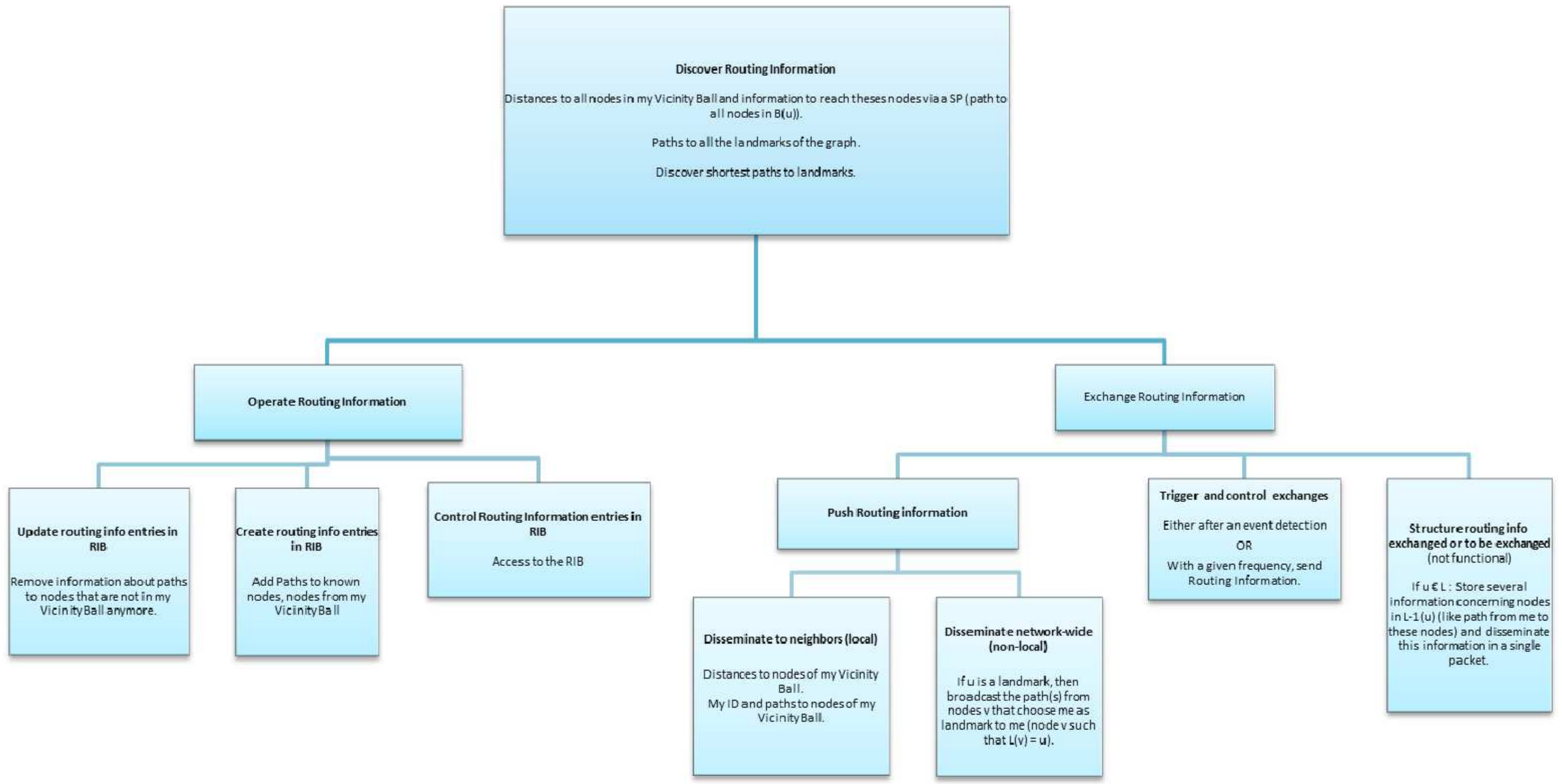

Discover Routing Information
Distances to all nodes in my Vicinity Ball and information to reach theses nodes via a SP (path to all nodes in B(u)).
Paths to all the landmarks of the graph.
Discover shortest paths to landmarks.
Operate Routing Information
Exchange Routing Information
Update routing info entries in RIB
Remove information about paths to nodes that are not in my Vicinity Ball anymore.
Create routing info entries in RIB
Add Paths to known nodes, nodes from my Vicinity Ball
Control Routing Information entries in RIB
Access to the RIB
Push Routing information
Disseminate to neighbors (local)
Distances to nodes of my Vicinity Ball.
My ID and paths to nodes of my Vicinity Ball.
Disseminate network-wide (non-local)
If u is a landmark, then broadcast the path(s) from nodes v that choose me as landmark to me (node v such that L(v) = u).
Trigger and control exchanges
Either after an event detection
OR
With a given frequency, send Routing Information.
Structure routing info exchanged or to be exchanged
(not functional)
If u € L : Store several information concerning nodes in L-1(u) (like path from me to these nodes) and disseminate this information in a single packet.

Fig.13.2: Discover Routing Information (DRI) function decomposition

iii) Pre-processing (PDAI) functional decomposition (see Fig.13.3):

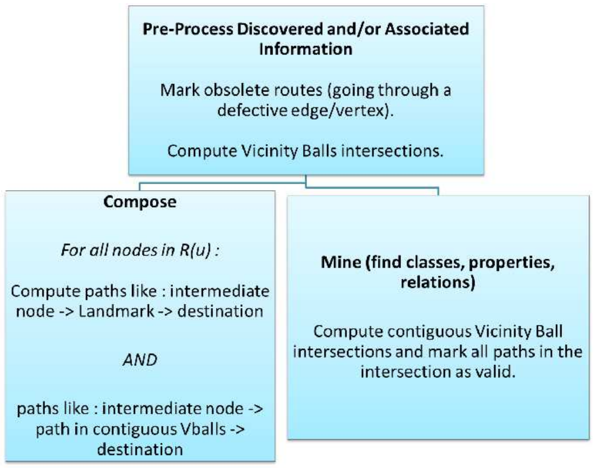

Pre-Process Discovered and/or Associated Information
Mark obsolete routes (going through a defective edge/vertex).
Compute Vicinity Balls intersections.
Compose
For all nodes in R(u) :
Compute paths like : intermediate node -> Landmark -> destination
AND
paths like : intermediate node -> path in contiguous Vballs -> destination
Mine (find classes, properties, relations)
Compute contiguous Vicinity Ball intersections and mark all paths in the intersection as valid.

Fig.13.3: Pre-processing (PDAI) functional decomposition

iv) Determine Routing Paths (DRP) functional decomposition (see Fig.13.4): every node u needs to fill its topology information base (TIB) with necessary information for these three kind of nodes i) nodes in its vicinity ball, ii) landmark nodes, and iii) nodes it manages.

In the following, we assume that the routing information base (RIB) contains lists of routing paths towards all the nodes recorded in the topology information stored in the TIB. The information needed is, for each of these nodes, at least one path, the distance to node u (length of the shortest known path), the node's color and the availability of the node. From this information, nodes can i) detect vicinity balls intersections (via some exchange of routing information), and ii) compute valid paths to known nodes and mark, as obsolete, paths that contain unavailable node. Then it is possible to order all the available (valid) paths by some preferential criteria like the shortest (example: select from paths that use landmarks or vicinity balls intersections), the path that uses the larger number of nodes of high degree, etc.

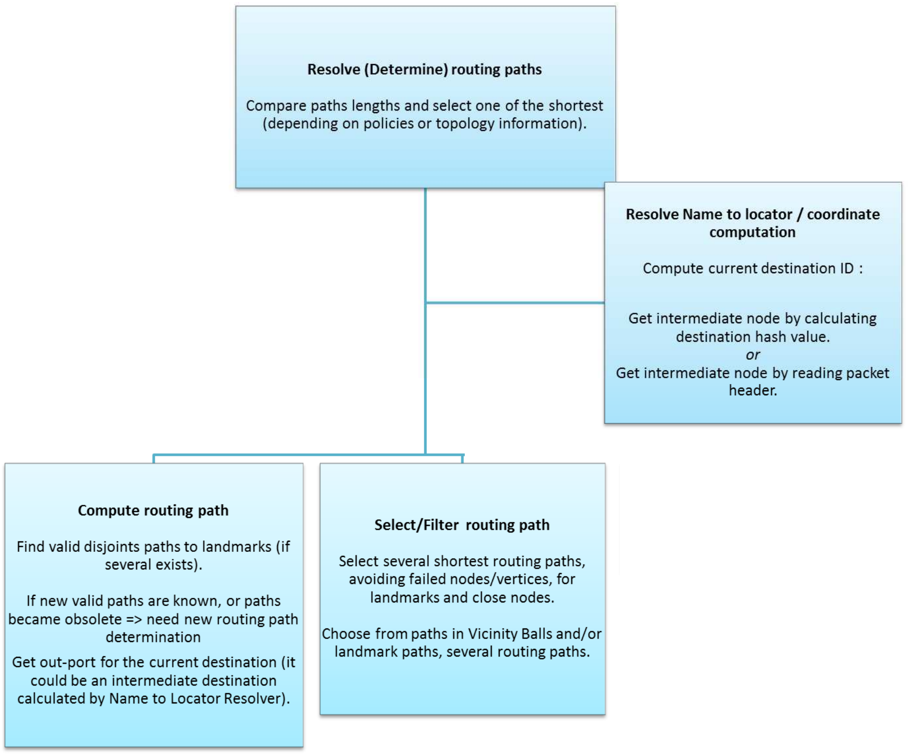

Resolve (Determine) routing paths
Compare paths lengths and select one of the shortest (depending on policies or topology information).
Resolve Name to locator / coordinate computation
Compute current destination ID :
Get intermediate node by calculating destination hash value.
or
Get intermediate node by reading packet header.
Compute routing path
Find valid disjoints paths to landmarks (if several exists).
If new valid paths are known, or paths became obsolete => need new routing path determination
Get out-port for the current destination (it could be an intermediate destination calculated by Name to Locator Resolver).
Select/Filter routing path
Select several shortest routing paths, avoiding failed nodes/vertices, for landmarks and close nodes.
Choose from paths in Vicinity Balls and/or landmark paths, several routing paths.

Fig.13.3: Determine Routing Paths (DRP) functional decomposition

v) Produce Routing Table Entry (PRTE) functional decomposition: all the routing paths considered as valid will be stored as entry in the routing table with respect to the order of selection. Then all favorite routing paths will be used to derive entry that will be stored in the forwarding table.

vi) Resolve Name to Locator / Coordinate (RNLC) functional decomposition: for each node v that does not fit in one of these three categories (vicinity ball, landmark nodes, nodes managed by node v), we need to use the name resolution in order to i) get a node's name that manages the node v (by using its hash value h(v)), and ii) get the next intermediate node in the path from a landmark to v (this could be sent by the source node in the header of the routing packet).

4.2.3 Relation between Functional Blocks

The next diagrams show the main relations between the functional blocks. We decided to focus on relations that are specific to AGMNT. In the original routing scheme, the description is static and centralized. Our goal is to extend the construction in a distributed and dynamic setting. We start with the following assumption: with a given frequency, neighbor nodes exchange their knowledge (TIB/RIB).

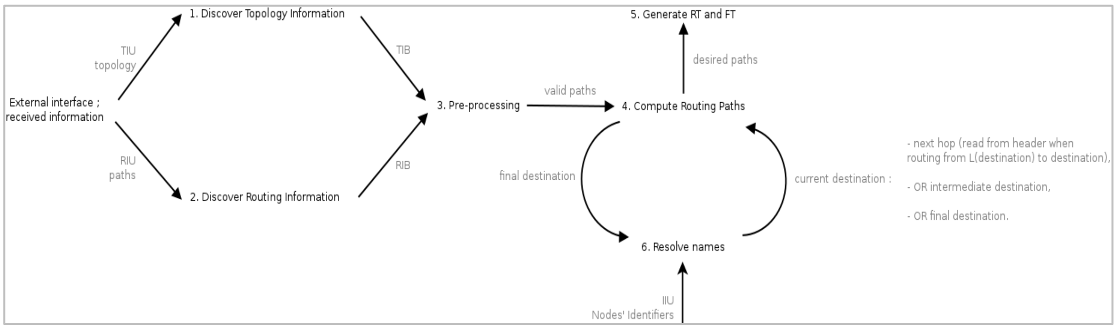


Fig.14: Main relations between the functional blocks

Another event can also change the knowledge of the current node: receiving a demand of routing, the node learns from the header of the routed message.

i) Discover Topology Information

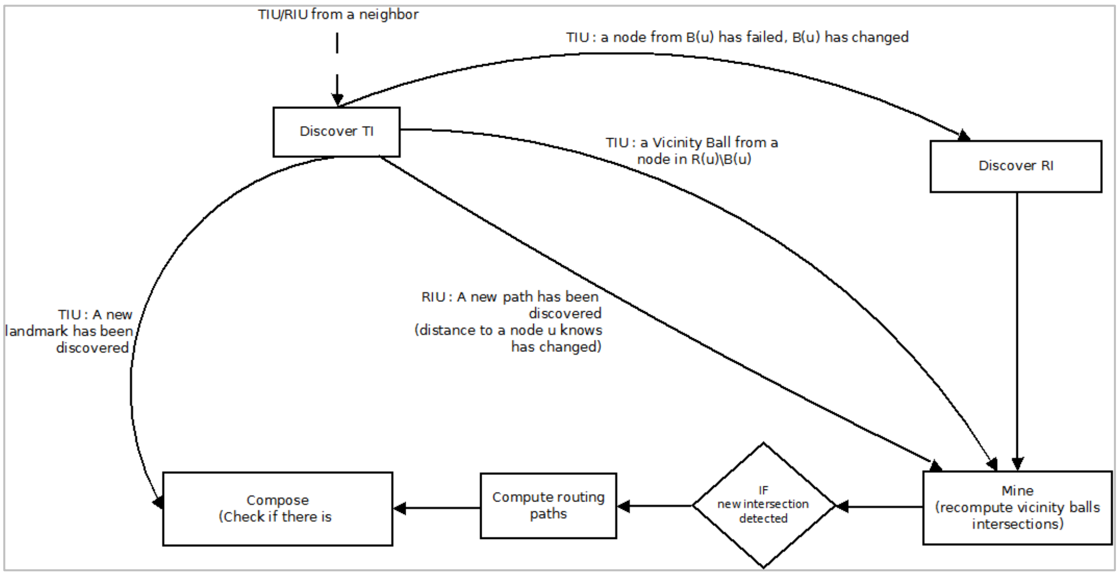

TIU/RIU from a neighbor
TIU : a node from B(u) has failed, B(u) has changed
Discover TI
TIU : a Vicinity Ball from a node in R(u)\B(u)
Discover RI
RIU : A new path has been discovered (distance to a node u knows has changed)
TIU : A new landmark has been discovered
Compose (Check if there is
Compute routing paths
IF new intersection detected
Mine (recompute vicinity balls intersections)

Fig.14.1: Relations of Discover Topology Information

A node u receives topology information from its neighbors:

- After a node v has been detected as failed, a notification is sent to its neighbors; when node u receives notification that node v has failed:
  - If node v $\in$ B(u), mark this node, and send a notification to Discover Routing Information in order to detect failed paths. Find a new node of color c(v) to add to my Vicinity Ball (u need to wait for its neighbors to send their Vicinity Balls to be able to determine which v to choose).
  - If node v $\in$ L, then all nodes from (R(u) $\cap$ L^{-1}(v)) may not be reachable (if B(u) and B(v) don't intersect) and node u needs to wait for every node w from this set to send their new landmark (L(w)).
  - If node v $\in$ R(u), mark this node as unreachable (it changes nothing for u)

- Notification from a failed node v (meaning that this node is not down any more): unmark the node and send a notification to Discover Routing Information in order to restore failed paths.

- Neighbor Vicinity Ball: Check my Vicinity to determine if there is any node that has to be added or removed? If nodes have to be removed, send notifications to Discover Routing Information in order to remove paths to these nodes from Routing Information Base.

- Notification of existence from a node v
  - such that h(v) = c(u) : store topology information about this node and add it to R(u) call Composing and Compute routing paths in order to compute (composed) paths to this node.
  - such that for every node w of B(u) and c(w) = c(v), d(u,v) > d(u,w): add node w to u's Vicinity Ball. Then recomputed B(u) (remove potentially useless nodes in B(u)).

- Vicinity Ball of a node v in (R(u) / B(u)): store B(v) and call the Composition/Mining function in order to determine if there is any Vicinity Balls intersection (it exists a shortest path from u to v, where all nodes are contained in (B(u) $\cap$ B(v))).

- There is a new Landmarks v: if node u does not know this landmark, then store this new landmark and the distance to it. If this landmark is closer to node u than all others known since so far, mark it as node u's closest landmark (L(u) = v).

Add to R(u) all nodes w from L^{-1}(v) such that h(w) = c(u). Then call the Composition/Mining function. If there is new intersection discovered, then call the Compute routing paths function.

When a node u discovers routing Information about a new path to a node, it checks if this node has to be added to node u Vicinity Ball B(u), it then calls the Composition/Mining function in order to determine new Vicinity Balls intersections; finally, it recomposes the routing paths (sequential or composite computation) by means of the Compute Routing Path function.

ii) Discover Routing Information

When a node u receives routing information from its neighbors:

- Path to a node of B(u) or L(u): it stores the path and then call the Composition function.
- Path from any node v of R(u) to L(v): it stores the composed path u $\rightarrow$ L(v) $\rightarrow$ v and then calls the Composition function.

Call the Resolve Routing Paths function for both these points in order to update routing tables for nodes u knows.

When a node u receives topology information by means of the Discover Topology Information (DTI) function, it computes new paths to the new nodes that node u knows from every received notification (node that has been added in B(u), R(u) for which node u does not know any path to it).

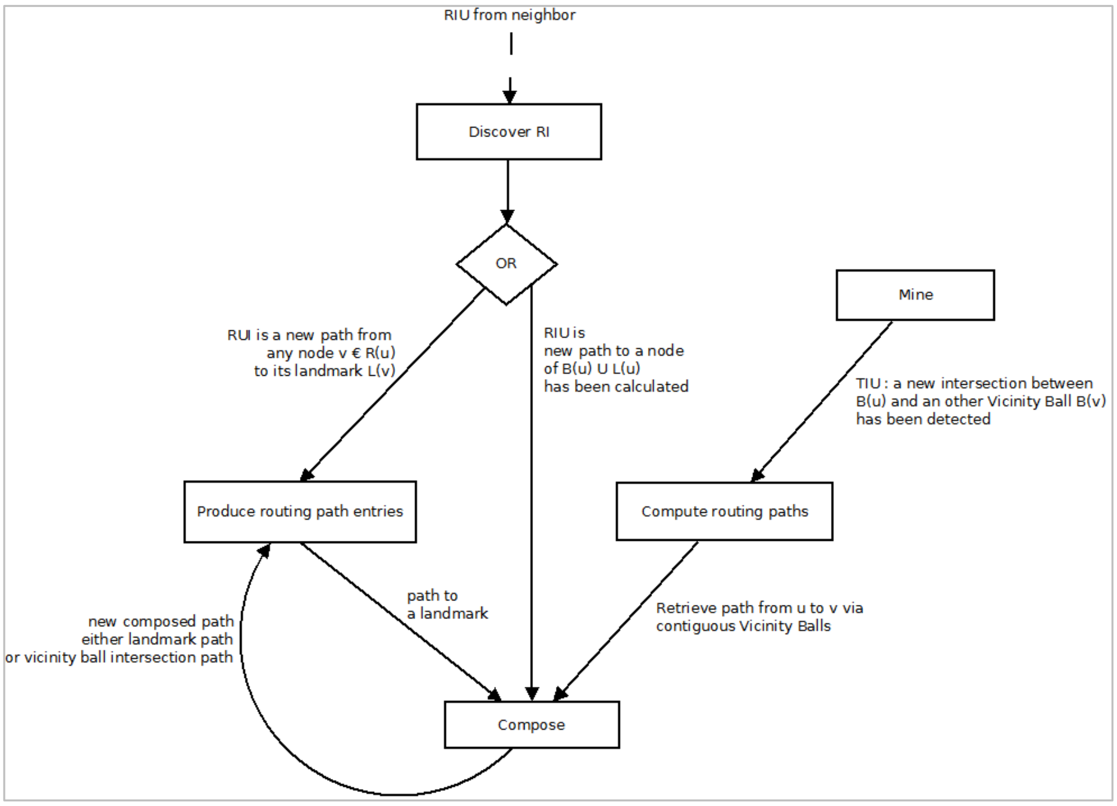

RIU from neighbor
Discover RI
OR
RUI is a new path from any node v € R(u) to its landmark L(v)
RIU is new path to a node of B(u) U L(u) has been calculated
Mine
TIU : a new intersection between B(u) and an other Vicinity Ball B(v) has been detected
Produce routing path entries
Compute routing paths
path to a landmark
Retrieve path from u to v via contiguous Vicinity Balls
new composed path either landmark path or vicinity ball intersection path
Compose

