VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NOVÝ ÚSVIT POJMENOVÁVÁNÍ, ADRESOVÁNÍ A SMĚROVÁNÍ NA INTERNETU

A NEW DAWN OF NAMING, ADDRESSING AND ROUTING ON THE INTERNET

TEZE K DIZERTAČNÍ PRÁCI
DISSERTATION THESIS NOTES

AUTOR PRÁCE AUTHOR	Ing. Vladimír VESELÝ
VEDOUCÍ PRÁCE SUPERVISOR	Prof. Ing. Miroslav ŠVÉDA, CSc.

BRNO 2013 – 2015

Abstrakt

Internet roku 2015 se potýká s problémy, které jsou důsledky špatného designu pojmenovávání a adresování v TCP/IP a jež mají přeneseny vliv i na škálovatelnost směrování. Problémy jako růst páteřních směrovacích tabulek, neefektivní multihoming sítí či mobilita zařízení a mnohé další zadávají k otázce, jestli není třeba architekturu Internetu pozměnit. V teoretické části je kvantifikován dopad problémů, možná řešení a zejména je formálně definována teorie kompilující poznatky významných publikací zabývajících se problematikou pojmenování, adresování a směrování v počítačových sítí. Tato práce se zabývá dvěma konkrétními technologiemi, jež mají ambicí Internet měnit - Locator/Id Separation Protocol a Recursive InterNetwork Architecture. Výstupem práce jsou vylepšení funkcionality obou výše zmíněných technologií. Za účelem praktického ověření dopadů našeho výzkumu jsou vyvinutý a popsány nové simulační modely pro OMNeT++, které jsou věrné úrovni detailu popisu ze specifikací.

Klíčová slova

Internetová architektura, pojmenovávání a adresování, směrování, oddělení lokátorů a identifikátorů, LISP, rekurzivní mezisíťová architektura, RINA, OMNeT++

Abstract

Internet of the year 2015 struggles with problems that are just implications of flawed naming and addressing the concept of TCP/IP, which have an impact on overall routing scalability. Problems such as default-free zone routing table growth, cumbersome multihoming or mobility motivate question whether the Internet deserves major architecture redesign. In the theoretical part, the impact of problems above is evaluated, solutions are discussed and unifying theory compiled and described using formal methods taking into account revered papers about naming, addressing and routing. This work provides in-depth Investigation of two technologies - Locator/Id Separation Protocol a Recursive InterNetwork Architecture. Research contribution is an operational improvement of technologies mentioned above. New OMNeT++, full-fledged simulation modules compliant with behavior in the specification are used to as verification tool.

Keywords

Internet architecture, naming and addressing, routing, locator/id split, LISP, Recursive InterNetwork Architecture, RINA, OMNeT++

Contents

1 Introduction

֍ –"*Yesterday is gone.Tomorrow has not yet come.We have only today.Let us begin.*" Mother Teresa
֍ *What are goals and motivations of this thesis?*

Nowadays Internet routing and addressing concept are facing a variety of challenges that were not so apparent in early days of the TCP/IP stack. Among those challenges, there are multihoming, mobility, traffic engineering, renumbering, **node** (a.k.a. **device**[1]) localization and identification and routing scalability connected with the growth of the global routing tables.

IRTF's RRG[2] was, and IETF's IAB[3] is for a long time in charge of observing trends in routing, collecting statistics and suggesting architectural recommendations influencing tendencies in future networking. In this thesis, we try to describe and evaluate the impact of these trends. Moreover, we gather relevant proposals and compare them with each other. Among documents and proposals discussed by IAB, there were also Locator/ID Separation Protocol (LISP) and Recursive Internet Architecture (RINA) that received both positive and also negative reviews. We believe that both of them are addressing the same fundamental issues (which serve as the motivation behind our research), but they employ a very different approaches. While the first one is trying to repair the most apparent problems using existing architecture. The objective of the second one is to redeem the Internet from scratch as the clean-slate architecture.

The encompassing (and challenging) dissertation goal is to define general naming and addressing theory. Moreover, we want to investigate properties of this theory regarding the impact on the routing. Because nothing impacts routing more (in either positive or negative way) than how names and addresses are employed to network objects. The underlying goal of this dissertation is to provide a detailed technical overview and analysis of two technologies (LISP and RINA) aimed at improving the current problems of the Internet. The contribution lies in enhancing LISP cache management algorithm and related data transfer to improve its performance. Moreover, we verified LISP contribution functionality with own accurate simulation models. For RINA, fundamental concepts were formalized using finite-state machine diagrams and a comprehensive set of simulation models was developed. Besides these two main achievements, this thesis provides a deep review of the building blocks of internetworking with the focus on naming and addressing concepts. The aim of the thesis is to shed more lights on the fundamental problems of the current Internet architecture and to evaluate the two of possible solutions

The thesis is divided into the following chapters. Chapter 2 provides an overview of current weaknesses of the Internet and describes factors that influence them. Chapter 3 outlines a general theory for addressing and naming using formalism and compares proposed or existing solutions. Chapter 4 presents protocol LISP, its implementation in simulator environment and covers proposed control plane improvements together with the measured impact of this proposal on the overall operation. Chapter 5 delineates RINA and its approach towards the system of recursive encapsulation of one general layer, and then it focuses on its globally first simulator implementation and measured aspects. Chapter 6 draws conclusions from the research outcomes.