Fig.14.2: Relations of Discover Routing Information function

iii) Composition

After a node u has compute a new Vicinity Ball intersection with B(v) by means of the Mining function, it calls the Compute routing paths function to compute each part of contiguous Vicinity Ball path and then composes the full routing path from node u to v (by composition of several simple routing paths stored in the Routing Information Base (RIB)). Finally, it stores the full routing path in the RIB and derives a route stored as an entry in the routing table.

iv) Compute Routing Paths

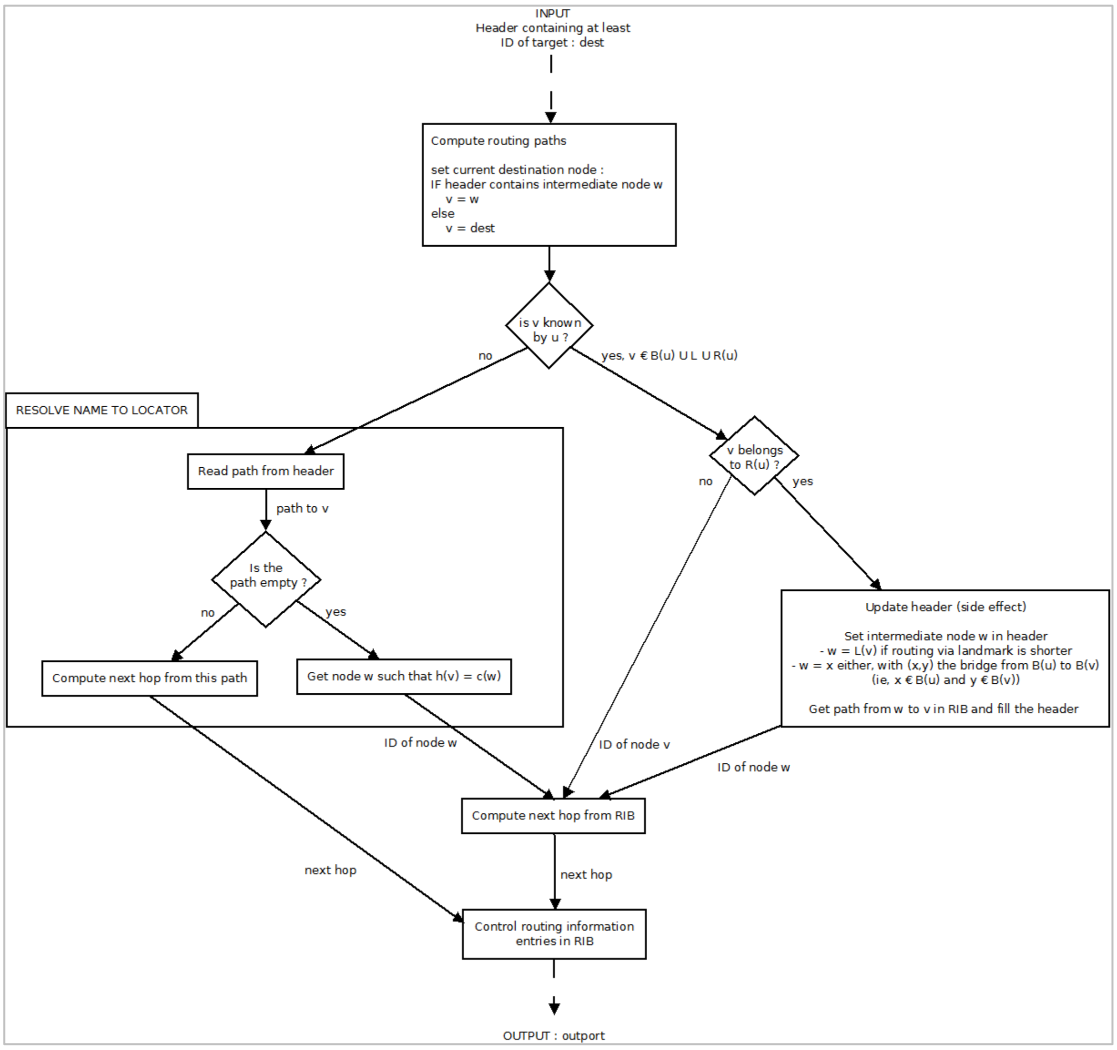

INPUT
Header containing at least
ID of target : dest
Compute routing paths
set current destination node :
IF header contains intermediate node w
v = w
else
v = dest
is v known by u ?
no
yes, v € B(u) U L U R(u)
RESOLVE NAME TO LOCATOR
Read path from header
path to v
Is the path empty ?
no
yes
Compute next hop from this path
Get node w such that h(v) = c(w)
v belongs to R(u) ?
no
yes
Update header (side effect)
Set intermediate node w in header
- w = L(v) if routing via landmark is shorter
- w = x either, with (x,y) the bridge from B(u) to B(v)
(ie, x € B(u) and y € B(v))
Get path from w to v in RIB and fill the header
ID of node w
ID of node v
ID of node w
Compute next hop from RIB
next hop
next hop
Control routing information entries in RIB
OUTPUT : outport

Fig.14.3: Compute Routing Paths

4.3 Compact Multicast Routing

Running compact multicast routing independently of the underlying unicast routing system would be beneficial to efficiently cope with the dynamics of large scale networks. This independence is even the fundamental concept underlying multicast routing schemes such as Protocol Independent Multicast [Fenner06]. Nevertheless, the scaling problems already faced when multicast routing received main attention from the research community, remain largely unaddressed since so far. Indeed, multicast currently operates as an addressable IP overlay (Class D group addresses) on top of unicast routing topology, leaving up to an order of 100 million of multicast routing table entries. Hence, the need to enable point-to-multipoint routing paths (for bandwidth saving purposes) while keeping multicast addressing at the edges of the network and build shared but selective trees inside the network.

Recently introduced in [Abraham09], dynamic compact multicast routing algorithms construct point-to-multipoint routing paths from any source to any set of destinations (or leaves). The tree determined by a point-to-multipoint routing path is commonly referred to as a multicast distribution tree (MDT) as it enables the distribution of multicast traffic from any source to any set of leaf nodes. By means of such dynamic routing scheme, MDTs can dynamically evolve according to the arrival of leaf-initiated join/leave requests. The routing algorithm creates and maintains the set of local routing states at each node part of the MDT. From this state, each node part of the MDT can derive the required entries to forward the multicast traffic received from a given source to its leaves. Two compact multicast routing schemes exist in the scientific literature i) the scheme proposed by I.Abraham, D.Malkhi, and D.Ratajczak (see [Abraham09]), referred hereafter to as AMR, and ii) the scheme proposed by P.Pedroso, P.Papadimitriou, D.Careglio (see [Pedroso11]), referred hereafter to as PPC.

Formally, consider a network topology modeled by an undirected graph $G = (V,E,c)$ where the set V, $|V| = n$, represents the finite set of nodes or vertices (all being multicast capable), the set E, $|E| = m$, represents the finite set of links or edges, and c a non-negative cost function $c: E \rightarrow Z+$ that associates a non-negative cost $c(u,v)$ to each link $(u,v) \in E$. For $u, v \in V$, let $c(u,v)$ denote the cost of the path $p(u,v)$ from u to v in G, where the cost of a path is defined as the sum of the costs along its edges. Let S, $S \subset V$, be the finite set of source nodes, and $s \in S$ denote a source node. Let D, $D \subseteq V \backslash \{S\}$, be the finite set of all possible destination nodes that can join a multicast source s, and $d \in D$ denote a destination (or leaf) node. A *multicast distribution tree* $T_{s,M}$ is defined as an acyclic connected sub-graph of G, i.e., a tree rooted at source $s \in S$ with leaf node set M, $M \subseteq D$.

4.3.1 AMR compact multicast routing

The authors present three different variants for compact multicast scheme: 1) static label-dependent, 2) static name-independent, and 3) dynamic label-dependent. In the context of the present document, we are interested only in the third scheme.

4.3.1.1 AMR scheme overview

The scheme comprises two parts: 1) an off-line construction of a bundle $\mathcal{B}_k$ of sparse covers; 2) an on-line construction of MDT supported by the sparse covers.

The first part consists of the off-line construction of a bundle $\mathcal{B}_k$ of sparse covers $TC_{k,2i}$, defined as $\mathcal{B}_k = \{TC_{k,2i}(G) \mid i \in I\}$ with $k = \log(n)$. Sparse covers are grown from a set of center nodes $c(T_i(v))$ located at distance at most $k2^i$ from node v, where $T_i(v)$ denotes the tree in the collection of rooted trees $TC_{k,2i}(G)$ that contains the ball $B(v,2^i)$. For each $i \in I$ and $T \in TC_{k,2i}(G)$, the center node $c(T(v))$ of each node $v \in T$ stores the labels of all nodes[1] contained in the ball $B(v,2^i)$, the ball centered on node v of radius 2^i. Further, the SPlabel(v) stores the label $\lambda(T,c(T))$ for every tree cover $T \in \mathcal{B}(v)$, defined as set

[1] For simplicity, we present here the label-dependent variant of the scheme. In the name-independence version, center nodes store label mappings from names to nodes.

of all covers T in the bundle $\mathcal{B}_k$ such that v ∈ T. In addition, each node v ∈ V stores the tree routing information μ(T,v) for all the trees in its own label SPlabel(v).

The second part consists of the on-line construction of a new branch of an MDT. When a leaf node u desires to join an MDT, it first determines whether or not one of the MDT nodes is already included in its local tree routing information table.

- If this is the case, it sends the join request to the center node c(T_i(v)) with minimum degree cover i ∈ I that is associated to that MDT node v. The center node c(T_i(v)) then passes the label μ(T_i(v),u) so that the selected MDT node v can forward the multicast traffic to the newly joining leaf node u (without further propagating this label to the source node s).

- Otherwise (some the leaf node covers define an empty intersection with the MDT), the leaf node u with SPLabel(u) queries the source node s to obtain the set of MDT nodes it currently includes. Among all index i ∈ I, it then selects the tree T_{i*}(v) whose intersection with its bundle $\mathcal{B}$(u) is minimum. Once the node, say v, part of this intersection is selected (T_{i*}(v) ∈ $\mathcal{B}$(u)), leaf node u directs the join request to the associated center node c(T_{i*}(v)). The latter passes a label μ(T_{i*}(v),u) so that the selected node v can forward the incoming multicast traffic to the newly joining leaf node u.

In both cases, in order for the source node s to reach node u, node v has to propagate the tuple [v,c(T_{i*}(v)),μ(T_{i*}(v),u)] to the source s. The leaf node u updates all nodes covered by its balls B(u,2^i) to allow them joining the MDT at node u.

4.3.1.2 *Hierarchical decomposition*

The construction of the sparse covers requires the knowledge of the network topology. Sparse covers can be computed off-line and configured statically or discovered by means of the DTI and DRI function and further composed by means of the PDAI function. In addition, any change of the topology may require recomputing the sparse covers. It also requires the DRP function to compute the tree covers for each node which means that each node stores the tree routing information and creates RT entries accordingly (PRTE function). Moreover, center nodes must store the labels of all nodes contained in their balls (PRTE function).

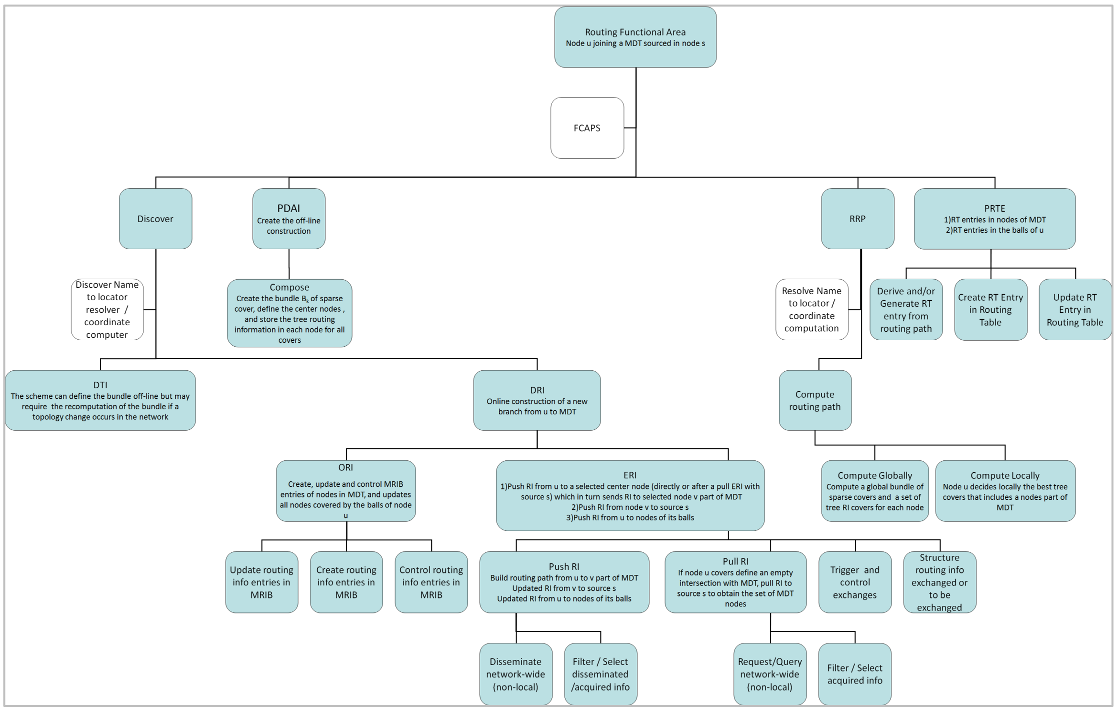

Routing Functional Area
Node u joining a MDT sourced in node s
FCAPS
Discover
PDAI
Create the off-line construction
RRP
PRTE
1)RT entries in nodes of MDT
2)RT entries in the balls of u
Discover Name to locator resolver / coordinate computer
Compose
Create the bundle B_k of sparse cover, define the center nodes , and store the tree routing information in each node for all covers
Resolve Name to locator / coordinate computation
Derive and/or Generate RT entry from routing path
Create RT Entry in Routing Table
Update RT Entry in Routing Table
DTI
The scheme can define the bundle off-line but may require the recomputation of the bundle if a topology change occurs in the network
DRI
Online construction of a new branch from u to MDT
Compute routing path
ORI
Create, update and control MRIB entries of nodes in MDT, and updates all nodes covered by the balls of node u
ERI
1)Push RI from u to a selected center node (directly or after a pull ERI with source s) which in turn sends RI to selected node v part of MDT
2)Push RI from node v to source s
3)Push RI from u to nodes of its balls
Compute Globally
Compute a global bundle of sparse covers and a set of tree RI covers for each node
Compute Locally
Node u decides locally the best tree covers that includes a nodes part of MDT
Update routing info entries in MRIB
Create routing info entries in MRIB
Control routing info entries in MRIB
Push RI
Build routing path from u to v part of MDT
Updated RI from v to source s
Updated RI from u to nodes of its balls
Pull RI
If node u covers define an empty intersection with MDT, pull RI to source s to obtain the set of MDT nodes
Trigger and control exchanges
Structure routing info exchanged or to be exchanged
Disseminate network-wide (non-local)
Filter / Select disseminated /acquired info
Request/Query network-wide (non-local)
Filter / Select acquired info

Fig.15: Hierarchical decomposition for the AMR compact multicast routing scheme

The on-line construction of a new branch of an MDT requires thus the discovery and the selection of the path (DRP function) from the joining node u to the most appropriate center node which in turn selects an appropriate label for the source node. Such join event requires the updates of all nodes in the ball of the joining node (PRTE function) and source node updates its forwarding table.

4.3.1.3 Functional Block diagrams

Regarding the block diagram, the AMR makes use of all general block diagrams discussed in Section 1.3.1. As commented above, the bundle of sparse covers can be uploaded in the nodes off-line or DTI, DRI and PDAI functions may be required as well as the DRP function to determine the tree covers for each nodes. In addition, if a change occurs in the network topology, the DTI and DRI functions require rediscovering the topological change and, in turn, recomputing the sparse covers and trees routing information. It means that each node stores the tree routing information and creates RT entries accordingly (PRTE). Moreover, center nodes must store the labels of all nodes contained in their balls (PRTE). The joining node u selects or computes (using the DRP function) the routing path from u to the center node c(T_{i*}(v)) associated to the selected node v (either directly or after query the source node s and find the set of nodes in MDT). The latter then sends a label μ(T_{i*}(v),u) to the node v belonging to MDT in order to forward multicast traffic to node u. Finally, in the name-independent version of the AMR scheme, application of the RNLC function is required to perform the node name to locator (label) resolution.

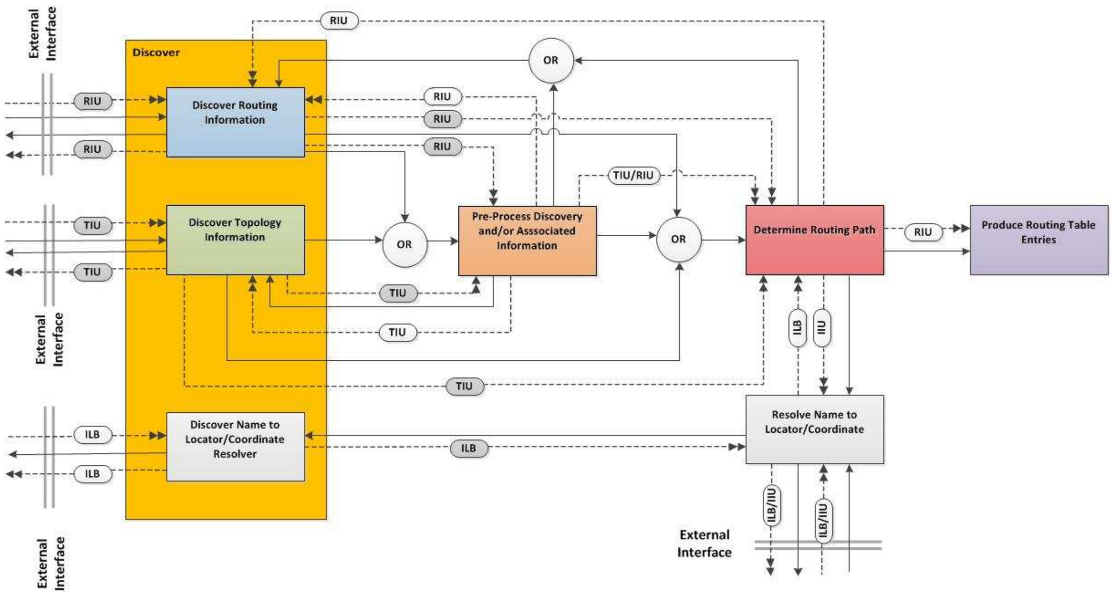

External Interface
Discover
RIU
Discover Routing Information
RIU
RIU
RIU
RIU
OR
TIU/RIU
Discover Topology Information
TIU
TIU
OR
Pre-Process Discovery and/or Associated Information
OR
Determine Routing Path
RIU
Produce Routing Table Entries
TIU
TIU
ILB
IIU
TIU
External Interface
Discover Name to Locator/Coordinate Resolver
ILB
ILB
ILB
Resolve Name to Locator/Coordinate
ILB/IIU
ILB/IIU
External Interface
External Interface

Fig.16: Functional blocks for the AMR compact multicast routing scheme

i) Discover Topology Information (DTI) function (see Fig.16.1): the construction of the sparse covers requires the global knowledge of the topology. It can be done off-line or by a DTI function. Moreover, the DTI function is required each time a topology change occurs.

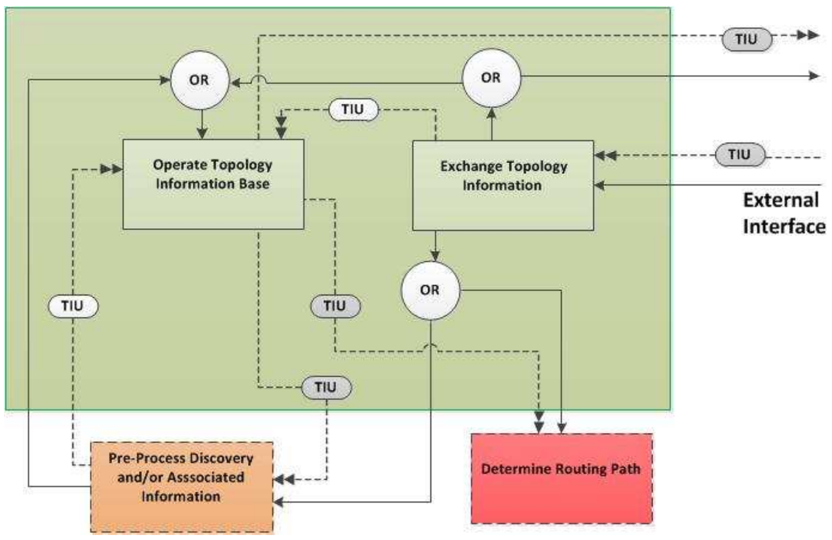

TIU
OR
OR
TIU
Operate Topology Information Base
Exchange Topology Information
TIU
External Interface
OR
TIU
TIU
TIU
Pre-Process Discovery and/or Asssociated Information
Determine Routing Path

Fig.16.1: Discover Topology Information (DTI) function for the AMR compact multicast routing scheme

ii) Discover Routing Information (DRI) function (see Fig.16.2): the construction of the tree routing information for all the trees in each node and the creation of the new branch for the joining node require the DRI function which implies both the exchange of the routing information and the operation of the routing information base.

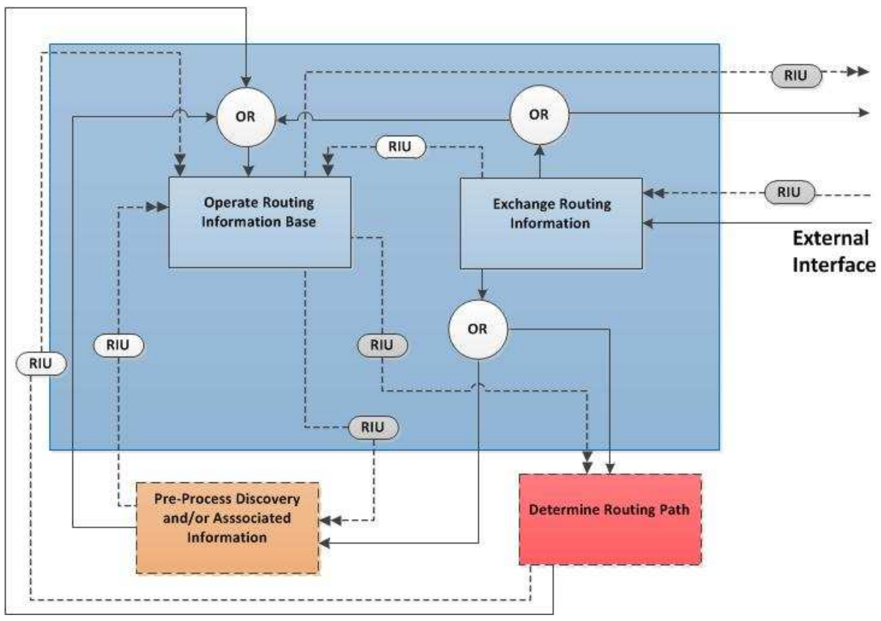

RIU
OR
OR
RIU
Operate Routing Information Base
Exchange Routing Information
RIU
External Interface
OR
RIU
RIU
RIU
RIU
Pre-Process Discovery and/or Asssociated Information
Determine Routing Path

Fig.16.2: Discover Routing Information (DRI) function for the AMR compact multicast routing scheme

iii) Pre-process Discovered and Associated Information (PDAI) function (see Fig.16.3): the PDAI function requires the composition of topology and routing information to create the bundle of sparse covers, to define the center nodes, and the construction the tree routing information.

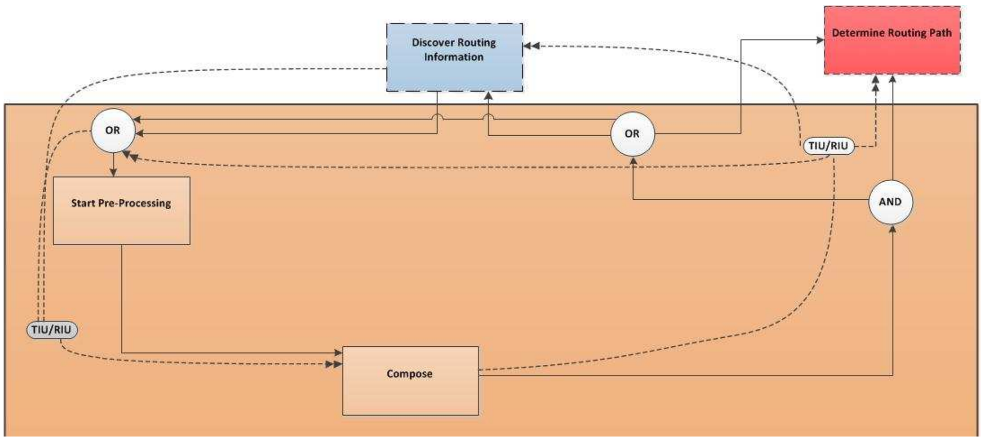


Fig.16.3: Pre-process Discovered and Associated Information (PDAI) function for the AMR compact multicast routing scheme

iv) Determine Routing Path (DRP) function (see Fig.16.4): the DRP function is required to compute the tree routing information for the trees of the sparse covers and to compute the best tree cover that include a node of the MDT. Moreover, using the DRP function, the joining node u selects or computes the routing path from u to the center node $c(T_{i*}(v))$ associated to the selected node v.

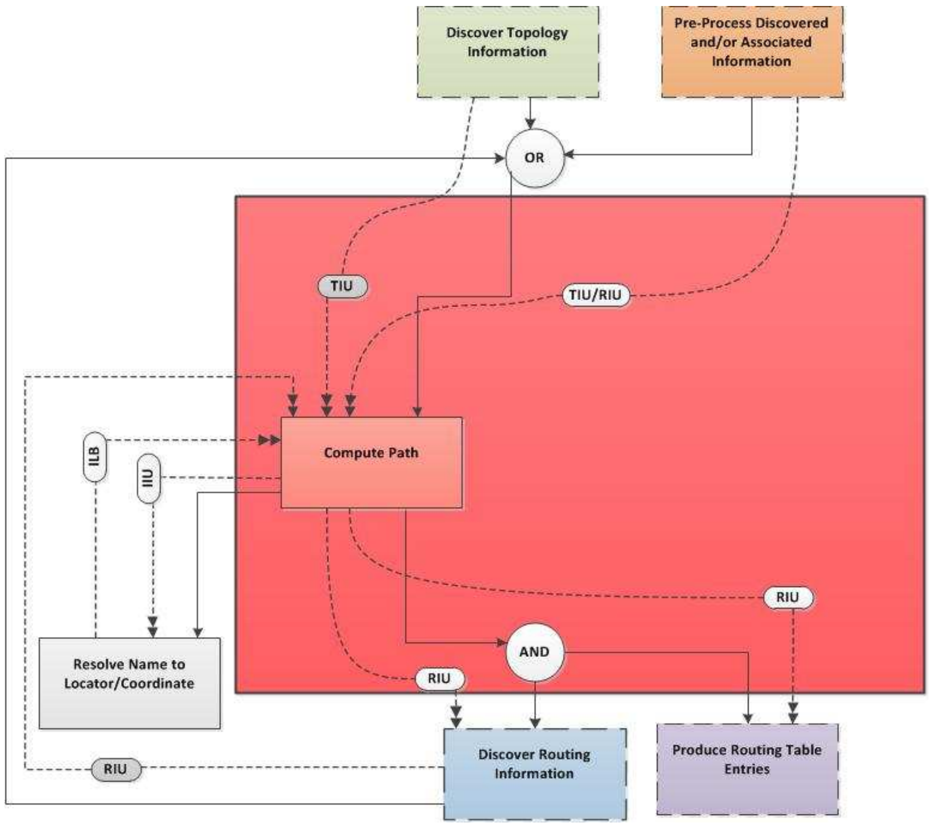

Discover Topology Information
Pre-Process Discovered and/or Associated Information
OR
TIU
TIU/RIU
ILB
IIU
Compute Path
RIU
AND
RIU
Resolve Name to Locator/Coordinate
RIU
Discover Routing Information
Produce Routing Table Entries

Fig.16.4: Determine Routing Path (DRP) function for the AMR compact multicast routing scheme

v) Produce Routing Table Entry (PRTE) function (see Fig.16.5): once node u selects the node v that belongs to the multicast distribution tree, relative MRIB entries are created so as to propagate the join message from node u to its associated center node $c(T_{i*}(v))$. Upon reception of the label $\mu(T_{i*}(v),u)$ node v belonging to MDT creates a routing table entry so as to direct incoming multicast traffic to the newly joining leaf node u. Moreover, a routing table entry is created by each node belonging to the balls $B(u,2^i)$ of joining node u and at source node s.

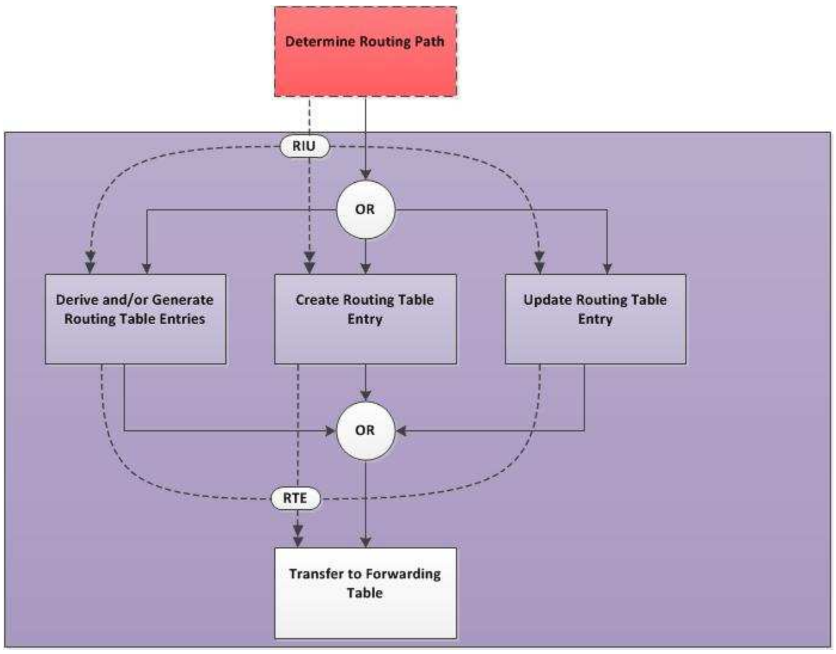

Determine Routing Path
RIU
OR
Derive and/or Generate Routing Table Entries
Create Routing Table Entry
Update Routing Table Entry
OR
RTE
Transfer to Forwarding Table

Fig.16.5: Produce Routing Table Entry (PRTE) function for the AMR compact multicast routing scheme

vi) Resolve Name to Locator / Coordinate Resolve (RNLC) function: in the name-independent version of the compact multicast routing node a resolution function is required to translate node name to locator (i.e., topology dependent label).

4.3.2 PPC compact multicast routing

PPC is a name-independent compact multicast routing algorithm for leaf-initiated, distributed and dynamic construction of MDT. The objective of this scheme is to minimize the routing table (RT) size at each node $v \in V$, $|V| = n$, at the expense of i) routing multicast packets on point-to-multipoint paths with relative small deviation compared to the optimal stretch obtained with the Steiner Tree (ST), and ii) higher communication cost compared to the shortest path tree (SPT). For this purpose, the PPC algorithm reduces the storage of routing information by maintaining during the MDT construction the direct neighbor-related entries per node v, i.e., only the local routing information (degree(v) entries) instead of global routing information ($|n-1|$ entries).

4.3.2.1 PPC scheme overview

Let consider a node u joins the multicast distribution tree $T_{s,M}$, where $M \subseteq D$ corresponds to the current set of nodes part of the MDT, and D the possible set of destination nodes. If node u is already part of $T_{s,M}$ ($u \in V_T$) then it is either a transit or a branching node of the MDT. Otherwise, node u is not part of $T_{s,M}$ ($u \in D \setminus V_T$) and it must search for the least cost branching path from node u to node $v \in T_{s,M}$. Among the set $P_{u,v}$ of possible paths p(u,v) from node $u \notin T_{s,M}$ to node $v \in T_{s,M}$, the least cost branching path p(u,v)* is defined as follows: $p(u,v)^* = \min\{c(u,v) \mid p(u,v) \in P_{u,v}\}$. In this equation, the cost c(u,v) of the path p(u,v) is defined as the sum of the cost c(u,w) of the edge (u,w), where w=succ(u) refers to the upstream neighbor node of u, and the cost c(w,v) of the path p(w,v). When each node along the least-cost branching path p(u,v)* from leaf node u to $v \in T_{s,M}$ determines its upstream neighbor node along that path, leaf node u can send to its selected upstream neighbor node a request message to join $T_{s,M}$. The join message is relayed along the selected least-cost branching path p(u,v)* until it reaches node $v \in T_{s,M}$. Once node u has joined $T_{s,M}$, $u \in V_T$, the set M comprises node u.

To reduce the number of messages (and thus the communication cost), the discovery process is divided into two parts:

- Local search: consists in a limited search within a certain perimeter around the joining leaf node u. The contiguous set of nodes covered by the local search is called the vicinity ball B of node u, B(u). If node u declares the multicast source s unreachable at the end of this local search (i.e., none of the MDT nodes belongs to B(u)), it then starts the global search.

- Global search: consists in searching of an MDT's branching node outside the vicinity of the joining leaf node.

In addition, as the most costly searches are resulting from the initial set of leaf nodes joining the MDT, each source constructs a source ball B(s) such that when a message reaches the boundary of that domain it is directly routed to the source. The size shall be at least as big as the average leaf node size. Each node in the vicinity ball of the source node s, must now maintain an additional MRIB entry to relay discovery messages towards the source node s.

At the end of this process, routing table of each node $v \in T_{s,M}$ ($v \in V_T$) includes i) one routing table entry (stored in the multicast routing information base or MRIB) that indicates the upstream neighbor node to which the join message is/to be sent for each source node s; this locally stored information enables performing Reverse Path Forwarding check so as to ensure loop-free forwarding of the incoming multicast packets, and ii) one multicast traffic routing entry (stored in the tree information base or TIB) to enable forwarding of incoming multicast traffic (generated from that source s) from its incoming port to a set of outgoing ports.

4.3.2.2 Hierarchical decomposition

The information needed by the joining node u to reach the multicast distribution tree $T_{s,M}$ sourced at node s is acquired by means of a search mechanism (DRI) that returns the upstream neighbor node along the least cost branching path (DRP function) to the MDT. Consequently, no off-line construction is required. After a node becomes member of the MDT, nodes involved in the least cost branching path stores i) one routing table entry (stored in the multicast routing information base or MRIB) that indicates the upstream neighbor node to which the join message is sent for each source node s (PRTE function); 2) one multicast traffic route then derived and the corresponding entry generated that is stored in the tree information base (PRTE function); a forwarding entry is then derived that allows to forward the incoming multicast traffic (generated from that source s) from its incoming port to a set of outgoing port directed toward the leaf nodes of the multicast distribution tree. Finally, as commented above, the source node constructs a vicinity ball (PDAI and DRI function).

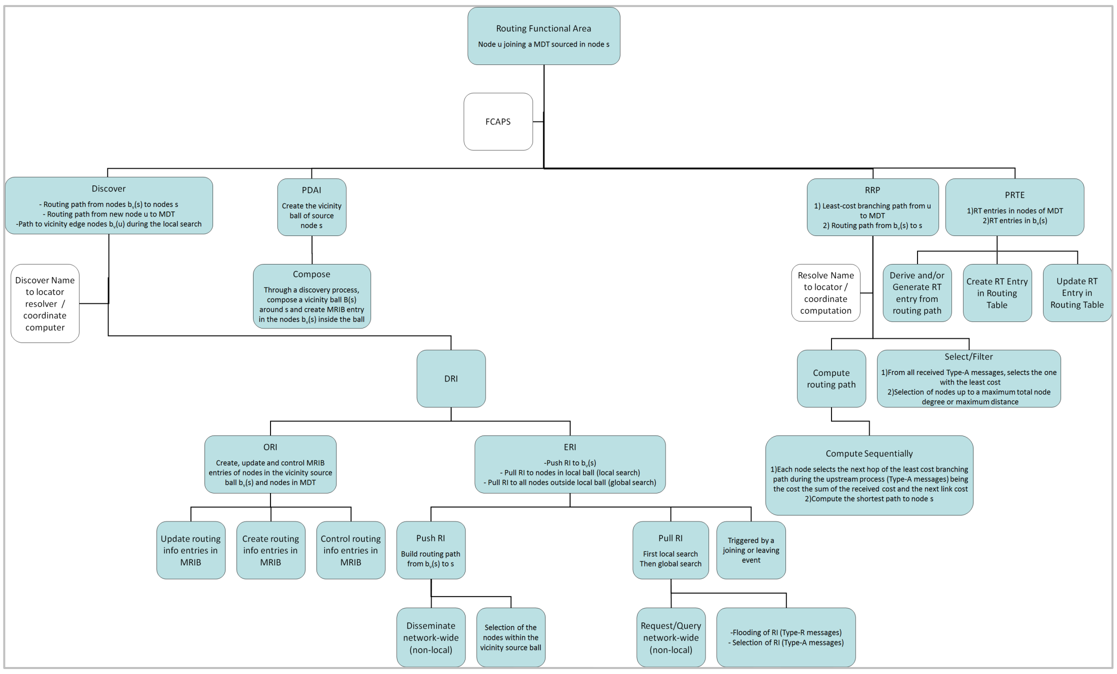

Routing Functional Area
Node u joining a MDT sourced in node s
FCAPS
Discover
- Routing path from nodes b_v(s) to nodes s
- Routing path from new node u to MDT
-Path to vicinity edge nodes b_v(u) during the local search
PDAI
Create the vicinity ball of source node s
RRP
1) Least-cost branching path from u to MDT
2) Routing path from b_v(s) to s
PRTE
1)RT entries in nodes of MDT
2)RT entries in b_v(s)
Discover Name to locator resolver / coordinate computer
Compose
Through a discovery process, compose a vicinity ball B(s) around s and create MRIB entry in the nodes b_v(s) inside the ball
Resolve Name to locator / coordinate computation
Derive and/or Generate RT entry from routing path
Create RT Entry in Routing Table
Update RT Entry in Routing Table
DRI
Compute routing path
Select/Filter
1)From all received Type-A messages, selects the one with the least cost
2)Selection of nodes up to a maximum total node degree or maximum distance
ORI
Create, update and control MRIB entries of nodes in the vicinity source ball b_v(s) and nodes in MDT
ERI
-Push RI to b_v(s)
- Pull RI to nodes in local ball (local search)
- Pull RI to all nodes outside local ball (global search)
Compute Sequentially
1)Each node selects the next hop of the least cost branching path during the upstream process (Type-A messages) being the cost the sum of the received cost and the next link cost
2)Compute the shortest path to node s
Update routing info entries in MRIB
Create routing info entries in MRIB
Control routing info entries in MRIB
Push RI
Build routing path from b_v(s) to s
Pull RI
First local search Then global search
Triggered by a joining or leaving event
Disseminate network-wide (non-local)
Selection of the nodes within the vicinity source ball
Request/Query network-wide (non-local)
-Flooding of RI (Type-R messages)
- Selection of RI (Type-A messages)

Fig.17: Hierarchical decomposition for the PPC compact multicast routing scheme

4.3.2.3 *Functional Block diagrams*

Regarding the block diagram, the PPC makes use of the DRI and PDAI functions to create the vicinity ball of source node s and create the RT entries (PRTE) in the nodes in this ball. Also, it uses the DRI function to discover and compute the least cost branching path (DRP) from joining node u to the node v belonging to the MDT and creates the RT entries (PRTE) accordingly in the path between u and v. As commented above, the discovery process is divided into two parts, local and, if it fails, global search. To start the global search from the border of the vicinity ball of the local search, temporary paths are computed from u to vicinity edge nodes. Finally, PPC is a name-independent routing scheme, so RNLC is required to perform the node name to locator resolution.

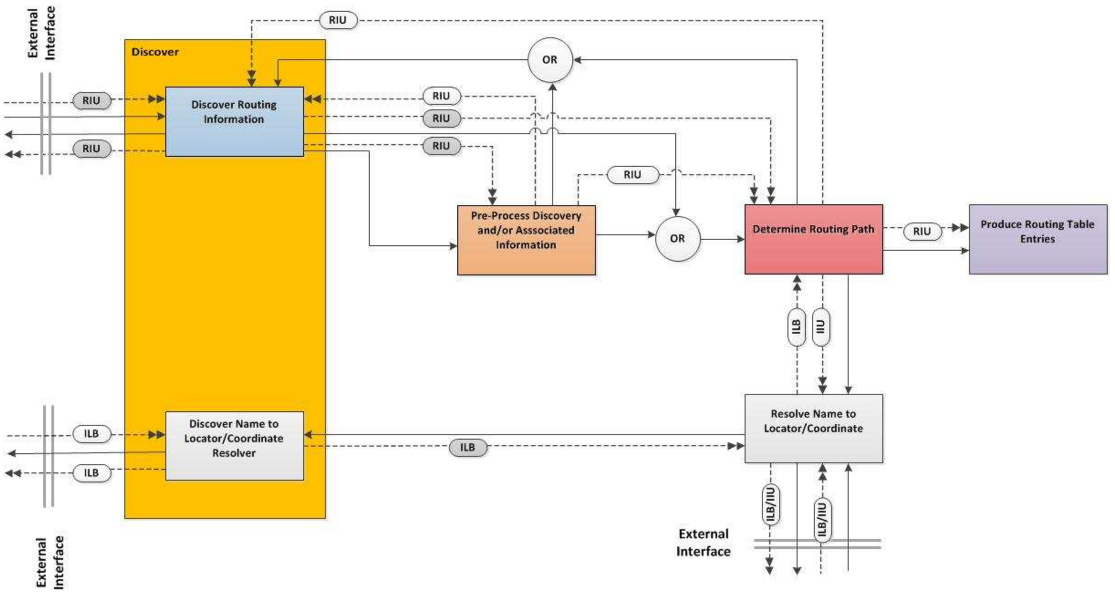

External Interface
Discover
Discover Routing Information
Discover Name to Locator/Coordinate Resolver
RIU
ILB
OR
Pre-Process Discovery and/or Asssociated Information
Determine Routing Path
Produce Routing Table Entries
Resolve Name to Locator/Coordinate
IIU
ILB/IIU
External Interface

Fig.18: Functional blocks for the PPC compact multicast routing scheme

i) Discover Routing Information (DRI) function: the PPC routing scheme is based on a discovery process that collects the needed routing information to subsequently enable the computation of the least cost branching path from joining node u to a node v which belongs to the MDT. Moreover, during the pre-process, source node s determines the source ball through the exchange of routing information which also entails the operation in the MRIB and the computation of the routing path.