[1] **Node** or **Device**: With reference of this thesis it is any equipment connected to Internet capable of communication. E.g. routers, switches, computers, etc.
[2] Routing Research Group (RRG). For more, please visit website of this former ad hoc group https://trac.tools.ietf.org/group/irtf/trac/wiki/RoutingResearchGroup.
[3] Internet Architecture Board (IAB). For more, please visit https://www.iab.org/.

2 Networking Fundamentals

֍ –"*You realize that our mistrust of the future makes it hard to give up the past.*" Chuck Palahniuk
֍ *What problems are tormenting the Internet now?*

If we want to be thorough when describing theoretical fundaments for this thesis, we need to start with the high-level overview of networking and work down to low-level parts. The Internet as technology is continuing sequence of evolutional steps. However, since its beginning it is all about few basic principles that had not changed. Internet architecture is about best-effort communication with global connectivity across the simple but resilient network where intelligence is on the end-to-end basis rather than hidden in the network as RFC 1958 [1] stated.

The goal of this chapter is to point out present problems of the Internet and discuss their impact on routing table size and control plane load.

2.1 Present Problems of Internet

Among some driven factors of nowadays Internet [2] are:

- the widespread availability of wireless (including Wi-Fi and cellular networks) connectivity allowing more non-PC devices perform ad hoc connections;
- deployment of virtualization increasing the number of logical computing systems;
- more cloud computing and peer-to-peer applications changing traffic characteristics towards less deterministic and stochastic models of CDNs[4];
- reaching the Zettabyte era more quickly due to the overall increase in broadband speeds.

Issues below are only consequences of Internet usage, which are completely different comparing to Internet conventions and user base 30 years ago.

What we are experiencing is that more and more **hosts**[5] and **routers**[6] are connected to the Internet every day using different wired and/or wireless technologies. Also growing the amount of transferred data comes hand to hand with an increasing number of users. Paths between nodes on the Internet are becoming shorter, faster, more redundant and more reliable. More existing IPv4 addresses are used as **Provider Independent (PI)**[7] rather than **Provider Aggregatable (PA)**[8] addresses of Internet Service Provider (ISP)[9]. The free IPv4 address space is depleted, and IPv6 is still fighting to reach at least 5% of overall traffic (see [3] as a representative statistic example of mid-size NREN[10]) despite the fact that it has been more than 17 years since its standardization.

During last years, lots of discussions were held (for more general information, please see [4], [5] and [6]) whether current Internet architecture could sustain its expansion in the middle and long range future. Somebody argues that new better resources are being invented faster than available technology could keep up tempo with them. Somebody disagree that every resource has physical boundaries that cannot be passed on, and that pose as a limiting factor. Nevertheless, the impact of the

[4] Content delivery network (CDN). For more, see https://en.wikipedia.org/wiki/Content_delivery_network.
[5] **Host**: Device that can send/receive packets, but does not participate in forwarding of packets.
[6] **Router**: Device that forwards packets across the network layer.
[7] **Provider Independent (PI)** addresses: Address prefix that organization receives from its Regional Internet Registry (RIR). Benefits of using PI addresses relies in fact that if organization needs to change ISP then it does not need to renumber its address space. ISP change means just slight change of routing information propagated to to DFZ.
[8] **Provider Aggregatable (PA)** addresses: Address prefix that organization receives from its provider. The PA address advantage is that all networks of a given ISP – components of ISP's address space – could be replaced with single aggregate prefix propagated to DFZ.
[9] Numbering of Internet is govern by Internet Corporation for Assigned Names and Numbers (ICANN) organization which assigns available prefixes to RIRs. RIR delegates prefixes to Local Internet Registries (LIR) which carry out assignments of address to their customers. LIRs usually operate as ISPs in that area.
[10] National research and education network (NREN). For more, see https://en.wikipedia.org/wiki/NREN.

current situation on routing on the Internet is something that we can clearly observe and at least partially predict future tendencies even though we have not yet reached limits of nowadays resources.

The most severe and apparent issues – namely routing table growth, lack of locator/identifier semantics split, cumbersome multihoming and mobility, ineffective inbound traffic engineering and renumbering due to the change of ISP – are listed down below in Sections from 2.1.1 to 2.1.6. Some of the problems are based on a review from RFC 6227 [7], RFC 4984 [8], some of them from mutual community observations of current trends. These problems are currently being solved by band-aid mechanisms and architecture patches (e.g., mobility frameworks). However, those solutions usually lack wide-spread deployment to be really deal breakers and/or do not seem to be long-term scalable.

2.1.1 Routing Scalability

The most affected nodes struggling with the situation are **Default Free Zone (DFZ)**[11] routers. Every year the size of **Routing Information Base (RIB)**[12] and **Forwarding Information Base (FIB)**[13] of those routers increases. The rate, at which a number of prefixes is growing in the RIB, is the object of discussions [9] but it seems to be slightly faster than linear (sometimes called *superlinear*) for a couple of last years [10], [11]. We can see historical progress in the size of Border Gateway Protocol (BGP) [12] RIB and FIB for IPv4 and also IPv6 on the following graphs depicted in Fig. 1, Fig. 2, Fig. 3 and Fig. 4 from [13]. The year is on the X-axis, and the number of prefixes is on the Y-axis.