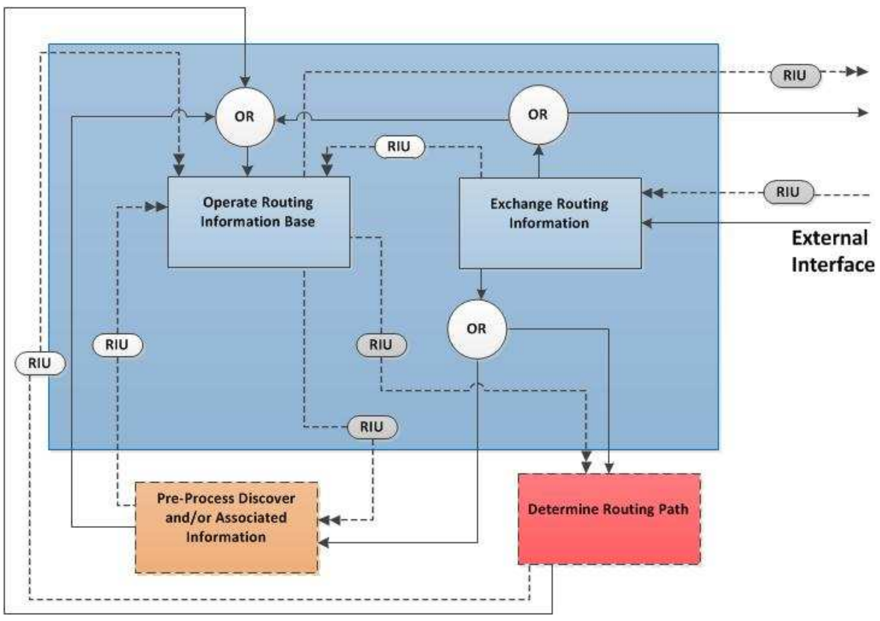

OR
RIU
Operate Routing Information Base
Exchange Routing Information
External Interface
Pre-Process Discover and/or Associated Information
Determine Routing Path

Fig.18.1: Discover Topology Information (DTI) function for the PPC compact multicast routing scheme

ii) Pre-process Discovered and Associated Information (PDAI) function: according to the hierarchical decomposition shown previously, the PPC routing scheme uses the PDAI function to determine the source ball of source node s. This operation is performed through a discovery process initiated at the when a given node decides to join the multicast distribution tree for the source s.

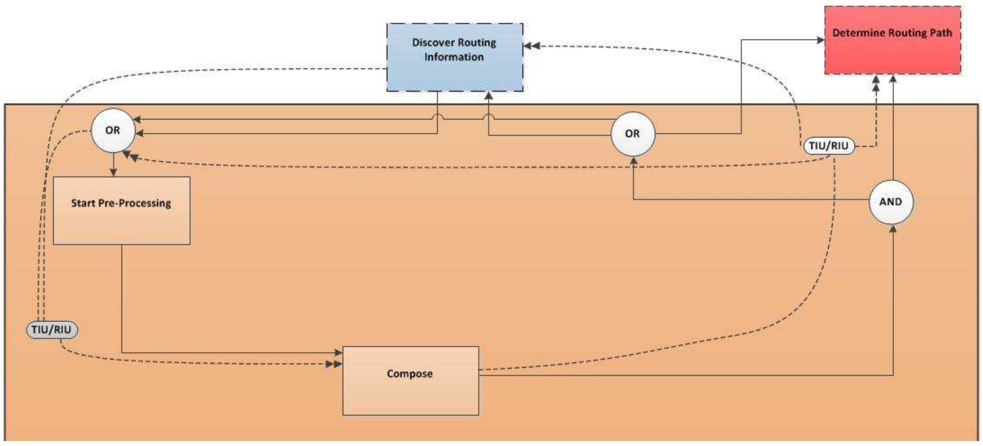


Fig.18.2: Pre-process Discovered and Associated Information (PDAI) function for the PPC compact multicast routing scheme

iii) Determine Routing Path (DRP) function: the computation of the routing path from node u to node v is performed sequentially together with a selection process: each node selects as next hop the neighboring node along the least cost branching path (sum of the cost received from the neighbor to the node v belonging to the multicast distribution tree and the cost to reach that neighbor). After this selection an MRIB entry is created along the least cost branching path from allowing node u to initiate a join message towards node v. Moreover, there is an execution of the DRP function during the determination of the source ball.

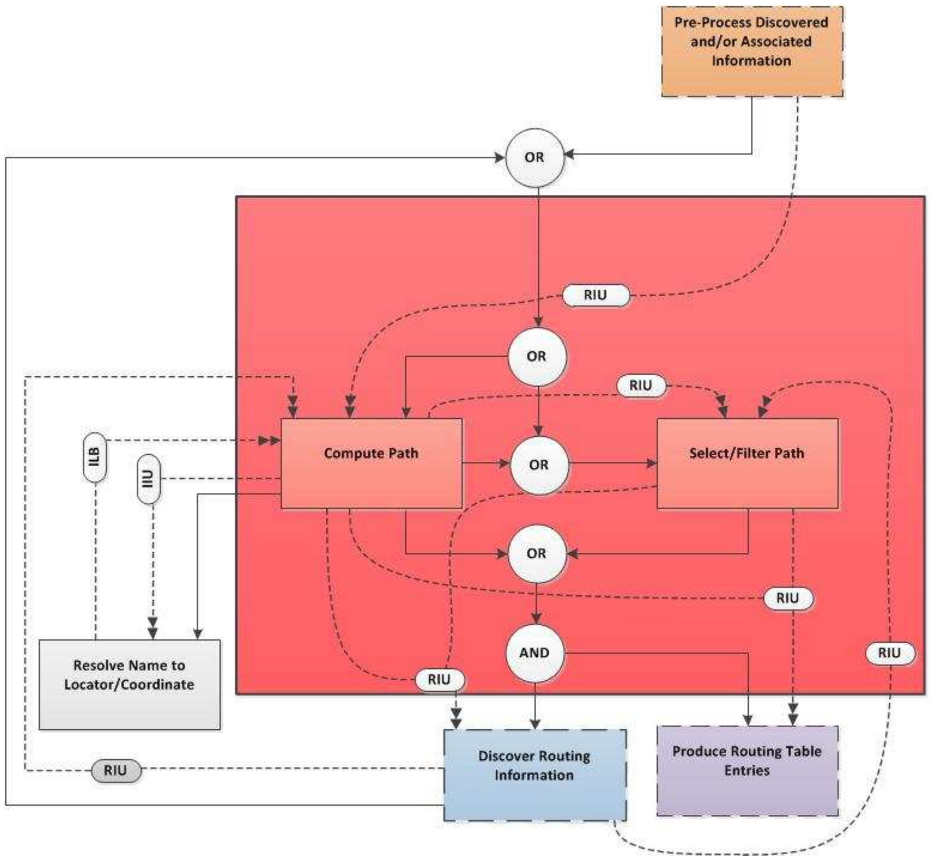

Pre-Process Discovered and/or Associated Information
OR
RIU
OR
RIU
ILB
IIU
Compute Path
OR
Select/Filter Path
OR
RIU
AND
RIU
RIU
Resolve Name to Locator/Coordinate
RIU
Discover Routing Information
Produce Routing Table Entries

Fig.18.3: Determine Routing Path (DRP) function for the PPC compact multicast routing scheme

iv) Produce Routing Table Entry (PRTE) function: after the least cost branching path selection process completes, an MRIB entry is created allowing the join message initiated by node u to reach node v (that belongs to the multicast distribution tree). At each node along this path, after reception of the join message initiated by node u, a multicast traffic routing entry (stored in the tree information base or TIB) is created to enable forwarding of incoming multicast traffic (generated from that source s) from its incoming port to a set of outgoing ports.

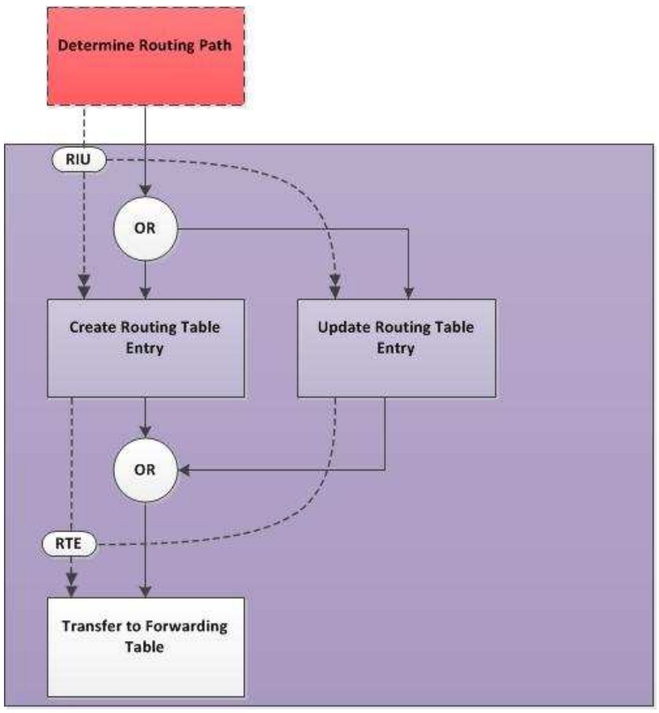

Determine Routing Path
RIU
OR
Create Routing Table Entry
Update Routing Table Entry
OR
RTE
Transfer to Forwarding Table

Fig.18.4: Produce Routing Table Entry (PRTE) function for the PPC compact multicast routing scheme

4.4 Geometric Routing

In geometric routing, nodes are assigned coordinates in a metric space, and these coordinates are used as addresses to perform point-to-point routing in this space. Geometric routing finds its roots in the research efforts directed towards the specification of lightweight routing protocols for wireless meshed environments. However, topologies of wireless networks comprises of the order of 10 to 100 nodes and show exponential node degree distribution compared to the Internet which comprises of the order of 10k autonomous systems (and of the order of 100k nodes) with a power-law degree distribution. Moreover, the Internet size and scope render the deployment of new network technologies, and in particular, new routing schemes, extremely challenging. Henceforth, applicability of geometric for Internet routing is still an open research area for which many variants have been proposed over last decade.

4.4.1 Geometric Routing on Greedy Embedding

A greedy embedding of an undirected graph G=(V,E) in a metric space (X,d) is a mapping function f: $V(G) \rightarrow X$ with the following property: for every pair of distinct vertices s, $t \in V(G)$ there exists a vertex u adjacent to s such that $d(f(u),f(t)) < d(f(s),f(t))$. The embedding and thus the mapping function f is distance preserving if $\forall$ x, $y \in V(G)$, $d_G(x,y) \rightarrow d_X(f(x),f(y))$.

Following this definition, a greedy embedding of a graph G in a metric space (X,d) is a mapping of G in (X,d) such that a distance decreasing path exists between every pair of vertices in G (a distance decreasing path from s to t in a greedy embedding of G is a path ($s = v_1, v_2, .., t=v_k$) such that $d(v_i,t) > d(v_{i+1},t)$, for i = 1, 2, .., k-1, see [Papadimitriou05]. Greedy routing can thus be seen as a natural abstraction of geometric routing in which nodes are assigned virtual coordinates in a metric space, and these coordinates are used as addresses to perform point-to-point routing in this space (along the most distance-decreasing path). The main challenge in greedy routing consists in finding the appropriate mapping function f together with a polynomial-time algorithm for embedding V(G) in the space X so as to allow for greedy routing using the metric d associated to that space.

4.4.1.1 Geometric Routing on Greedy Embedding in the Hyperbolic Plane

Introduced by R.Kleinberg in 2005 [Kleinberg05], greedy embedding in hyperbolic metric space was a crucial step in search of means to overcome known limitations of geometric routing on undirected graph G = (V,E) embedded in the Euclidean space. Advancing his work on Internet topology embedding in hyperbolic spaces, R.Kleinberg further demonstrated in 2007 that every connected finite graph has a greedy embedding in the hyperbolic plane [Kleinberg07]; however, the reconstruction such embedding upon topology modification (link and/or node joining or leaving the topology) still requires O(n) operations per update. Indeed, the main challenge in greedy routing does not only consists in finding the appropriate mapping function f together with a polynomial-time algorithm for embedding V(G) in the space X but also updating this embedding over time so as to prevent performance degradation of greedy routing.

In this greedy routing scheme, referred to as Kleinberg greedy routing, every node is assigned a coordinate in Hyperbolic Plane and the routing is performed with respect to the coordinates of the destination and neighbors of a node. Using the coordinates, a node decides to forward the packet to a neighbor which is closer to the destination than others. Using greedy embedding for coordinate determination, avoids the situation that a packet gets stuck in a local minimum. Robert Kleinberg proved that every connected finite graph has a greedy embedding in Hyperbolic Plane. In order to apply the Kleinberg’s approach in a network and calculate the coordinates of nodes next steps should be followed.

1- First a spanning tree of the network should be constructed
2- Then the maximum degree (d) of this tree is determined
3- Finally infinite d-regular tree in hyperbolic plane is generated and every node in the spanning tree is labeled with the corresponding coordinate of the d-regular tree node.

The functional process of this greedy routing can be decomposed to the following processes. Fig.19.1 depicts the hierarchical decomposition of the greedy routing functional area.

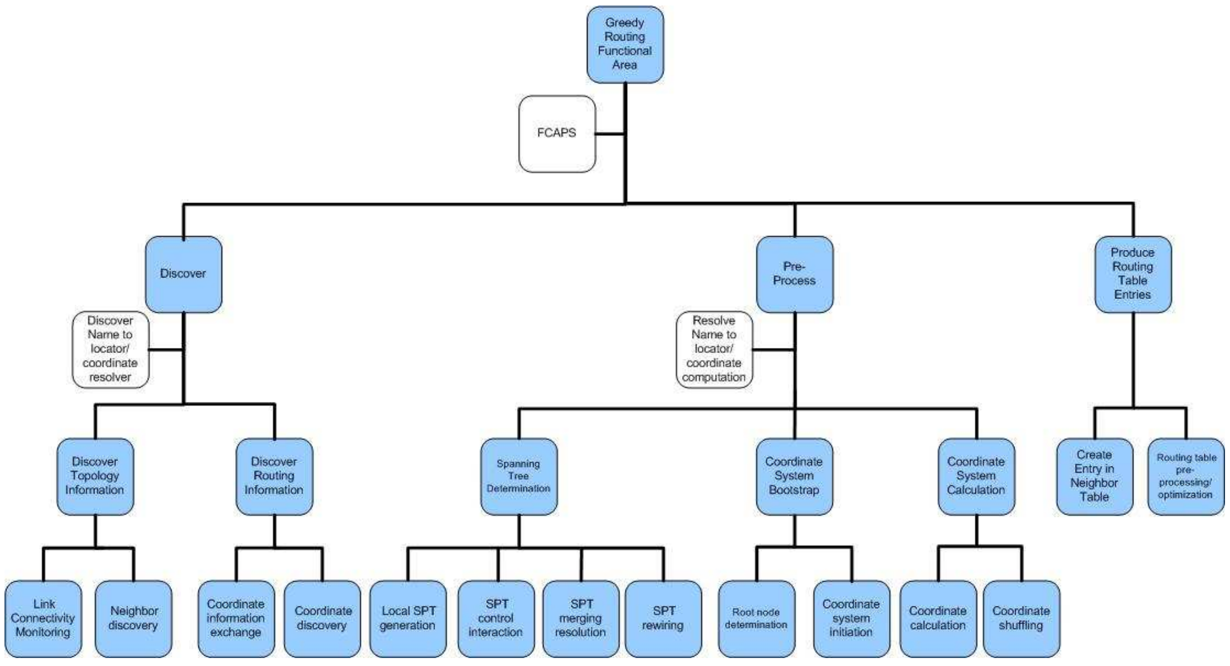

Greedy Routing Functional Area
FCAPS
Discover
Pre-Process
Produce Routing Table Entries
Discover Name to locator/ coordinate resolver
Resolve Name to locator/ coordinate computation
Discover Topology Information
Discover Routing Information
Spanning Tree Determination
Coordinate System Bootstrap
Coordinate System Calculation
Create Entry in Neighbor Table
Routing table pre-processing/ optimization
Link Connectivity Monitoring
Neighbor discovery
Coordinate information exchange
Coordinate discovery
Local SPT generation
SPT control interaction
SPT merging resolution
SPT rewiring
Root node determination
Coordinate system initiation
Coordinate calculation
Coordinate shuffling

Fig.19.1: Hierarchical decomposition of the Kleinberg greedy routing scheme

In its base version, this variant of greedy routing makes sequentially use the following functional blocks: i) the Discovery Topology Information (DTI) function, the ii) the Pre-process function, the iii) the Discover Routing Information (DRI) function, and the Produce Routing Table Entry (PRTE) function. The Determine Routing Path (DRP) function is not used by this routing scheme. Indeed, incoming traffic forwarding relies at each node on the locally stored coordinates of its neighbors obtained by means of the Discover Routing Information (DRI) function. This information is required to compute the distance of every neighbor to the destination node and select the neighbor node along the most distance-decreasing path to the destination.

The **Discover Topology information** (DTI) functional block is composed of two blocks (see Fig.19.2): the **Link connectivity monitoring** and (block 1.1) and the **Neighbor discovery** (blocks 1.2). Topology Information of this block is the input for Pre-process block. This information is used for construction of the spanning tree which is needed for coordinate determination.

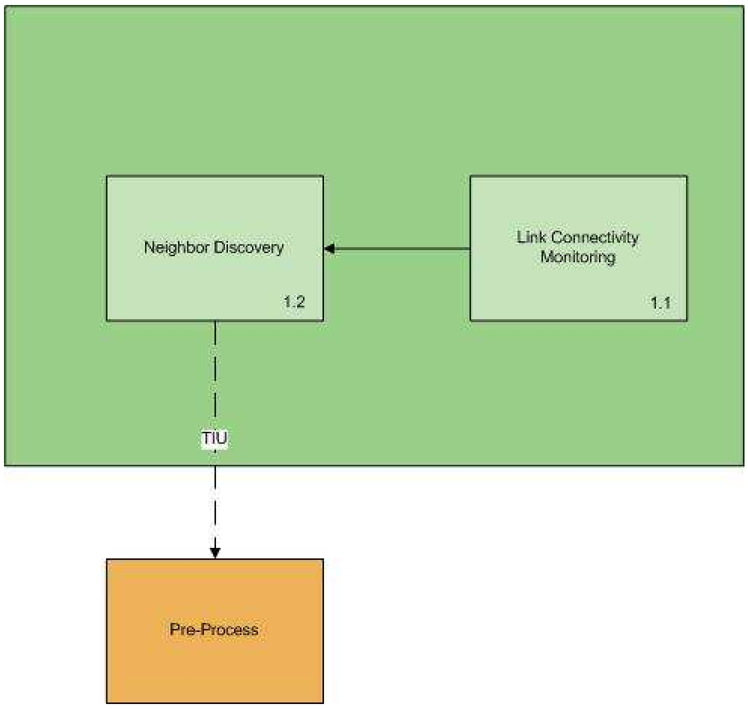


Fig.19.2: Discover Topology Information (DTI) block of the Kleinberg greedy routing scheme

The **Pre-process** function receives information about the topology of the network and uses this information to generate a spanning tree of that network used for coordinate calculation. Therefore, the Pre-process functional block (blocks 3) comprises three functional blocks (see Fig.19.2):

i) The **Spanning tree interaction and determination** (block 3.1): a distributed method used to construct the required spanning tree. As some studies showed that low depth and high degree in spanning tree result in a higher quality greedy routing in terms of stretch and robustness, the goal of the spanning tree generation algorithm is to construct such a spanning tree. In this method some local spanning trees are generated which means that at some point of the time we have a spanning forest of the network, then based on some rules these spanning trees merge together. The algorithm works in such a way that the final spanning tree is rooted at the node with maximum degree and the tree has minimum depth from this root;
ii) Knowing the root node of the spanning tree, the **Coordinate system bootstrapping** (block 3.2) starts the calculation of the coordinates;
iii) Upon spanning tree construction, every node knows its parent and its children in the tree. Receiving some coefficients from the parent, every node is able to calculate its coordinate and the required coefficients for its children by means of the **Coordinate computation block** (block 3.3).