Current numbers are taken from a router in one of the APNIC research and development **autonomous systems (AS)**[14]. They are relevant to the date of this publication:

- IPv4 RIB = 1 682 113 prefixes;
- IPv4 FIB = 573 400 prefixes;
- IPv6 RIB = 96 409 prefixes;
- IPv6 FIB = 24 857 prefixes.

Previous numbers mean that this particular router sees 573 400 IPv4 destination networks in today’s Internet where there are 1 682 113 different paths to them and vice versa for IPv6. The prefixes count is going to increase with advancing depletion of IPv4 space and progressing deployment of IPv6.

Each prefix must be processed which increases the **control plane**[15] load. This raises consumption of router’s CPU performance and memory and last but not least increases the size and a potential number of exchanged routing updates. This presents **routing scalability** issue for future routers – sometimes the same problem is also known as *DFZ RIB/FIB growth*.

It is believed [8] (assuming nowadays growth and available hardware and software) that we will still have resources to build devices capable of dealing with this problem efficiently. However, what is becoming a concern is a price of these devices.

The worst consequence can be that only large Tier 1 ISPs could afford investments connected with maintenance and operation of DFZ. Technologically the routers: a) must maintain increasing state information in RIB and converge usable routes quickly enough; b) must populate FIB from RIB fast and must be prepared for enlarging the size of FIB itself; c) must perform forwarding lookups (and at best also routing decisions) at line-rate speeds; d) must use HW that does have reasonable power consumption or cooling demands. From the business perspective, we must understand that DFZ is run

[11] **Default Free Zone (DFZ)**: Backbone of the Internet where routers must keep complete routing tables with all reachable destination networks. In opposite of this are Tier 3 ISP or networks or end customers that are using usually only partial routing information – they have complete knowledge about local connectivity and any other network beyond is available via default route.

[12] **Routing Information Base (RIB)**: Basically abstract data structure holding information from a given routing source that holds information about all reachable destination networks and paths to those destinations.

[13] **Forwarding Information Base (FIB)**: The FIB is optimized version of RIB. It is consulted most of the time when forwarding packets because it is supported by specialized HW.

[14] **Autonomous System (AS)**: Set of devices under one administration domain.

[15] **Control plane**: Part of the router responsible for maintaining routing table with the help of routing protocols and handling L3 issues (defragmentation, filtering, traffic classification/marking, QoS policing/shaping, cryptographic operations, etc.)

solely by private entities without any centralized supervision [14]. They are making a profit from it, so all the policies (like acceptance and processing of prefix) are in their hands. From their perspective, it is not beneficial that cost of routing infrastructure would grow too rapidly especially due to the factors they are unable to control (e.g. increasing number of Tier 3 ISP customers that want to multihome). To put it simpler – no ISP (especially the one operating at DFZ) would be happy from upgrading its infrastructure (buying better routers) to maintain the same level of service just because of the rising requirements of the routing system.

The final verdict (see [15]) is that vendors and ISPs are (under current conditions) able to deal with the growth of the Internet. However, scalable and cheaper solution would be welcomed to reduce costs and prepare for future demands.