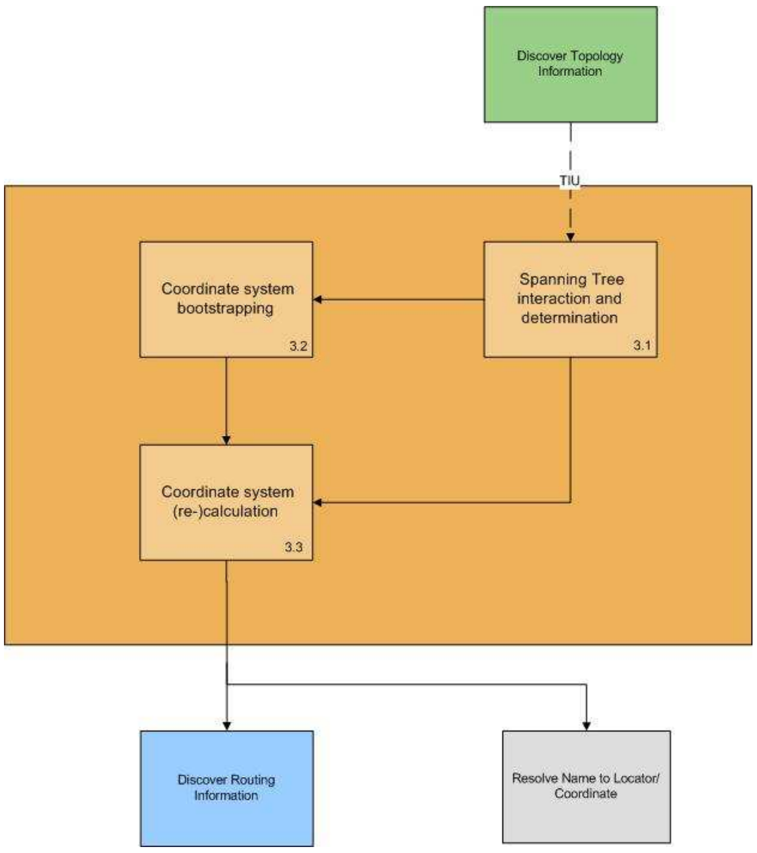

Discover Topology Information
TIU
Spanning Tree interaction and determination
3.1
Coordinate system bootstrapping
3.2
Coordinate system (re-)calculation
3.3
Discover Routing Information
Resolve Name to Locator/ Coordinate

Fig.19.2: Pre-process block of the Kleinberg greedy routing scheme

A distributed method is used to construct the required spanning tree (block 3.1). As some studies showed that low depth and high degree in spanning tree result in a higher quality greedy routing in terms of stretch and robustness, the goal of the spanning tree generation algorithm is to construct such a spanning tree. In this method, depicted in Fig.19.3, some local spanning trees are generated (block 3.1.1) which means that at some point of the time we have a spanning forest of the network, then based on some rules these spanning trees merge together (block 3.1.3). The algorithm works in such a way that the final spanning tree is rooted at the node with maximum degree and the tree has minimum depth from this root.

In greedy routing, Routing Tables are not required any more to enable traffic forwarding. Instead of producing Routing Table entries, every node locally stores the coordinates of its neighbors in order to calculate the distance of every neighbor to the destination node. For this purpose, after coordinate determination (block 3), each node exchanges the coordinates of its neighbors coordinates by means of local discovery of routing information (block 2). As depicted in Fig.19.4, this block is composed of two functional blocks. Using this local information, each node can populate its neighbor (coordinate) table by means of the Produce Routing Table entry (PRTE) function (block 4).

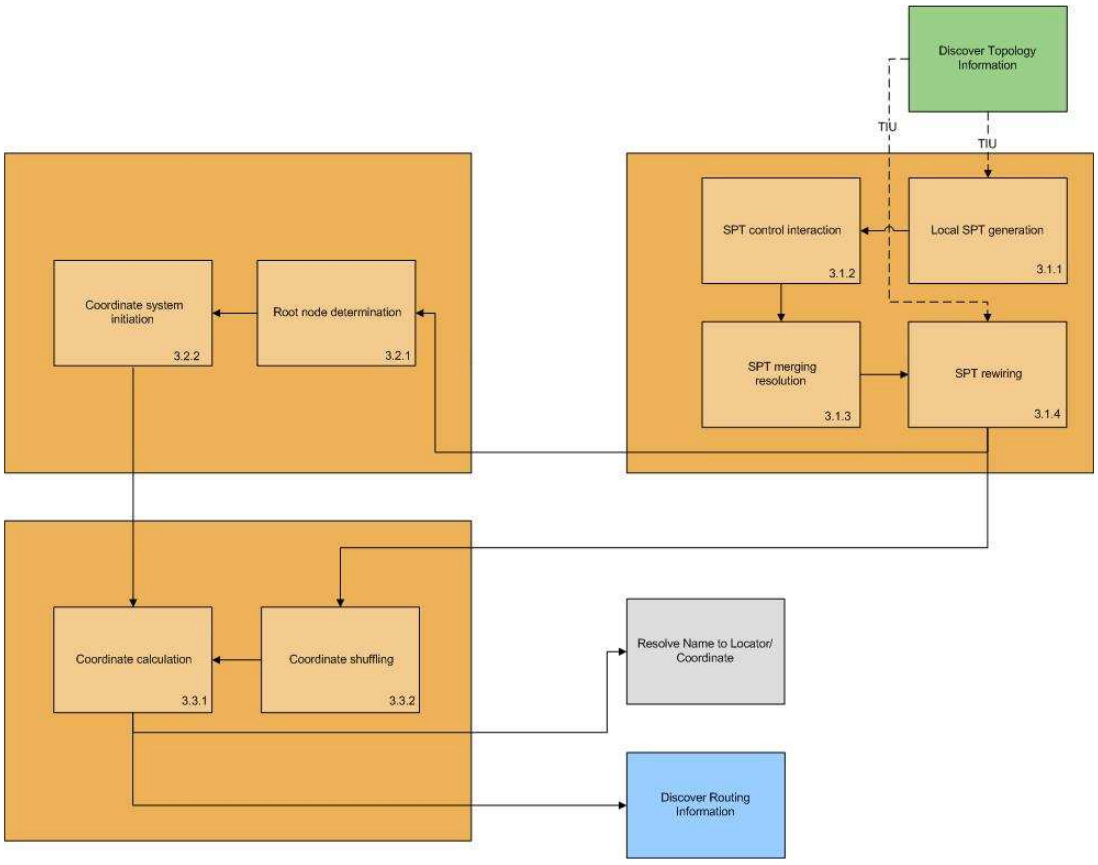


Fig.19.3: Pre-process sub-blocks of the Kleinberg greedy routing scheme

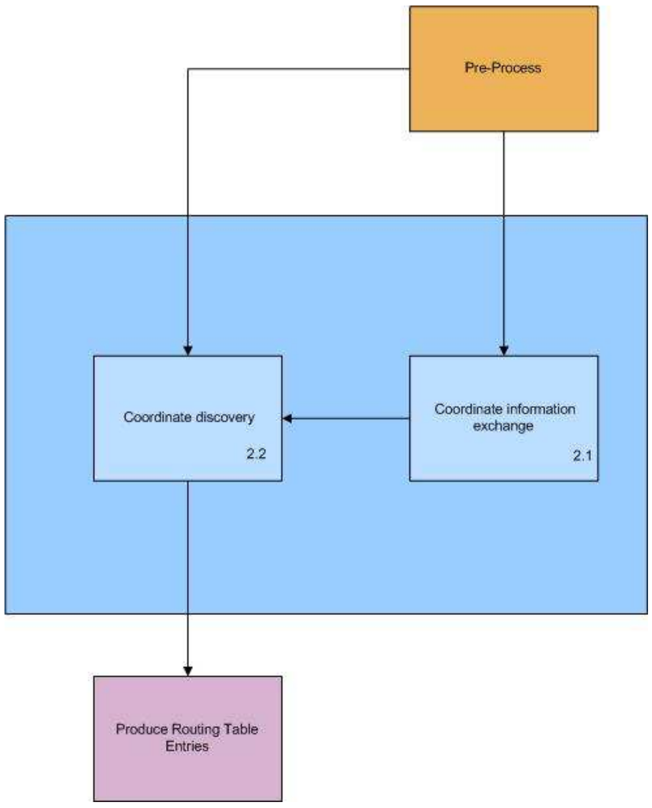

Pre-Process
Coordinate discovery
2.2
Coordinate information exchange
2.1
Produce Routing Table Entries

Fig.19.2: Discover Routing Information (DRI) block of the Kleinberg greedy routing scheme

5. Information Model

Information model enables to represent the properties, relationships, constraints, rules, and operations in order to formally specify the information semantics for a given application domain/area. Hence, an information model is also referred to as semantic information/data model or conceptual data model.

Different methods exist for developing an information model. Among these methods, the entity-relationship (E-R) approach that expresses in terms of entities, the relationships (or associations) among these entities, and the attributes (properties) of both the entities and their relationships [Chen76]. The ultimate goal of applying such model is to capture as much of the meaning of the information (semantic) as possible so as to obtain a better design that is scalable and easier to maintain.

The E-R model in its original form does not support specialization/generalization abstractions (also termed hierarchies). The E-R model has been extended to include all modeling concepts of basic ER with additional concepts: set-subset relationships (sub-classes/super-classes), specialization/ generalization, categories, and attribute inheritance. The resulting model is called the enhanced-ER or Extended ER (E2R or EER) model. It is used to model applications more completely and accurately if needed.

The E-R model is the most commonly used information model. Hence, we will also make use of this model following the objectives of this document in terms of modeling the routing system information; this will requiring that following conditions are met by the information modeling technique:

- The information model should be encompass most of existing routing protocols (e.g., path vector routing, distance-vector routing, link-state routing) but also routing protocols that will be developed in the EULER project context (e.g., compact routing, greedy routing and their variants).

- The information model should be flexible enough to facilitate different interfaces and accommodate other data models that may have a different logic. Different relationships between the entities may determine different interfaces.

- The information model should be easy to be implemented in real world. We talk here generically about any implementation, whether on the Internet or other kinds of networks where a routing scheme is needed, although the Internet is the primary target for us. By easiness of implementation, we mean in terms of technicalities than in terms of economic concerns or compatibility with previous protocols. By definition, the architectural complexity measures the complexity of architecture proportionally to its number of components (in the present case objects/entities) and the interactions (relationships) among these components.

5.1 Information Model for Routing

5.1.1 Information Model: Entity-Relationship Diagram

We represent the different kinds of information manipulated by the routing scheme and how they relate to each other by an entity-relationship diagram.

- The orange boxes represent the entities, which are the various kinds of information

- The green diamonds represent the relationships between the entities

- The ellipses represent the attributes of entities or relationships. Every entity has a key attribute: we may see the information type as a database of some sort, where one of the attributes serves as a key. A piece of information is requested through its key attribute, and the other attributes are then delivered. For instance, given a destination, one may request the corresponding entry of the routing table and get the next hop, etc.

Note: Some relations may themselves have attributes. For instance, the 'Select' relation may be based on several selection criteria: smallest cost, smallest number of hops, or various policies of administrative nature.

- When the relationship between two entities is directed (non symmetric), an arrow indicates the direction on the line between the entity boxes and the relationship diamond.
- The relationships are possibly not just binary. The arrow indicates the directionality, and a number on the arrow indicate the arity: m and n on the extremities mean m-to-n arity for the corresponding information entities based on specified relationship, etc.

Let us consider what occurs, generically, when a routing scheme seeks to locally construct or update the routing table at each node. Note that the nodes in the routing system are distinguished by their IDs. Information exchanged with the rest of the system through Exchanged Information Units (EIUs) gives the node a (partial) view on how the rest of the system is constituted or has been recently modified. These Discovered Information Units fall into two categories: Routing Information or Topological Information. The former is typically composed of pieces of the other nodes' routing tables that contain the necessary information how a message at current node can be routed to others. The latter provides information on how the nodes are connected to each other. Topological information may be of two types: either local (what happens in a neighborhood of the node) or global (which involves transit of information from far away in the network). Note that topological information may contain proper topological attributes (existence or not of links, etc.) but also traffic-related attributes, indicating how traffic is handled on specific links. The messages used to construct and compute the corresponding information could be a combination from several different information units. The Routing Information Base input (RIB in) is the place used to store all routing information which does not specify any particular routing algorithm or protocol. Similarly, Topology Information Base (TIB) stores all topology information without respect to any particular routing protocol. Based on the employed routing protocol, the routing information in RIB, the topology information in TIB and the structured information, a routing path can be produced. Consequently, it also derives the routes which are the transmissions of output from the employed routing protocol. In order to optimize the system performance, a subset of routes (Selected Route) will be selected (e.g., based on some metrics) to generate the routing table entries that are part of the routing table. The raw information collected on the topology and routing tables of the rest of the network is now treated and structured into Structured Information Units (e.g., existence of specific shortest paths, trees, etc.), which are used to compute the entries of the routing table.

Depending on the routing scheme, the raw discovered information units may be mined (i.e., only features are extracted and dominant features selected) or embedded (into a space where the information appears in a more usable way, see for instance, greedy routing on a hyperbolic space), to compose Supplementary Information, helping to compute the routing table entries. The information obtained by the node may also be transferred to other nodes, of course, in order to help them manage their own routing table. Information exchanged with the routing system may also be a packet to be transferred, of course. Then, essentially, the destination is read and the routing table is consulted to find the next hop.

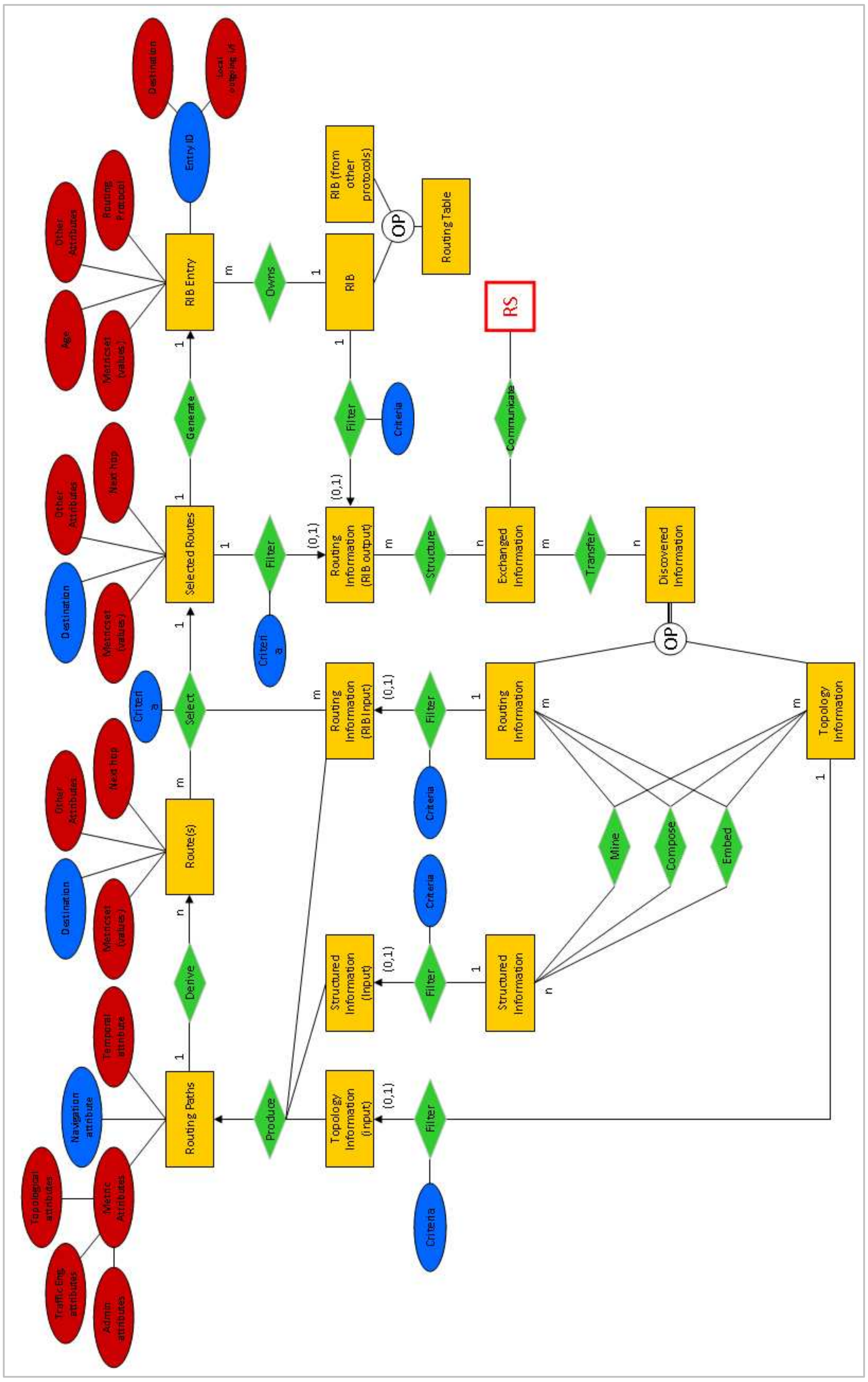


Fig.20: Generic Entity-Relationship Model

After this general description, we now detail more systematically the entities and relationships composing the entity-relationship model.

5.1.1.1 *Entities*

The following entities are comprised in the entity-relationship model:

- **Exchanged Information**: information exchanged and communicated between routers part of the routing system. The exchanged topology information is structured in units, referred to as exchanged information units (EIU).
- **Discovered Information**: a crucial entity of the information model; this information is used to compute and/or select routing paths.
  - **Topology Information**: includes pattern of interconnections of the various network components (e.g., node and links) which can be either physical or logical, i.e., the information related to local and remote interfaces (and their properties), incident links (and their properties), nodes adjacent to incident links (and their properties) as well as non-local environment information including remote links and (abstract) nodes. The discovered topology information is structured in units, referred to as topology information units (TIU). Topology information attributes are detailed in Fig.20.1.
    Moreover, topology information can be further sub-divided into.
    - Local (Neighbor) Information: entity denoting the information about the topology of the network is the immediate neighborhood of the local router.
    - Global (Network) Information: entity denoting the non-local information about the topology of the network for the corresponding router.
  - **Routing Information**: includes non-local distances (or information to derive these distances) and/or paths (segments) together with their attributes. The discovered routing information is structured in units, referred to as routing information units (RIU). Topology information attributes are detailed in Fig.20.2.
- **Topology Information Base (TIB) Entry**: entity comprising a given topology information unit.
- **Topology Information Base (TIB)**: structured entity comprising the collection of the locally stored TIB entries.
- **Routing Information Base (RIB) input**: structured entity comprising all input (discovered or configured) routing information; in case of BGP, this information includes routing paths and associated attributes).
- **Structured Information**: information resulting from the application of a combination of the following operations to the topology information and/or routing information: composition (combination of topology and/or routing information so as to build more complex topology and/or routing information (called structures), embedding/mapping, and mining
- **Routing Path**: the path resulting from local computation (output of the computation algorithm which is in most cases is routing protocol dependent) or filtering. The attributes of this entity are protocol dependent. Generically, we have a key navigation attribute, which contains the information about the direction to follow in order to reach the desired destination(s), a list of metric attributes, which quantify the different possible paths to follow, and an optional temporal attribute. The navigation attribute is typically composed of a vector (e.g. path vector or distance vector) characterizing a set of possible destination, and a data structure composed of nodes (e.g., tree, list, etc.) containing enough information to retrieve a way to reach the destination. The metric attributes can of topological (number of nodes/routers, autonomous system, etc.), traffic engineering (bandwidth, delay, failure probability, etc.) or administrative (cost, color, etc.) nature.
- **Route**: entity resulting from the routing path(s) output of the computation algorithm (route(s) are derived from routing paths) that is used to route the traffic/messages from the source to the destination. Among its attributes, the metric values constituting quantitative criteria on which the selection of particular routes is based. For example this may be the number of hops to destination. The destination attribute may itself have attributes (not shown on the diagram), such as the nature of the destination: is the destination referred to by a locator or an identifier? Depending on the protocol, other attributes may also be added.
- **Selected Route**: a subset of routes resulting from the application of route selection criteria (metric-selection, qualitative-selection, etc.). The attributes are of the same type as Routes.