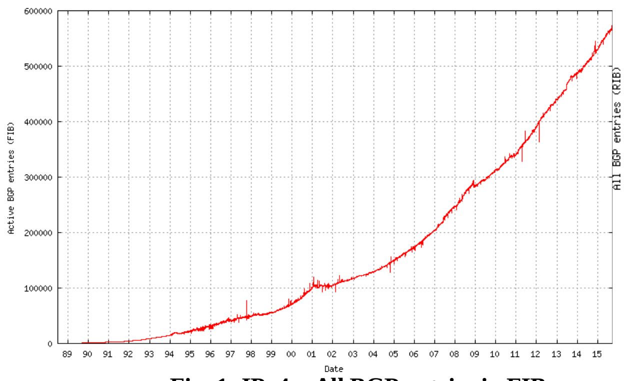


Fig. 1: IPv4 – All BGP entries in FIB

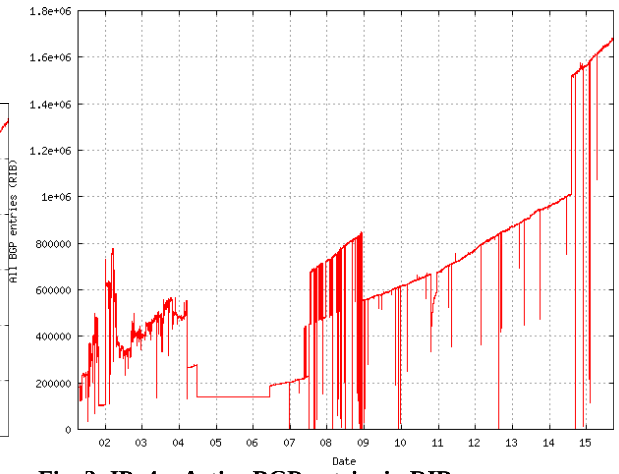


Fig. 2: IPv4 – Active BGP entries in RIB

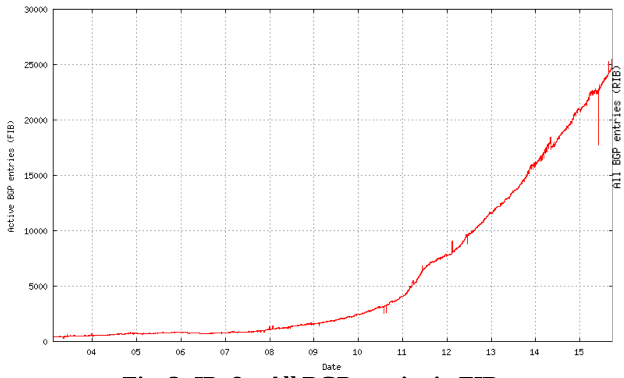


Fig. 3: IPv6 – All BGP entries in FIB

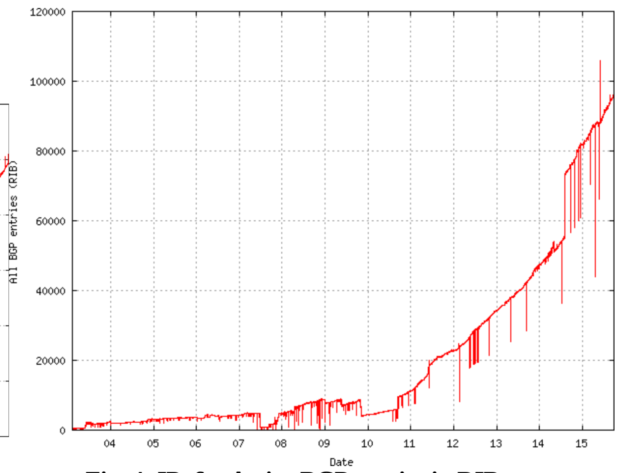


Fig. 4: IPv6 – Active BGP entries in RIB

2.1.2 Decoupling Identification and Location

RFC 2101 [16] was the first to lay down some theoretical fundaments decoupling identification and location (sometimes also referred to as *loc/id split*). IP address serves multiple roles nowadays:

1) *Identification* – **Identifier** is a bit string that is used during the communication's lifetime. It identifies communicating parties in a way that IP address verifies the source of packets.
2) *Localization* – **Locator** is a bit string that specifies packet destination where it should be delivered. It locates the place on the Internet, where a device is attached. Routing protocols interpret IP address as a locator and build up routing tables based on the situation that routers route traffic towards a destination. The locator is also known as **Point of Attachment (PoA)**[16].

[16] **Point of Attachment (PoA)**: Device's interface (and address of this interface) by which it is connected to some network reachable via Internet. Device could have and use simultaneously more than one PoA for communication.

Identifiers and locators have different requirements on uniqueness and lifetime. Identifiers must be unambiguous on each set of communicating parties while locators must be unambiguous within one or more routing domains. Identifiers must be valid at least during the maximum lifetime of communication between given devices. Locators must be valid as long as a routing system within a routing domain needs them.

Let us focus on real-life implications of the fact that IP address is used both as identifier and locator. What if any node has more than one IP address, which one identifies it? The topological device is situated at one place, although PoA addresses express the networks to which device is connected. Moreover, PoA could have a completely different location from the perspective of DFZ. Another example is multiple virtual machines on one host system. One approach is that we have a virtual network inside host system. However, in this case, we might run into the problem that we have to use NAT[17], or we lack address space. Another approach is that virtual machines share host system address. However, how can we then differentiate between virtual machines from a network perspective?

Of course that current TCP/IP status quo without any distinction between identifiers and locators works. However, locator/identifier separation would be the inherent solution for some problems discussed in this subchapter. The most notable advantages (see [17] for details) of decoupling locator and identifier are: a) reduction of DFZ routing tables because they would contain only locators, which would improve scalability of control plane; b) be design support for mobility and multihoming by employing mapping between two distinct namespaces (to one identifier may belong multiple locators) comparing to hacks when using only single namespace of blurred locators and identifiers.

2.1.3 Multihoming

Multihoming stands for the situation when the customer is using two or more ISPs for transit services as it is defined in RFC 4116 [18].

Below are some of the reasons why customers demand multihoming:

- *Redundancy* – Customers are looking for high availability of their services. Hence, their (both customers and ISPs) networks should be operational at best 99.999% of all the time (this represents approximately 5 minutes of allowed outage during whole year) to meet this constraint. From the perspective of Internet connection, this could be accomplished by having more than one ISP to avoid a single point of failure.
- *Load-balancing* – Traffic could be load-balanced between multiple working links leading into/out from customers AS to avoid congestion or to increase the available communication bandwidth.
- *Traffic Engineering* – Customer wants to influence how traffic is handled and treated, e.g., for example, to avoid problematic paths, to isolate some sets of addresses, etc. (for more, see RFC 2260 [19])
- *Transport-Layer survivability* – BGP driven multihoming provides at some level (i.e., successful convergence in certain time frame) session survivability for transport protocols;

A mandatory prerequisite for multihoming is that every customer is uniquely identified on the Internet – this is done by **autonomous system number (ASN)**[18]. Multihoming is nowadays accomplished with the help of BGP, which informs others about the path to customer's network via two or more ISP transit systems.