- **Routing Information Base (RIB) entry**: structured entity derived from the (selected) route; its attribute include the destination, the local outgoing interface identifier together with other optional attributes such as the routing protocol (that generated the entry), the age of the entry, the route attributes and metrics, etc.
- **Routing Information Base (RIB)**: the indexed collection of RIB entries
- **Routing Information Base (RIB) output**: structured entity comprising all locally computed and/or selected routing information (i.e., routing paths).
- **Routing Table Entry**: a structured entry part of the Routing Table that lists the route(s) to particular destination and the metrics associated with those routes. The routing table contains information about the topology of the network immediately around it. The routing table is used to determine that the forwarding or next-hop router according to specific routing protocols. We may find such entries as constructed by the protocol, or entries computed according to other protocols (obtained from the past, or exchanged with the rest of the system, etc.)
- **Routing Table**: the indexed collection of routing table entries.

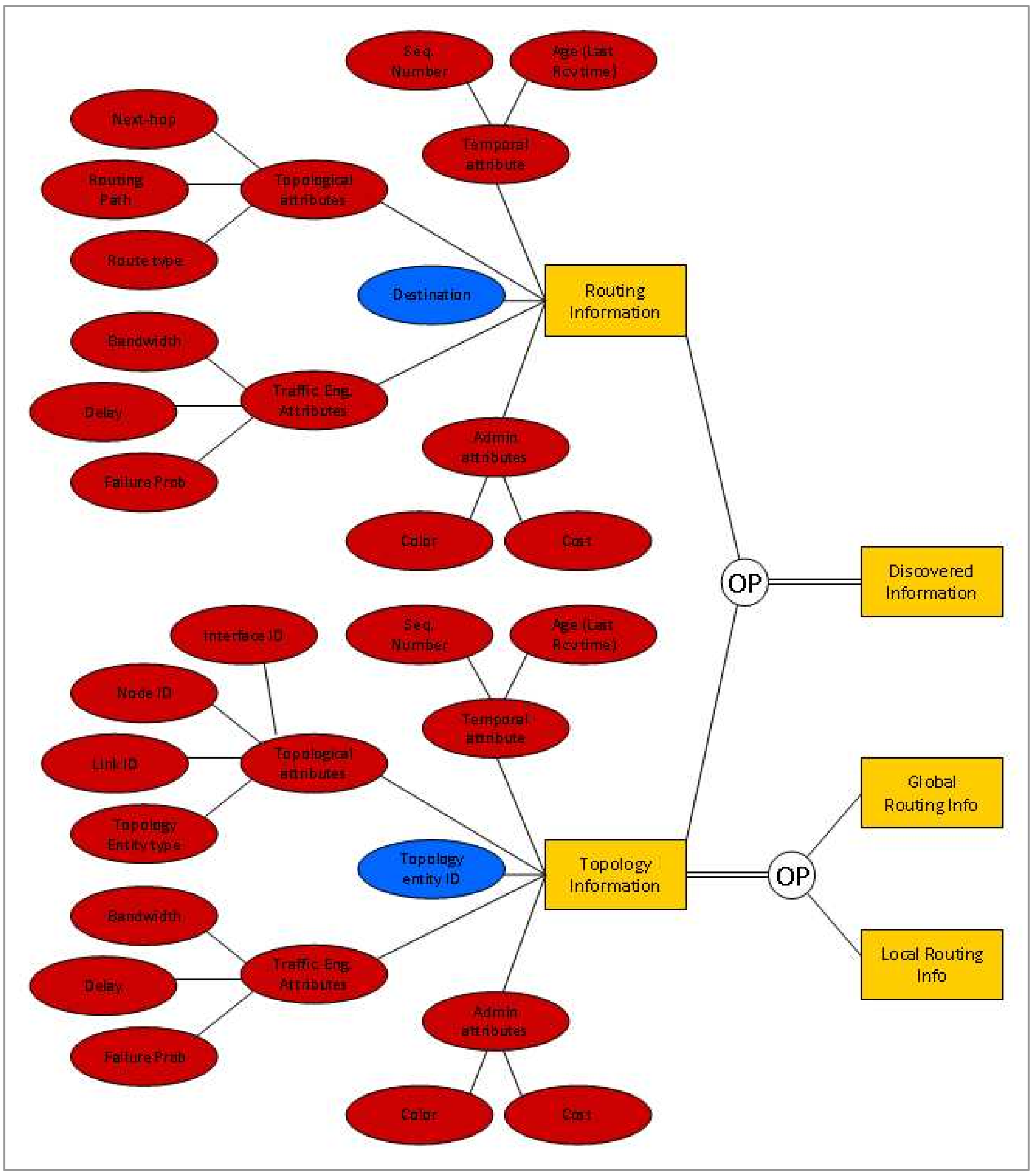


Fig.20.1: Routing and Topology Information Attributes

5.1.1.2 Relationships

The meaning of most relationships is self-explanatory. It should be noted that the meaning of some relationships overlap to some extent. For instance Filter or Select both refer to the fact that the destination of the relationship is a carefully chosen part of the origin of the relationship.

The 'Compose', 'Map/embed' and 'Mine' relationships can be seen as particular cases where the destination of the relationship has an intricate dependency on the origin, showing that the destination has been obtained from the origin after a computation. 'Compose' bundles several pieces of information into one new piece, 'Mine' extracts a partial but particularly relevant information from a mass of raw information (example: extract an optimal spanning tree from a graph), and 'Map/embed' represents an information as another kind of information (example: in greedy routing, we embed the graph into a geometric space, thus nodes become geometric points, cost become lengths, etc.).

5.2 Application of the Generic Information Model

This section provides several examples of application of the generic information model detailed in Section 5.1. For this purpose, we refer to the following routing schemes: BGP path-vector routing protocol, compact unicast routing greedy routing (on greedy embedding of the observed Internet topology and on a constructed Internet topology), and information routing. These examples are provided to demonstrate applicability of the proposed generic information model (and not to delimit a subset of routing schemes and algorithms that will be developed in the context of the EULER project).

5.2.1 BGP Routing Protocol

We consider here the base BGP routing protocol as specified in RFC 4271 and outlined in Section 4.2.1. Using the generic information model, the BGP specific information model is depicted in Fig.21.

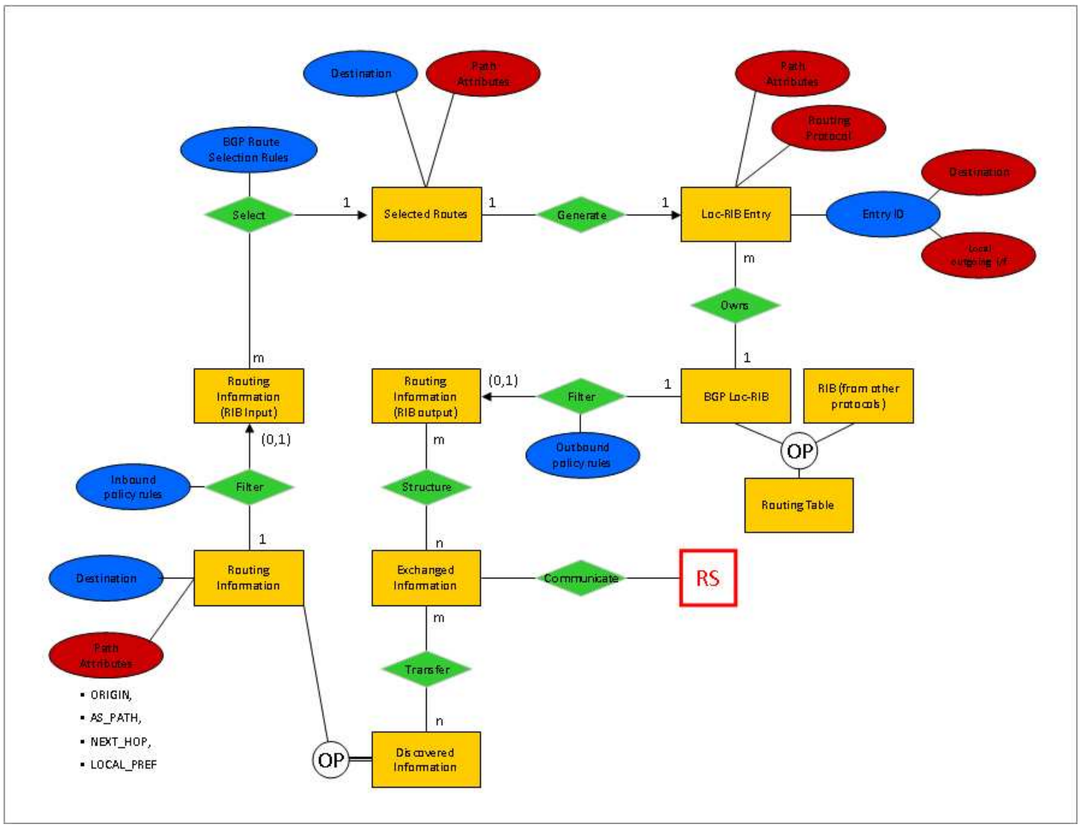


Fig.21: Generic Entity-Relationship Model applied to BGP

5.2.2 Compact Routing

The generic information model depicted in Fig.20, can be used to model the properties, and relationships between the different entities involved in compact routing. The examples provided in this section includes a distributed version AGMNT Compact Routing Scheme

For this purpose, the general information model we develop in this section does not assume centralized computation step (of the routes or other structures required by the routing scheme), and implies distributed manipulation of information. It must be underlined here that the AGMNT compact routing scheme as such is not distributed (but centralized and static). Indeed, this routing scheme provides no distributed way to compute tree-routes as the scheme heavily relies on both local tree routing (in vicinity ball) and global tree routing (landmark trees) but also vicinity balls, etc.. The present section describes what a distributed version of AGMNT would or should look like, leaving unspecified the exact way topology and/routing information is gathered.

5.2.2.1 *Preliminaries*

To clarify the presentation, we introduce key concepts and terminologies from the AGMNT compact routing scheme.

- Vicinity Balls: for any integer k ≥ 1, and for a node u ∈ V, the vicinity of u, denoted by $B_k(u)$ is the set consisting of u and the k closest nodes to u. An essential property of vicinities is that if v ∈ $B_k(u)$ and w is on a minimum cost path from u to v, then v ∈ $B_k(w)$. In other words, the vicinity ball for a node contains a bounded size of the nodes around itself.

- Coloring: the nodes are participated into different color-sets properly such that each color-set has a bounded size and every node has in its vicinity at least one node from each color-set. We also assume a mapping h from node names to colors. Each node u should be able to compute h(t) for any destination node t.

- Landmarks: designate one color to be special and call it the landmark color. A landmark node refers to a node colored by the landmark color.

Note that the vicinity ball for each node has a bounded size and contains at least one landmark node.

5.2.2.2 *Outline of AGMNT Compact Routing*

The AGMNT compact routing scheme consists of two phases: pre-processing phase and routing phase.

i) In the pre-processing phase, each node will compute and store the necessary information on how to route the message to the nodes within its vicinity ball (e.g., minimum spanning trees) and the nodes with exactly same color (e.g., labeled routing schemes) in a minimum cost path. Consequently, the landmark nodes have the knowledge how to route the message between them in a minimum cost path.

ii) In routing phase, the source node will route the message to the destination node based on the information stored in its header, corresponding landmarks and the inter-media nodes on the selected routing path. There are three cases to select the routing path.

The AGMNT compact routing scheme operates as follows:

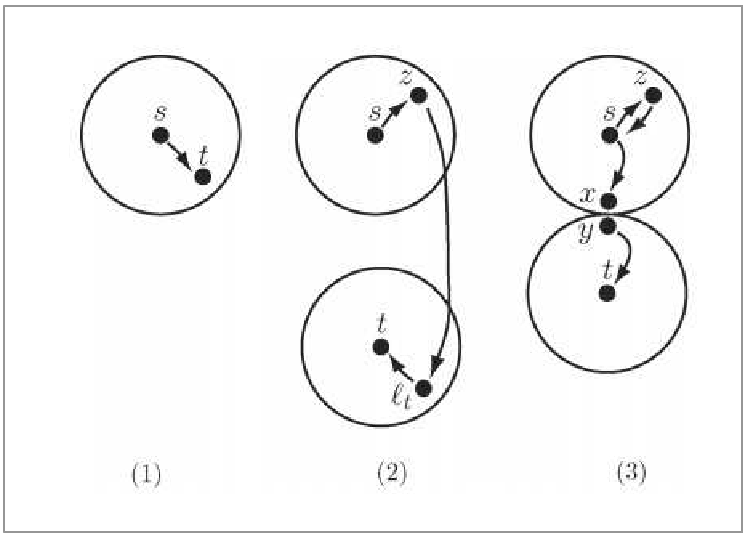


Fig.21: Routing paths in AGMNT compact routing scheme

i) If destination $t \in B(u)$ or $t \in L$ (t is a landmark node) or $c(s) = h(t)$ (informally this means that nodes s and t are assigned the same color), then node s routes to t using its own routing information:

(1) If the destination node t is inside of the source vicinity ball (intra-vicinity ball routing)
Then source s routes on a minimum cost path directly to the destination node t.

(2) If the distance of the vicinity balls between source and destination nodes is far apart (inter-vicinity ball routing), i.e., On every minimum cost path from s to t there is a node y such that $y \notin B(s)$ and $y \notin B(t)$.
Then the source s routes along a minimum cost path the message to the landmark node l_t included in the vicinity ball of the destination t. Then, the message will be delivered on a minimum cost path directly to t from l_t. ($s \rightarrow z \rightarrow l_t \rightarrow t$). Note that source node s can directly route the message to destination node t along a minimum cost path based on its own information if both nodes s and t are in the exactly same color-set.

(3) If source and destination nodes vicinities are close but their vicinity balls do not intersect but are contiguous (inter-vicinity ball routing between contiguous balls)
Then the source s routes the message to a node x along a minimum cost path such that x is within node s vicinity ball and there exists a node y such that y belongs to the destination t vicinity ball and node x can directly communicate with node y, e.g., $(x,y) \in E$. Then the message is forwarded to node y. Finally, the message is delivered on a minimum cost path from node y to t ($s \rightarrow x \rightarrow y \rightarrow t$).

ii) Otherwise, node u forwards the packet to node $w \in B(u)$ such that $c(w) = h(t)$. Then from node w the packet goes to node v using node w’s routing information.

5.2.2.3 Information model for AGMNT compact routing

Fig.22 illustrates the E-R information model for the AGMNT compact routing scheme. The entities comprised in this model are the following:

- Exchanged Information: entity denoting the units of information that are communicated between nodes.

- Discovered Information: comprises both topology and routing information
  - Topology Information:
    - Local (Neighbor) Information: set of neighbors in node's u vicinity ball B(u)
    - Global (Network) Information: set of nodes in node's u vicinity ball B(u), the nodes with the same color as node u, and landmark nodes
  - Routing Information Unit: shortest path to corresponding nodes (e.g., all nodes in B(u) and landmark nodes)

- Structured Information (also referred to as supplemental information): resulting from the application of mining and composition
  - Mine: color of each node (and all nodes with the same color as node u), but also doubling dimension and other network topology properties
  - Compose: vicinity ball B(u)

- Routing Table Entry: each node u stores the following routing information
  - $\forall v \in B(u)$, minimum cost routing path to the destination node v (this implies storing at node u, $\forall w \in B(u)$, the name w and the port name $u \rightarrow y$, where node y is the next hop on a minimum cost path from u to w)
  - $\forall l_v \in L$ (t is a landmark node): routing path from u to $l_v \in B(v)$ along the tree $T(l_v)$ and from l_v to v along the same tree $T(l_v)$: $s \rightarrow l_v \rightarrow v$.
  - $\forall v \notin B(u)$, $\forall u \in B(w)$, and there exists an edge (x, y) along the minimum cost path from w to v such that $x \in B(w)$ and $y \in B(v)$, the lowest cost path $u \rightarrow w \rightarrow x \rightarrow y \rightarrow v$.

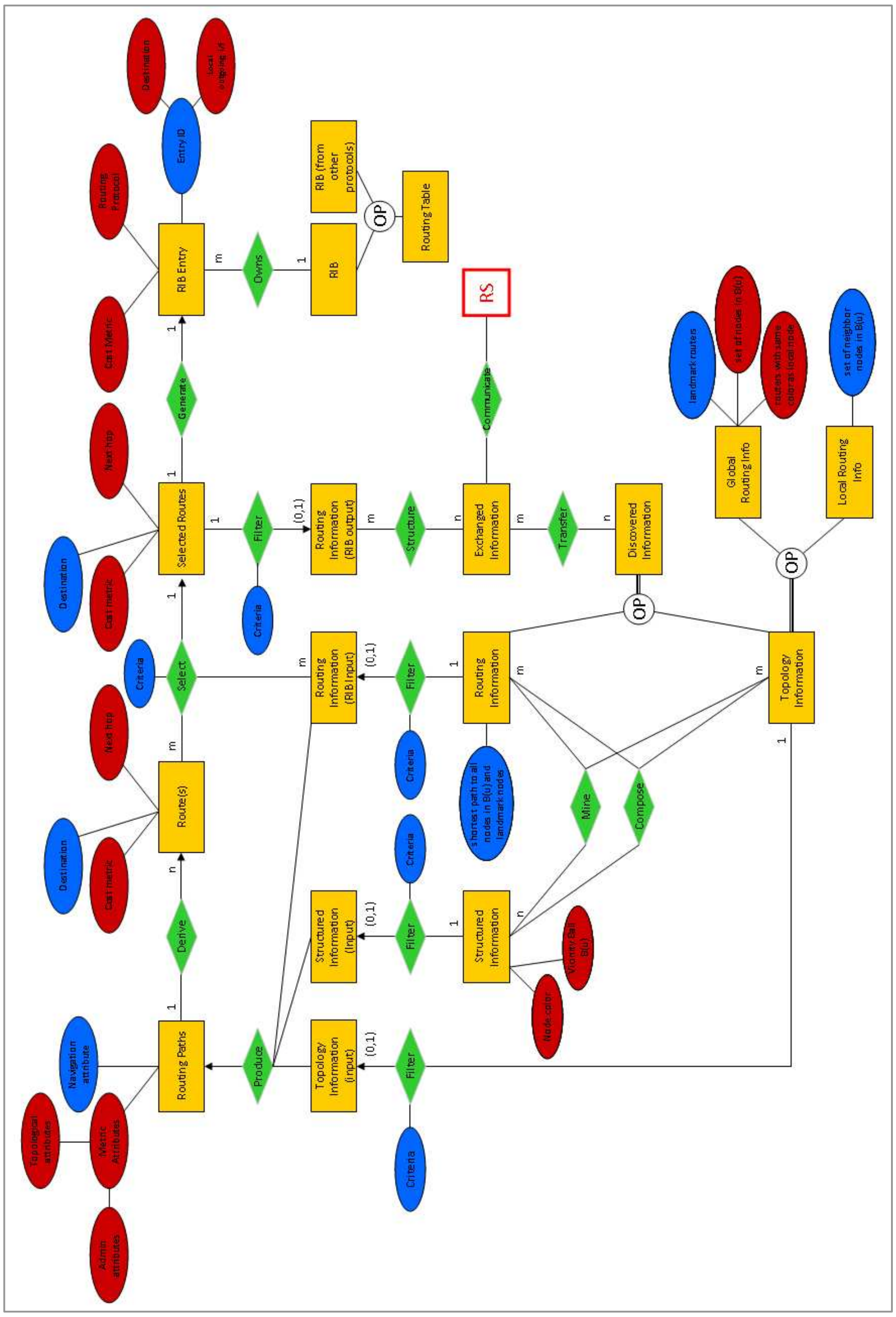


Fig.22: E-R Information model for distributed version of AGMNT compact routing

5.2.2.4 Distributed AGMNT Compact Routing Scheme

One of potential solutions to build the distributed fashion in the traditional AGMNT compact routing scheme is to combine the pre-processing stage for information collection and routing stage together. The distributed information collection can be performed by the distributed gossiping (ALL-to-ALL) approaches described in the literature. Consequently, the routing strategy will be made according to the collected information at each node. However, the complexity on information exchange in the gossiping schemes may not be affordable in practice. Note also that the global information will be necessary if we really want to guarantee a proper performance for the routing scheme in terms of the stretch in the worse case scenario. Whether the AGMNT compact routing scheme can be made purely distributed in a more efficient way (e.g., combining with a time-efficient randomized gossiping scheme) is an important question for the future of this scheme.

Let us now comment on complexity measures for distributed routing schemes. The information model provides a natural complexity measure of a distributed routing scheme: the amount of information (number of bits or messages) exchanged in the system in order to converge to the construction of a routing table in all nodes. This communication complexity complements the usual trade-off stretch/size of the tables.

5.2.3 Geometric Routing on Greedy Embedding

We consider the particular case of geometric routing on greedy embedding of the Internet topology in the hyperbolic plane, following the seminal approach proposed by Kleinberg in 2007 [Kleinberg07]. Information model for other geometric routing schemes will be described in the next release of this document.

In this context, the information model for this particular greedy routing scheme is centered on the deduction of coordinates information for network nodes. All other present information serves this result. The information propagation is started with the event of network nodes detecting neighboring nodes (adjacencies). Two types of information processes can be initiated using this information: i) a direct determination of the spanning tree information, or ii) an intermediate step of topology information distribution and subsequently a SPT deduction. In the latter case, the resulting adjacency information is captured in topological information messages (e.g. link-state messages), and is distributed over the network. Every node stores the resulting topology information received from other nodes. The latter (e.g., a database of link entries) forms the basis for deducing a spanning tree (SPT) of the resulting graph topology.