Multihoming works with PA and PI addresses. For both cases, customer's prefix is propagated to DFZ. Nevertheless, for PA case also primary ISP (assignee of customer addresses) must advertise both aggregates and also PA prefix. Otherwise, the path via primary ISP would not work because of **longest-prefix match**[19] routing lookup.

[17] Network Address Translation (NAT). For more, see RFC 1631.

[18] **Autonomous System Number (ASN)**: Globally unique identifier 16 or 32 bits long assigned by ICANN and maintained in online database on http://www.iana.org/assignments/as-numbers/as-numbers.xhtml.

[19] **Longest-prefix match**: Algorithm used by routers to retieve best available (the most accurate) entry from routing/forwarding table or any other table containing IP network entries. For more see D.E.Comer, Computer Networks and Internets (5th ed.), p. 368, ISBN 978-0-13-606698-9, 2008.

The trouble with multihoming is closely connected with IP address semantics problem described in the previous section – IP addresses PoA not a node. Reachability of multihomed networks is dependent on the route. However, IP routing takes into only account destination and next-hop IP addresses which identify PoAs.

Assume network graph in Fig. 5 with one router connected with two interfaces (two PoAs) to different ISPs for the sake of requested connection redundancy. If one PoA experiences outage (e.g., 192.168.1.1 on primary red route), then it does not imply that router and LANs behind it are unavailable. The routing algorithm can find a backup route for LANs, but it cannot help to reroute PDUs intended for PoA, which is currently down. Multihoming is not inherent use-case to IP. Route dependency of multihomed networks remains unsolved despite the fact that it firstly appeared in 1972 (more than 40 years ago) as Tinker Air Force Base multihoming request [20].

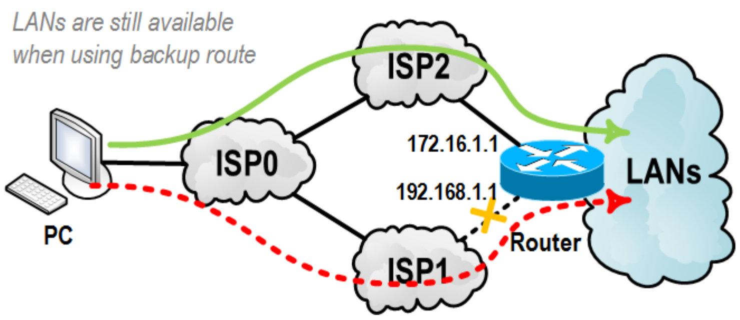


Fig. 5: Multihoming problem illustration

2.1.4 Mobility

During the last years, the idea of the **Internet of Things (IoT)**[20] became more real and widely accepted as probable use-case of Internet. Some predictions expect that 20-75 billion nodes will be connected to the Internet by the year 2020 [21]. Basically, a throng of devices with own IPv6 addresses would need access to the Internet. **Mobility** is the ability of a node or whole network to change its topological connectivity without disruption of ongoing communication (remark: *application mobility* is not covered in this thesis though it is often associated with this term). Authors of TCP/IP stack had never thought about this use-case. Thus, IETF had to supplement solutions like Mobile IP [22], Mobile IPv6 [23] or HMIPv6 [24] or Multipath TCP [25] later.

These solutions include:

a) Dynamic renumbering of mobile entity – considered unsuitable because dynamic IP address change without any further notice may disrupt existing communication.
b) Renumbering and creating a tunnel between old and new location – it requires the deployment of the *home agent* and *foreign agent* concepts known from cellphone networks.
c) The ability of a mobile entity to actively announce its new location – usually comes hand to hand with dynamic changes to DFZ routing tables as the mobile entity moves from one location to another.

A looming current problem (for not just IoT) is how to accommodate possibly billions of smartphones, tablets, printers, and PDAs with the IPv4/IPv6 capability to access the Internet and to provide session survivability when those devices roam from one network to another. NAT is often being used to overcome this limitation. However, NAT breaks **end-to-end principle**[21] [26] and due to that NAT is being considered as the temporary fix rather than a solution. Mobility should not be attained feature of some special protocol or technique. Therefore, mobility support should be inherent to the network architecture.

[20] **Internet of Things (IoT)**: Refers to unique identification of objects in Internet where nearly any device is equipped with IP address and capable of communication via IP. It is merely buzzword overused in marketing expectations of future Internet growth. More at http://en.wikipedia.org/wiki/Internet_of_Things

[21] **End-to-end principle**: Application-specific functions ought to reside in the end hosts of a network rather than in intermediary nodes.

2.1.5 Traffic Engineering

Traffic directing and diversion to use other paths than those precomputed by IGP[22]/EGP[23] is called **traffic engineering (TE)**. We differentiate between two types according to direction of traffic flow:

- *Outbound traffic engineering* – Intra-AS TE, where we try to influence how traffic is leaving AS. IGP metrics is usually altered to support this goal so that preferred exit from AS is utilized. Another way, how to accomplish outbound TE, is to depreferentiate or to filter some routes from BGP neighbors.
- *Inbound traffic engineering* – Inter-AS TE, where more specific routes are propagated with the help of BGP to divert traffic from normal paths (aggregated prefixes). Those altered specific routes are more preferred because they temper BGP decision process [12], [27].

Nowadays inter-AS TE is done rather than intra-AS TE. Reasons, why to do TE, are similar just as in the case of multihoming. Among those reasons are load-balancing (to match traffic with network capacity), policing (to restrict transition of certain traffic through a given AS), cost reduction and support of various QoS and **Service-level Agreements (SLA)**[24].

TE is performed by tuning BGP attributes of the certain routes and/or introducing more specific prefixes into DFZ routing tables. This effectively increases RIB and FIB sizes and presents an additional load to the control the plane. Moreover, network administrators spent hours configuring TE only to discover that the neighboring BGP peer completely rewrites (or ignores) routes attributes, thus preventing the rest of the Internet to learn and conform to intended TE. Hence, network architecture should support nonrefusable TE by design.

2.1.6 Renumbering

Usually, the organization has one ISP where its network is completely inside ISP's AS. In this case, the **organization**[25] does not need to advertise its network prefix globally because it is a part of provider address space – PA addresses is assigned to the organization. However, if an organization wants to change ISP, then it must be prepared to **renumber** all its nodes according to PA address block enforced by a new ISP. Another option is to ask Regional Internet Registry (RIR)[26] for PI address block, but there are two drawbacks associated with it:

1) The organization still would not avoid at least initial renumbering when changing from PA to PI addresses.
2) The demand could not be met because RIR is already missing PI prefixes large enough (especially with IPv4 address space depletion), or it is against RIRs regulations. PI addresses make the process of migrating between ISP easier; still each PI prefix must be separately advertised to DFZ.

Not only renumbering process could be costly and error-prone (see RFC 5887 [28]) even with the existence of automated tools (e.g. DHCP, SLAAC, etc.), but also some of the organizations may feel stuck or being held as a hostage of theirs ISPs that provide them with PA prefix.

The renumbering problem grows with the size of the network and number of nodes it contains. Moreover, change of host's addresses negatively affects **access control lists** and firewall setups or configuration files outside the scope of renumbered network.

[22] Interior Gateway Protocol (IGP). For more, see http://en.wikipedia.org/wiki/Interior_gateway_protocol.
[23] Exterior Gateway Protocol (EGP). For more, see http://en.wikipedia.org/wiki/Exterior_gateway_protocol.
[24] **Service-level Agreement (SLA)**: SLA is an agreement between two or more parties, where one is the customer and the others are service providers. SLAs commonly include segments to address: a definition of services, performance measurement, problem management, customer duties, warranties, disaster recovery, and termination of agreement. For more, see https://en.wikipedia.org/wiki/Service-level_agreement.
[25] **Organization** a.k.a. **Customer**: Entity operating end network with own addressing plan and routing policies.
[26] **Regional Internet Registry (RIR)**: Organization that manages allocation and registration of internet numbers (IP addresses, autonomous system numbers, well-known port, etc.). Currently world is divided into five RIR based on geographical position: AfriNIC, ARIN, APNIC, LACNIC and RIPE NCC.

2.2 Burden on Control Plane

The count of routing updates has the major influence on control plane processing delay. Among elements impacting it belongs:

- *Interconnection Richness* – The Internet is becoming flatter in a sense that more and more different paths exist between the same ASes [29]. Unfortunately, this interconnection richness is stressing control plane seriously, and it occurs even though the prefix count remains the same;
- *Traffic Engineering* – More specific prefixes with different attributes expressing desired TE effect place more overhead on control plane;
- *Multihoming* – Multihoming AS neighboring with more than one ISP (transit AS) requires more than one interconnection leading towards DFZ;
- *Rapid Shuffling of Prefixes* – Some ASes deploy rapid shuffling of prefixes in order to divert traffic to less loaded links or to optimize traffic by depreferencing (or even canceling) certain routes that do not meet SLA criteria;
- *Anti-Route Hijacking* – Owning AS advertises purposely more specific prefixes as the countermeasure when fighting against **IP hijacking**;

The previous list outlined some of the reasons, why are there many more specific prefixes in BGP and why is the router's control plane bothered with irregular routing updates.