The SPT is actually a subset of the initial topological information, as it just marks for every original link if it is disabled or not. The SPT information is used to derive some properties such as the depth, degree, and root of the tree. This information is used as input for information characterizing structure of the resulting coordinate system. The Kleinberg greedy embedding encodes this structure in terms of two Mobius transformations. The Mobius transformations are used to determine the network coordinates themselves in a distributed way. The resulting coordinates are stored locally and distributed towards the neighbors of network nodes. Neighboring nodes store the received coordinates in their routing table.

5.2.4 Information routing systems

The aim of this section is to provide models for relating the application level (such as the one described in peer-to-peer systems) to the induced traffic. This work is directly in relation with WP3 T3.2 (subtask 4 dedicated to the modeling of diffusion phenomena) and WP4 T4.2 (generation of realistic diffusion traces).

Our model is composed of two layers. At the upper layer, peers are identified and have two kinds of dynamic attributes: a list of files that they may provide and a list of peers that they may contact directly (called neighbors). This relation between peers is called the overlay network. But peers are also nodes (end hosts) of the physical topology, connected together through the physical internet (wires and routers). Communication between them including control traffic, query routing and file exchanges, relies therefore on an underlying routing protocol (defined by other tasks in the project). Our model makes it possible to describe different possible strategies at peer-to-peer level (for finding files, for routing answers, etc) and their relation with the underlying routing. Depending of the goals (for instance, preservation of user privacy or efficiency in terms of induced traffic), one may use different variants. Our model provides a simple and rigorous framework for describing such protocols.

Fig. 23 describes the main features of a typical peer, which faces several possible situations:

- the peer may receive a query (with its attributes) from one of its neighbors. Then, the peer searches for the keywords of the query in the file names in its file list. If a matching file is found, it sends an answer to the query, composed of the list of providers it knows for this file (in most protocols, only itself). In addition, it may forward the query if its TTL is larger than 0; in such cases, it decreases the TTL and simply forward the query to (some of) its neighbors (depending on the protocol).

- the peer may receive an answer to a query. If the query was issued by this peer, then the processing of this query is finished. The peer may then download the file from the provider(s). In a classical system, the providers directly own the files; in an anonymous system like freenet, the providers are neighbors of the peers, which themselves download the file from another provider if they did not have it previously.
- the peer may send a query (not forward it as above) for a file it is interested in. This query is composed mainly of a set of keywords to search for and an initial TTL which defines the maximal number of hops for the query in the overlay. The query is generally sent to all neighbors, but more subtle protocols may be defined.

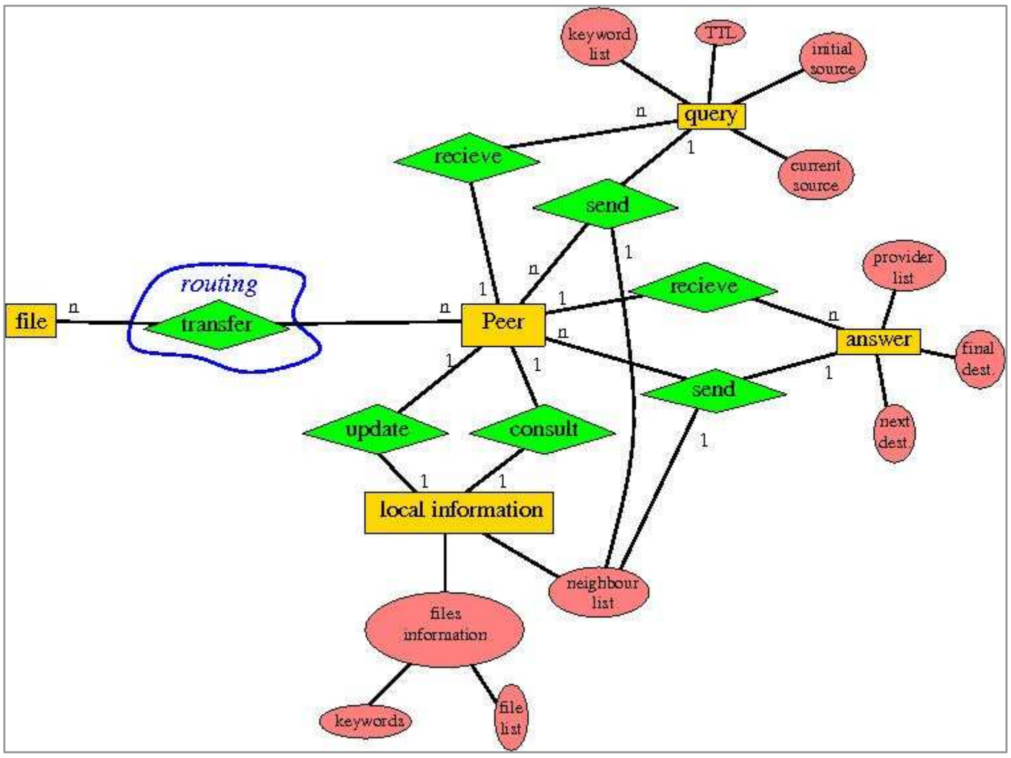


Fig.23: Information model for Information routing system

This global scheme makes it possible to define a wide variety of protocols. For instance, the peer may update its local information (both the list of files it knows of and its neighbor list) when it gets new information about this (for instance when it forwards an answer to a query); the peer may choose forwarding strategies in different ways (flooding or more selective choices, for instance). Also, each peer may identify itself as the source of any query when forwarding it, thus hiding the original sender of the query and leading to anonymous networks like freenet.

In the context of the EULER project, the purpose is to determine the impact of such exchanges on the routing and load of the network. Therefore, we consider control traffic (routing of queries and answers) as negligible; instead, file transfers have significant impact on routing. Such transfers may occur directly between an owner of a file and a peer downloading it, but it may also follow links in the overlay (which induces much heavier traffic). Our model captures such variants and will be able to generate realistic traffic from this regard. The following picture describes how the former scheme may be used to induce traffic on real networks.

In Fig.23, the overlay network is represented in green on the right. On the left is the real underlying network on which the communication takes place. This figure explicates how the nodes in the overlay network coordinates in order the retrieve the information of which peers are exchanging information. Once this is done the actual communication occurs and this triggers traffic on the underlying network. This is done by mapping each green node to a real end-host black node of the real network. One can see in particular on Fig.24 that the interaction between two neighbors in the overlay network may in turn triggers complex traffic demands on the real underlying network.

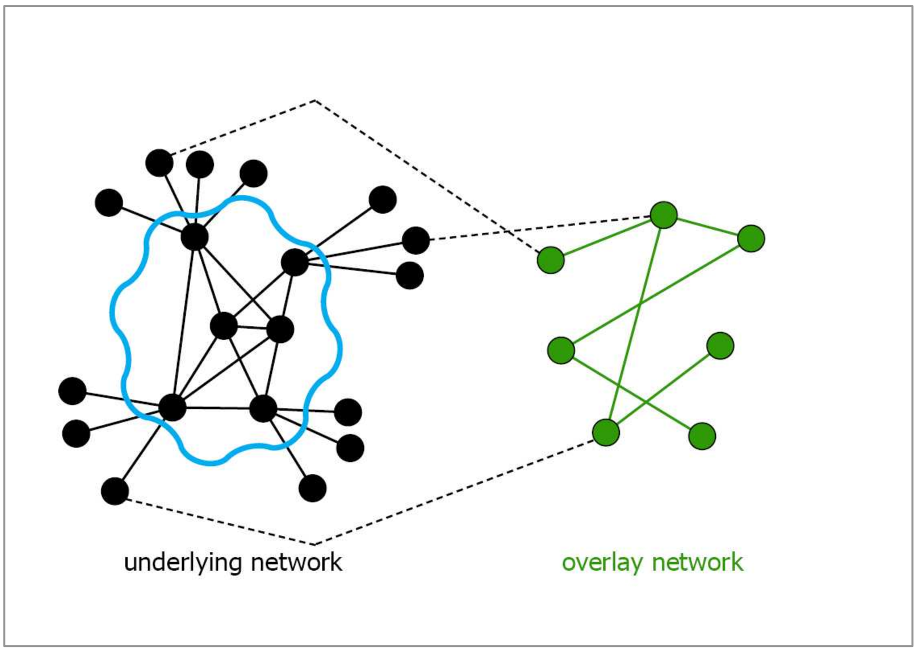


Fig.24: Interaction between the overlay and the underlying network

6. Conclusion

In order to provide the architectural baseline of the Internet routing system, this document proposes to follow a systematic modeling approach relying on the specification of a functional and an information generic model. Indeed, past experience in designing routing protocols shows that without well defined system architecture, extending, adding or removing routing functionality leads to further complexity without actually meeting scalability, adaptivity and performance objectives (see IP control plane design today). More generally, finding the suitable tradeoff between both short- and long-term adaptivity and performance is critical to ensure longevity of the routing system architecture. In this context, by starting from a top-level view down to the design of the routing scheme (and their components), we initiate our design effort from a holistic approach that is complementary to experimental/bottom-up approach.

The proposed generic functional and information models provide a structuring and increase cohesion between the different components that the specification of a routing scheme requires; this, by taking into account the challenges of adaptivity (resulting from dynamicity of the topology) and distribution (resulting from the need to scale to large topologies partitioned into different units of operation). Through several representative routing schemes, we show the applicability of these models and that they can effectively cover all routing schemes considered in the context of the EULER project including, compact routing, geometric/greedy routing, etc. It is indeed interesting to observe that the first levels of the functional specification do require limited per-routing scheme customization. More precisely, the specialization/customization mostly occurs relates to the exchange mode of routing and topology information (and associated control), the pre-processing of exchanged/discovered information and obviously the routing path computation algorithm. Further, and corroborating this observation, distributed and dynamic routing schemes shows the importance of the discovery function (and related information exchanges) which itself leads to reconsider for the some of the methods and objects/data structures these routing schemes use to build the (topological) structures on which routing path computation relies. At this point, the rationale and the purpose of the information model become clearer: the specification of information/entities adapted for distributed computation and adaptive/dynamic capability of the routing scheme. Note also that in distributed systems like the Internet routing, the choice/decision is locally performed by each (abstract) node independently of the others using the exchanged information (i.e., discovered information) but individual nodes choice/decision affects other agent's choice/decision. It is therefore fundamental to capture these interactions as part of our functional model up to the level appropriate for further routing systems engineering (establish routing system engineering controls over the allocations and interfaces).

Finally, we have also identified open routing research areas by determining which routing (sub-)functions are currently under-specified or mis-specified but also which routing (sub-)functions can be replaced, added or even removed from their specification as documented in existing scientific literature. The comparison between the different routing schemes that will be designed in context of Task 2.2, models (and associated operations) will also be facilitated as functional modeling offers at the same time a detailed functional analysis grid. In order to reach our goal of routing model specification, the present work will be further progressed by the specification of a procedural model (formal description of procedures) and a data model (formal description of data structures and their relationships together with data operators applied to these structures).

Annex 1: FFBD and EFFBD diagramic modeling techniques

1. Introduction

The FFBD (Functional Flow Block Diagrams) and EFFBD (Enhanced FFBD) are diagramming techniques that can be used for modeling the functional behavior of a system. A complete functional model must depict both the “control” and “data” aspects of the system. The simple FFBD technique models the control (or the logical) environment of a system, while the EFFBD technique models both the data and control environment of a system. Note that in this document we bundle the basic FFBD technique together with extensions referred in the literature as "extended FFBD".

2. Functional Flow Block Diagrams (FFBD)

The Function Flow Block Diagram (FFBD) was the first to be favored by systems engineers and continues to be widely used today (DSMC 1989, Blanchard and Fabrycky 1990). Figure 1 shows a sample FFBD. The Functional Flow Block Diagram (FFBD) provides a hierarchical decomposition (multi-tier) of the system's function with a control structure that dictates the order in which the (sub-) function can be executed at each level of the decomposition. The FFBD presents thus the logical sequencing of the same (sub-)functions as those identified through functional decomposition by displaying them in their logical, sequential relationship. The decomposition of the function into the sequence of activities is carried out by asking the question "WHAT" needs to be done to perform the particular function (but does not assume a particular on how the function shall be performed).

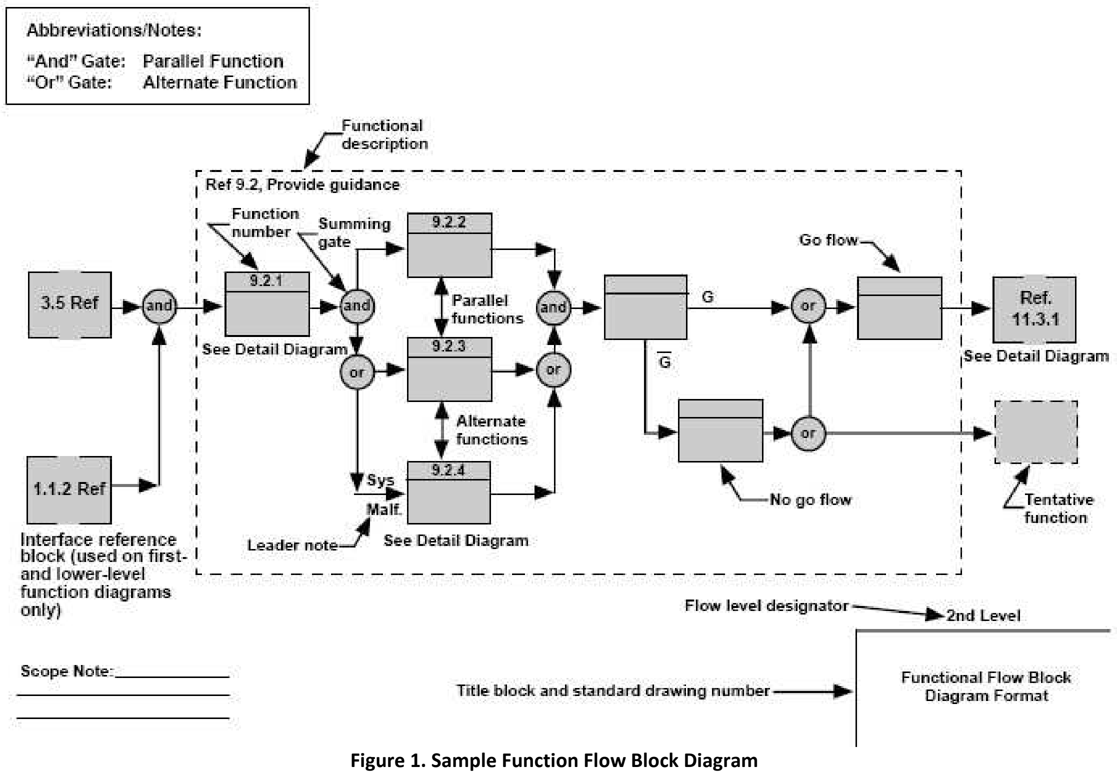


Figure 1. Sample Function Flow Block Diagram

FFBD is a commonly used tool in functional modeling to define the functional level and the sequences of activities. The purpose of the FFBD is to show the (sub-)functions that a system is to perform, the order in which they are to be performed (logical sequence), and their relationships. The order of execution is specified from the set of available control constructs. The basic FFBD syntax includes four types of control structure: series, concurrent/parallel (AND), selection (OR), and multi-exit function together with the completion criterion (IF-THEN-ELSE). Note that in order to include iteration (IT symbol), loop (LP symbol), and replication (RP symbol), the extended version of FFDB shall be used because basic FFDB syntax only allows to depict 4 types of control structures being series, concurrency, selection and multi-exit). The FFBD does not contain any semantics relating to the flow of information between functions, and therefore does not distinguish between triggering and non-triggering inputs.

3. Logic symbols (Constructors) in FFBD

3.1. Contextual and Administrative Data

Each FFBD shall contain the following contextual and administrative data:

- Date the diagram was created.
- Name of the engineer, organization, or working group that created the diagram.
- Unique decimal delimited number of the function being diagrammed.
- Unique function name of the function being diagrammed.

Figures 2 and 3 present the administrative data in an FFBD. Figure 3 is a decomposition of the function F2 contained in Figure 3 and illustrates the context between functions at different levels of the model.

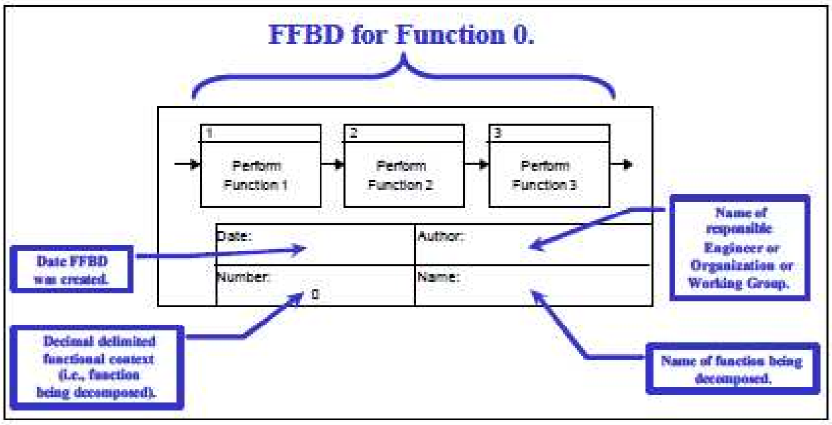

FFBD for Function 0.
1
Perform Function 1
2
Perform Function 2
3
Perform Function 3
Date:
Author:
Number:
0
Name:
Date FFBD was created.
Decimal delimited functional context (i.e., function being decomposed).
Name of responsible Engineer or Organization or Working Group.
Name of function being decomposed.

Figure 2. FFBD Function 0 Illustration

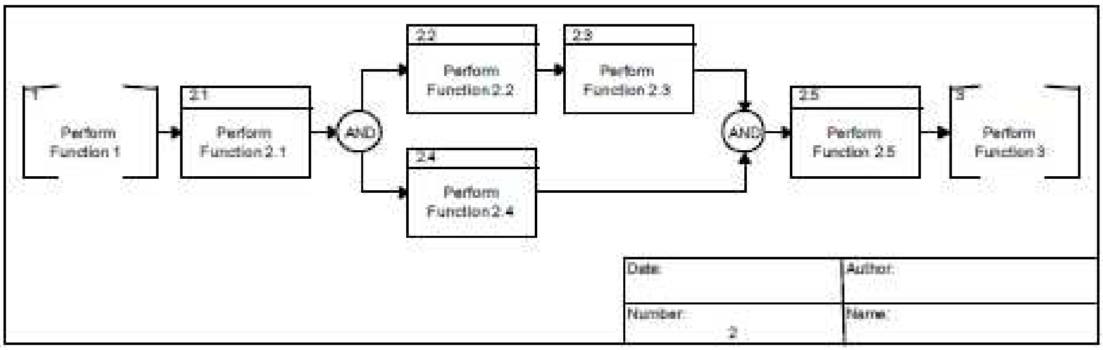


Figure 3. FFBD Function 2 Illustration

3.2. Function

A function shall be represented by a rectangle containing the name of the function (an action verb followed by a noun phrase) and its unique decimal delimited number. A horizontal line shall separate this number and the title, as shown in see Figure 4 above (the figure also depicts how to represent the so-called reference function, which is a type of function that provides context within a specific FFBD).

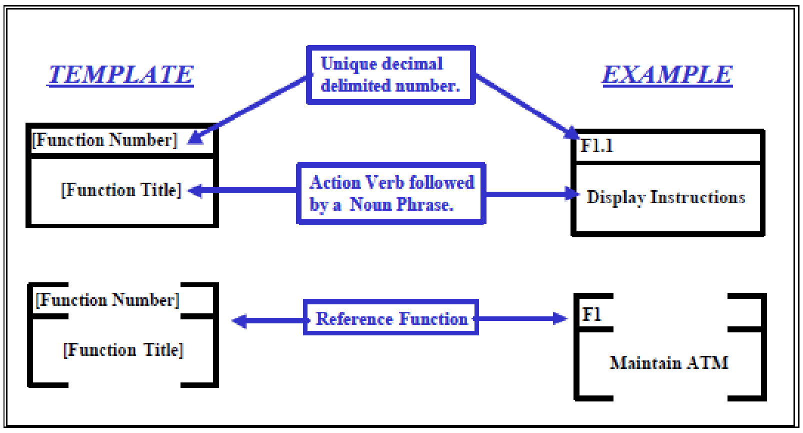


Figure 4. Function Symbol

3.3. Directed Lines

A line with a single arrowhead shall depict functional flow from left to right.