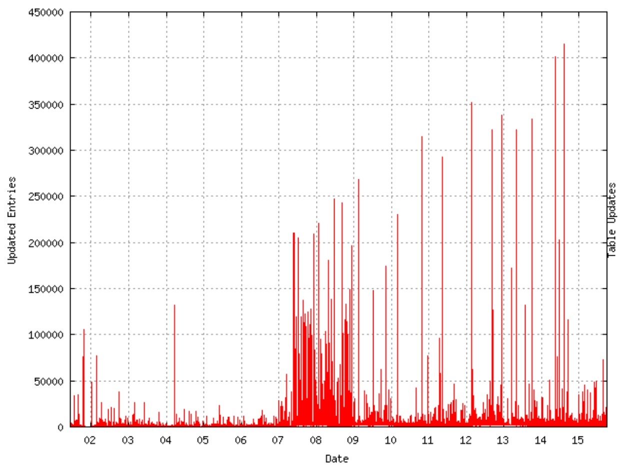


Fig. 6: IPv4 FIB table updates

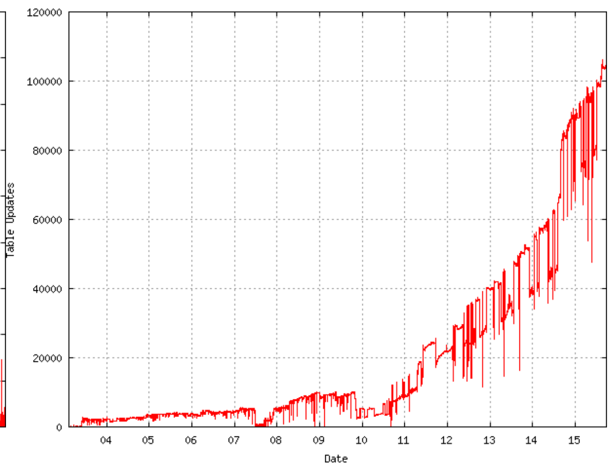


Fig. 7: IPv6 FIB table updates

Graphs in Fig. 6 and Fig. 7 depict the number of FIB table updates for both IPv4 and IPv6. Currently, BGP is experiencing approximately 1 500 updates per hour for IPv4. If there are peaks in IPv4 then they are getting larger and massive (two orders of magnitude) comparing to the usual state.What is more surprising is that this number is approximately 104 400 updates per hour for IPv6. This implies that current IPv6 setup is more intensive on the control plane.

2.3 Chapter Summary

We mentioned problems tormenting nowadays Internet – routing (in)scalability, decoupling location and identification, cumbersome multihoming, overcomplicated mobility, the impact of inbound traffic engineering and unwieldy renumbering of end-networks. We outlined negative consequences in the frame of TCP/IP for each mentioned problem and discuss their impact on the control plane.

This chapter content should support the conclusion that also others come to – the current Internet architecture shows design flaws and sooner or later it will face the crisis emerging from consequences of its poor design.

3 Naming and Addressing Concepts

֍ –"*Now you people have names. That's because you don't know who you are. We know who we are, so we don't need names.*" Neil Gaiman
֍ *Can we formulate any encompassing theory of naming, addressing and routing?*
֍ *What about any solutions dealing with aforementioned problems?*

Problems of addressing and naming are closely connected with networking since its beginning. It directly affects the efficiency of routing and forwarding. Once syntax and semantic of device addressing are employed, the whole system is hard to change. The current Internet addressing scheme is the most obvious example of this problem. Although the present IPv4 address scheme has improved since its definition in the 1980s, it currently represents the major obstacle not only because of address depletion problem but also for deployment of multihoming and mobility to name a few of the issues.

The role of IPv4 is to identify and localize the communication interfaces of connected devices. This principle works fine if address assignment follows the network topology and network devices are preserving their membership to local networks. Many other concepts of the Internet were built around this assumption. Communication between network applications requires identifying addresses of network interfaces where the applications are reachable. Enabling IP address change during communication would require modification of datagram delivery mechanism causing complications for network devices as well as for end points. Routing architecture can efficiently react to connectivity changes detecting dead routes or identifying new routes or routes with better metrics. While exterior gateway routing protocol BGP provides flexibility for propagating information about relocating IP address this always leads to growing global routing tables because of breaking address to topology location dependency. This has a negative impact on routing performance.

The goal of this chapter is to provide the necessary background for practical part of this dissertation thesis (next two chapters). We try to outline basic motivation why naming and addressing are still issues of current Internet architecture, which is majorly based on Vint Cerf's and Robert Kahn's TCP/IP from 1974.

In the first subchapter, we layout basic terminology using formal apparatus. In Subchapter 3.2, we try to synthesize working theory employing knowledge from revered articles on this topic. Then, we test compliance with TCP/IP related protocols and tools with this theory. The longest Subchapter 3.3 describes conceptual properties of the ideal solution and introduces many of existing candidates.

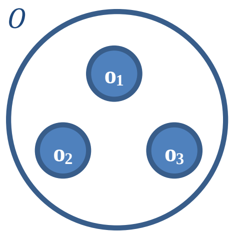
3.1 Basic Terminology

This introduction provides theoretical foundations of naming and addressing. Namely it puts together all pertinent knowledge regarded with deep and utmost respect to papers by John Shoch [30], Carl Sunshine [31], Jerome Saltzer [32], Noel Chiappa [33] and John Day [20].

Natural thinking about basic terms yields following meanings:

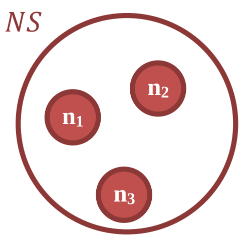
- the object is a structure that is considered to be worthy of the distinct name or address;
- the name identifies what object is;
- the address identifies where the object is;
- the route identifies which direction object is;

Naming

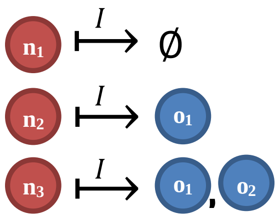


Let us start with an object. **Object** o is a software (or hardware) structure that is considered to be worthy of identification (e.g., variable, service, interface). All *objects* of the same type form a set $O = \{o | o \text{ is } object\}$. We can work with a single *object* or a subset of *objects*, thus it is important to define *power set of objects* $\mathcal{P}(O)$.