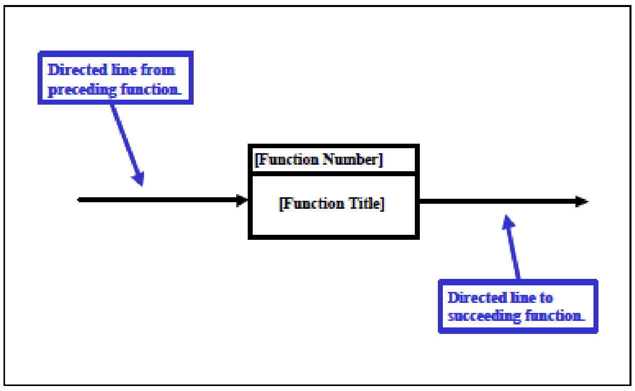

Directed line from preceding function
[Function Number]
[Function Title]
Directed line to succeeding function

Figure 5. Directed Lines

3.4. AND

The "AND" constructor is a condition in which all preceding or succeeding paths are required: it may contain one input with multiple outputs (i.e., the input enables in parallel all the outputs) or multiple inputs with one output (i.e., the output is enabled when all inputs enable it). It cannot contain multiple inputs and outputs combined. See the example in Figure 6: F2 and F3 begin in parallel after completion of F1, and F4 begins after completion of F2 and F3.

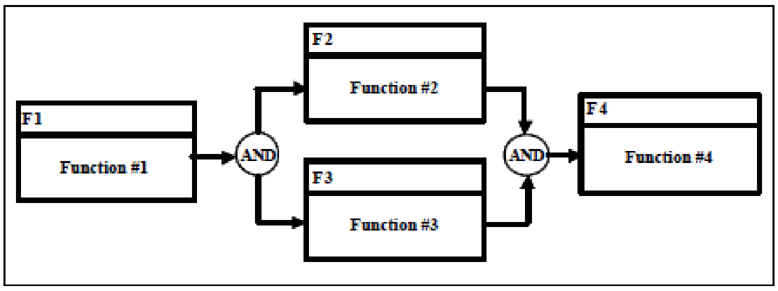


Figure 6. AND Symbol

3.5. Exclusive OR

The "Exclusive OR" constructor is a condition in which one of multiple preceding or succeeding paths is required, but not all: it may contain one input with multiple outputs (i.e., the input enables only a single output) or multiple inputs with one output (i.e., the output is enabled when one of the inputs enable it). It cannot contain multiple inputs and outputs combined. See the example in Figure 7: F2 or F3 may begin after completion of F1, and F4 may begin after completion of either F2 or F3.

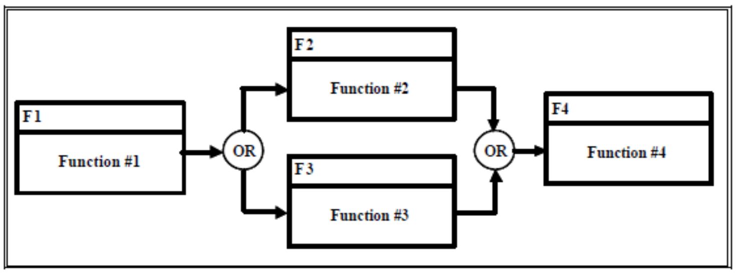


Figure 7. "Exclusive OR" Symbol

3.6. Inclusive OR

The Inclusive OR constructor is a condition in which one, some, or all of the multiple preceding or succeeding paths are required. Figure 9 depicts inclusive OR logic using a combination of the AND symbol (Figure 6) and the Exclusive OR symbol (Figure 7). Read Figure 8 as follows: F2 or F3 (exclusively) may begin after completion of F1, or (again exclusive) F2 AND F3 may begin after completion of F1. Likewise, F4 may begin after completion of either F2 or F3 (exclusively), or (again exclusive) F4 may begin after completion of both F2 and F3.

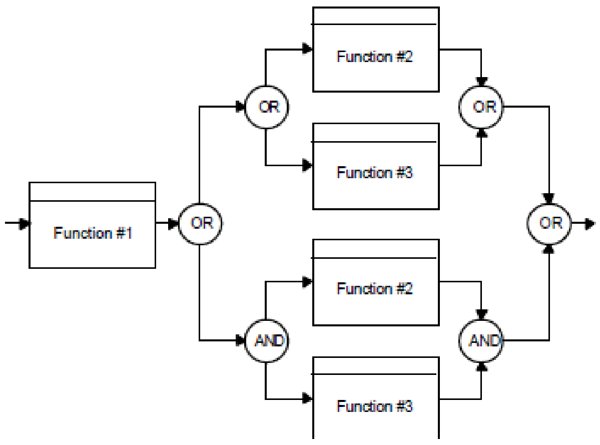


Figure 8. "Inclusive OR" Symbol

3.7. Multiple Exit

Commonly only a single output line is shown for a function on a flow diagram, but enhanced flow diagramming recognizes a multiple output constructor. A multi-exit constructor indicates that the function may enable multiple (one, two, etc.) functions according to some exit conditions. In Figure 9, whichever output is enabled in function A, the following function B or C is accomplished.

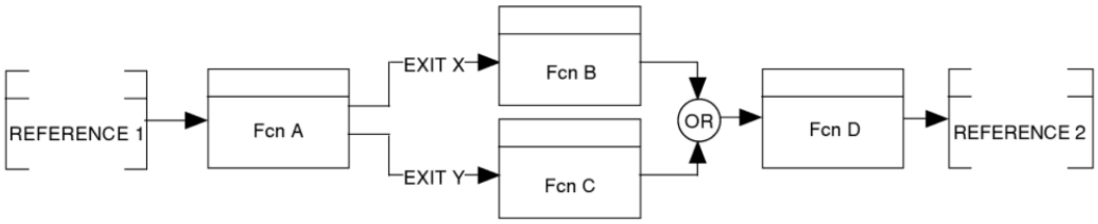


Figure 9. Multiple output

3.8. Iteration

Iteration allows a repeat of a specific function or "thread" of function over a given "domain set", where a domain set represents i) a certain number of times, ii) a rate, or iii) a set definition. A "thread" of functions can be anything from one function within the iteration to a complicated set of functions controlled with multiple control constructors. In Figure 10 the execution of Reference 1 enable function A, function A is executed, and control is returned to the first iteration node which then enables the function A again, and this is done as directed by the domain set: if the domain set is 3, then function A is executed three times. When the iteration has been completed, Reference 2 is enabled.

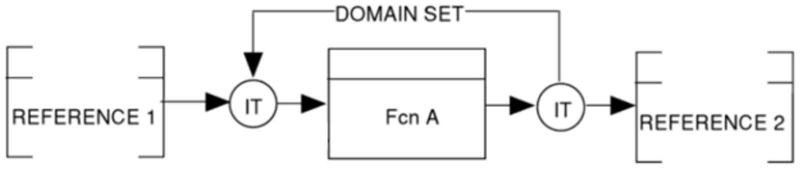


Figure 10. Iteration Symbol

3.9. Loop

The loop construct permits a repeat of a function or "thread" of functions until a specific loop exit condition is achieved. In Figure 11, Reference 1 is executed and enables function A, function A is enabled and executed. If loop exit condition is achieved, function B is enabled. If the loop exit condition is not satisfied, control is transferred back to the first LP node and function A is enabled again. The loop will continue to transfer control and enable function A indefinitely until the loop exit condition is achieved.

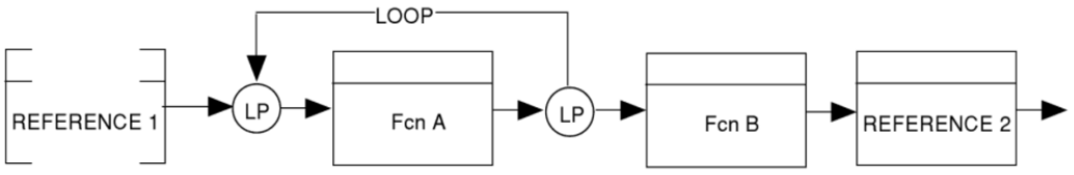


Figure 11. Loop Symbol

4. Enhanced FFBD (EFFBD)

4.1. The data flow

A key concept in modeling functional flow is that for a function to begin, the preceding function or functions within the "control" flow must have finished. For example, an "eat food" function logically would not begin until a "cook food" function was completed. The logical sequence of functions (i.e., the functional flow) describes the "control" environment of the functional model. However, in addition to a function being enabled, it may also need to be triggered with an input. So, in the example, the "eat food" function is enabled once the "cook food" function is completed, and once it receives the "prepared food" as input. This second aspect—triggering a function—speaks to the "data" environment, which the Enhanced FFBD (EFFBD) diagram captures.

Enhanced FFBD (EFFBD) displays the control dimension of the functional model in an FFBD format (using more logical symbols) with a data flow overlay to effectively capture data dependencies. Thus, the Enhanced FFBD represents: (1) functions, (2) control flows, and (3) data flows. The logic constructs allow you to indicate the control structure and sequencing relationships of all functions accomplished by the system being analyzed and specified.

4.2. Data Triggering

When displaying the data flow as an overlay on the control flow, the EFFBD graphically distinguishes between triggering and non-triggering data inputs. Triggering data is required before a function can begin execution. Therefore, triggers are actually data items with control implications. In Figure 12, data triggers are shown with green backgrounds and with the double-headed arrows. Non-triggering data inputs are shown with gray backgrounds and with single-headed arrows.

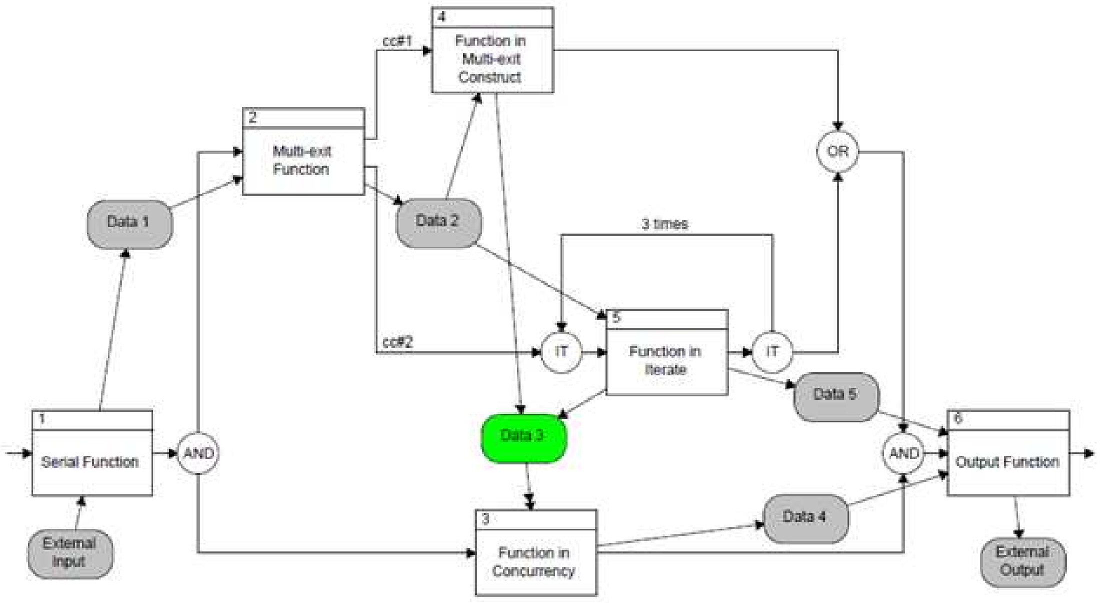


Figure 12. Sample Enhanced Function Flow Block Diagram

In EFFBD a function begins execution if it is both enabled by control and triggered by data (if no data trigger is specified, it begins execution enabled by control). A function is enabled by control when the function(s) that precede it in the control flow have completed their execution. A function is triggered by data when the required stimulus data item becomes available to the function. We are not concerned here with other execution requirements (such as the availability of necessary resources).

Using EFFBD one has the freedom to use either control constructs or data triggers (or a combination of both) to specify execution conditions for individual system functions. Note that the trigger constructor offers a more complicate relationship. In Figure 13, function A or B will be enabled only if the prior function has been completed and the trigger is applied. In case of the inclusive OR arrangement depicted, i) both functions may execute if both are triggered, or ii) one or the other if only one is triggered, or iii) neither of them if none is triggered.

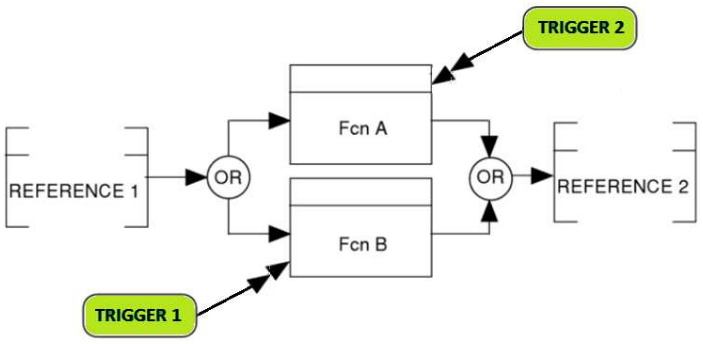
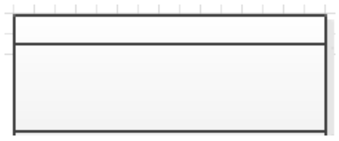
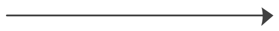
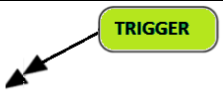
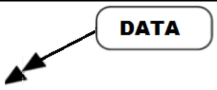
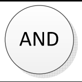
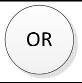
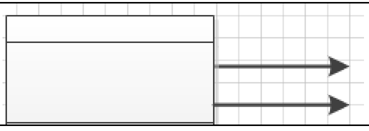
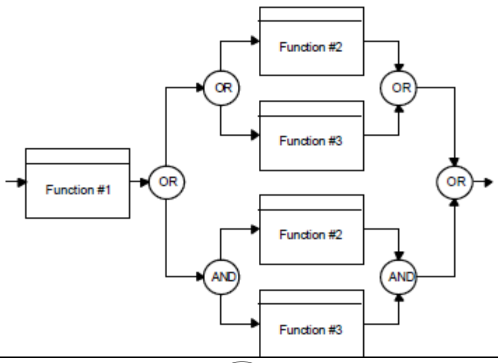
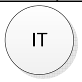
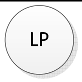


Figure 13. Trigger Constructor

Logic Symbols Summary

The following table summarizes the main symbols used in FFBD and EFFBD.

Name	Symbols	Used in FFBD	Used in EFFBD
Function		X	X
Control (enabling) line		X	X
Data Triggering Line			X
Data (Non-Triggering) Line			X
AND		X	X
Exclusive OR		X	X
Multi-Exit Constructor		X	X
Inclusive OR	 Function #1 OR OR Function #2 Function #3 OR AND Function #2 Function #3 AND OR	X	X
Iteration		X	X
Loop		X	X

References

[Abraham06d] I.Abraham, C.Gavoille, and D.Malkhi, On space-stretch trade-offs: upper bounds, In Proceedings of the 18th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'06), pp.217-224, Cambridge, USA, July 2006.

[Abraham08] I.Abraham, C.Gavoille, D.Malkhi, N.Nisan, and M.Thorup, Compact name-independent routing with minimum stretch, ACM Transactions on Algorithms (TALG), vol.4, no.3, art.37, June 2008.

[Abraham09] I.Abraham, D.Malkhi, and D.Ratajczak, “Compact multicast routing,” Proc. 23rd Int. Symp. DISC'09, Elche, Spain, pp.364–378, Sep.2009.

[Awerbuch08] B.Awerbuch, Y.Azar, A.Epstein, V.Mirrokni and A.Skopalik, Fast convergence to nearly optimal solutions in potential games, In Proceedings of the 9th ACM Conference on Electronic Commerce (EC ’09), pp.264-273, 2008.

[Barabasi99] A-L. Barabasi, and R. Albert, Emergence of scaling in random networks, Science 286, 509, 1999.

[BGPReport] http://bgp.potaroo.net/index-bgp.html

[Boguna10] M.Boguna, F.Papadopoulos, and D.Krioukov, Sustaining the Internet with Hyperbolic Mapping, Nature Communications, art.1, p.62, 2010

[Booch99] Presentation at the Software Developers Conference in 1999

[Buede00] D.Buede, Engineering Design of Systems - Models and Methods, John Wiley & Sons, 2000.

[Chen76] P.Chen, "The Entity-Relationship Model--Toward a Unified View of Data". In: ACM Transactions on Database Systems 1/1/1976 ACM-Press ISSN 0362-5915, S. pp.9–36

[Elmasri04] Elmasri, Ramez, B. Shamkant, Navathe, Fundamentals of Database Systems, Fourth ed., Addison-Wesley, Menlo Park, CA, USA, 2004.

[FAA07] Federal Aviation Administration, “System Engineering Manual”, 2007 (available in http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/operations/sysengsaf/seman/).

[Fenner06] B.Fenner, et.al., “Protocol Independent Multicast - Sparse Mode (PIMSM),” Internet Engineering Task Force (IETF), RFC 4601, Aug.2006.

[Garlan95] Garlan and D.E.Perry, Editorial to the IEEE Transactions on Software Engineering, April 1995.

[Grady06] J.O.Grady, “System Requirements Analysis”, Ed. By Academic Press, ISBN 012088514X, 2006.

[Kleinberg05] R.Kleinberg, J.Kleinberg, Isomorphism and Embedding Problems for Infinite Limits of Scale-Free Graphs, In Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA’05), pp. 277-286, Vancouver, Canada, Jan. 2005.

[Kleinberg07] R.Kleinberg, Geometric routing using hyperbolic space, In Proceedings of the 26th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM’07), pp.1902-1909, Anchorage, Alaska, May 2007.

[Korman06] A.Korman, and D.Peleg, Dynamic routing schemes for general graphs, In Proceedings of the 33rd International Colloquium on Automata, Languages, and Programming (ICALP’06), Part I, LNCS 4051, Springer, pp.619-630, Venice, Italy, July 2006.

[Krioukov04] D.Krioukov, K.R.Fall, and X.Yang: Compact Routing on Internet-like Graphs, In Proceedings of the 23th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’04), Hong Kong, China, March 2004.

[Krioukov07] D.Krioukov, K.C.Claffy, K.R.Fall, and A.Brady, On Compact Routing for the Internet, ACM SIGCOMM Computer Communication Review (CCR), Vol. 37(3), 2007.

[Krioukov08] D.Krioukov, F.Papadopoulos, M.Boguna, and A.Vahdat, Efficient Navigation in Scale-Free Networks Embedded in Hyperbolic Metric Spaces, Caida, May 2008.

[Labovitz99] C.Labovitz, R.Malan, and F.Jahanian, Origins of Internet Routing Instability, In Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’99), pp.218-226, New York, USA, March 1999.

[Labovitz00] C.Labovitz, A.Ahuja, A.Bose, and F.Jahanian, Delayed internet routing convergence. In Proceedings of the ACM SIGCOMM’00 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pp.175-187, Stockholm, Sweden, Sept. 2000.

[Labovitz01a] C. Labovitz, A. Ahuja, A.Bose, and F.Jahanian, Delayed Internet Routing Convergence, IEEE/ACM Transactions on Networking, Vol.9(3):293-306, 2001.

[Labovitz01b] C.Labovitz, R.Wattenhofer, S.Venkatachary, and A.Ahuja, The impact of Internet policy and topology on delayed routing convergence, In Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01), pp.537-546, Anchorage, Alaska, April 2001.

[Long02] J.Long, Vitech Corporation, "Relationships between Common Graphical Representations in System Engineering", Proceedings for the NCOSE Symposium, 2002.

[McInnes11] A.McInnes, B.Eames, R.Grover, "Formalizing Functional Flow Block Diagrams Using Process Algebra and Metamodels", IEEE Transactions On Systems, Man, And Cybernetics—Part A: Systems And Humans, Vol. 41, No. 1, January 2011.

[Papadimitriou05]C. H. Papadimitriou and D. Ratajczak, On a conjecture related to geometric routing, Theoretical Computer Science, Vol.344(1), pp.3-14, 2005.

[Papadimitriou10]D.Papadimitriou, Internet Routing - New paradigms and Schemes, SISCom-Bretagne Research School on Networks and Telecommunications, Rennes, France, June 2010.

[Pedroso11] P.Pedroso, D.Papadimitriou, D.Careglio "Dynamic compact multicast routing on power-law graphs" in Proc. Globecom 2011, Houston, USA, Dec. 2011.

[Peleg89] D.Peleg and E.Upfall, A Trade-off between Space and Efficiency for Routing Tables, Journal of the ACM, vol.36, no.3, pp.510–530, 1989.

[Perry92] D.E.Perry and A.L.Wolf, "Foundations for the Study of Software Architecture", ACM SIGSOFT Software Engineering Notes, Vol.17, No.4, October 1992.

[RFC4271] Y.Rekhter, T.Li, and S.Hares, Ed., A Border Gateway Protocol 4 (BGP-4), Internet Engineering Task Force (IETF), RFC 4271, January 2006.

[Serrano08] M.A.Serrano, D.Krioukov, and M.Boguna, Self-Similarity of Complex Networks and Hidden Metric Spaces, Physical Review Letters, Vol.100, 078701, 2008.