Now, let us settle on the meaning of the following terms regarding naming. To be more accurate and consistent within this theory, we define **name** as a *string over the alphabet*[27]: $\forall n$ is *name* $\Leftrightarrow n \in \Sigma^*$. However, it is important to note that *name* may be any kind of identifier (e.g., string, color, number). All names form the **namespace** as a set of *names* $NS = \{n | n \text{ is } name\}: NS \subseteq \Sigma^*$ from which all *names* for a given set of *objects* are taken.

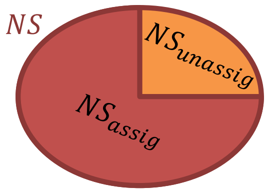


Any *name identifies* (a subset of) *object*(s), **identify** is relation $I: NS \times \mathcal{P}(O)$. Previous definition allows *name* to *identify* none, one or even more *object*(s) of O. Identifying more than one *object* may be useful for use-cases such as multi-cast or broadcast communication.



Imagine space of IPv4 addresses; some address blocks are assign to owners (e.g., FIT-BUT's address block 147.229.0.0/16), some addresses from these blocks are being used by devices (e.g., private addresses), some addresses cannot be even sold (e.g., block 240.0.0.0/4 of reserved class E addresses). Naming theory should be granular enough to support all previous use-cases.

Assignment marks *name* in the *namespace* as available for *binding*, **deassignment** reverses this operation. Hence, the *namespace* is composed of two disjunctive sets of *names*, *assignable* NS_{assig} and *unassigned* $NS_{unassig}$: $NS = NS_{assig} \cup NS_{unassig}: NS_{assig} \cap NS_{unassig} = \emptyset$.

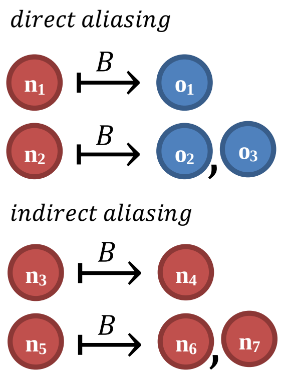


Binding is choosing a mapping from *assigned name* to a particular (subsets of) *object*(s) xor (subsets of) *name*(s); unbinding reverses this operation:

- *binding* is relation $B: NS_{assig} \times M, M = \mathcal{P}(NS) \cup \mathcal{P}(O)$.

  Name can be either *bound* or *unbound* (available for *binding*):

- *name* $n \in NS_{assig}$ is *bound* $\Leftrightarrow \exists m \in M: (n, m) \in B$;
- *name* $n \in NS_{assig}$ is *unbound* $\Leftrightarrow \forall m \in M: (n, m) \notin B$.

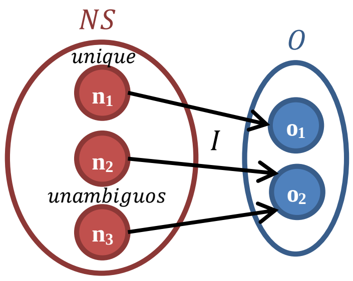


Please notice, that *name* can be *bound* to either *object*(s) a.k.a. **direct alias** or other *name*(s) a.k.a. **indirect alias**. Improper *indirect aliasing* may cause circular referencing (e.g., *name* "a" is bound to *name* "b" and *name* "b" is *bound* to *name* "a"), which is undesired. Hence, a chain of bindings should end with *direct aliasing* providing *identification* of (set of) *object*(s).

We can measure distinctiveness of *name* using following adjectives. **Unique** indicates that there is one and only one *identifying name*, whereas **unambiguous** indicates that there is possibly more than one *identifying name*:

- *name* $n \in NS$, which *identifies* $o \in \mathcal{P}(O)$, is *unique* $\Leftrightarrow \nexists n, \bar{n} \in NS: (n, o) \in I \land (\bar{n}, o) \in I \rightarrow n = \bar{n}$.
- *name* $n \in NS$, which *identifies* $o \in \mathcal{P}(O)$, is *unambiguous* $\Leftrightarrow \exists n, \bar{n} \in NS: (\bar{n}, o) \in I \land (n, o) \in I$.

Indirect aliases may be *bound* to *unique name* without breaking its uniqueness. Usage of multiple *direct aliases* changes the *unique name* to *unambiguous*.



Making Address Topological

Before investigating terms concerning *address*, we need to define terms related to *topology*, which are based on [34]. **Topology** on a set X is a collection $\mathcal{T}$ of subsets X having following properties:

- $\emptyset$ and X are in $\mathcal{T}$;

[27] Let Σ be alphabet, the set of symbols $\{a\}$. Σ^* is the set of all finite sequences w in alphabet Σ in form $w = a_1 a_2 a_3 \ldots a_n$, where any symbol $a_i \in \Sigma$ for $i = 1, \ldots, n$. We call w as the **string over alphabet**.

- The union of the elements of any subcollection of $\mathcal{T}$ is in $\mathcal{T}$;
- The intersection of the elements of any finite subcollection of $\mathcal{T}$ is in $\mathcal{T}$;

Fig. 8 illustrates three examples of topologies $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ (in compliance with definition) and three examples of non-topologies $\mathcal{T}_4, \mathcal{T}_5, \mathcal{T}_6$ (properties of topology are not met).

Topological space is an ordered pair $(X, \mathcal{T})$ consisting of a set X and *topology* $\mathcal{T}$ on X.

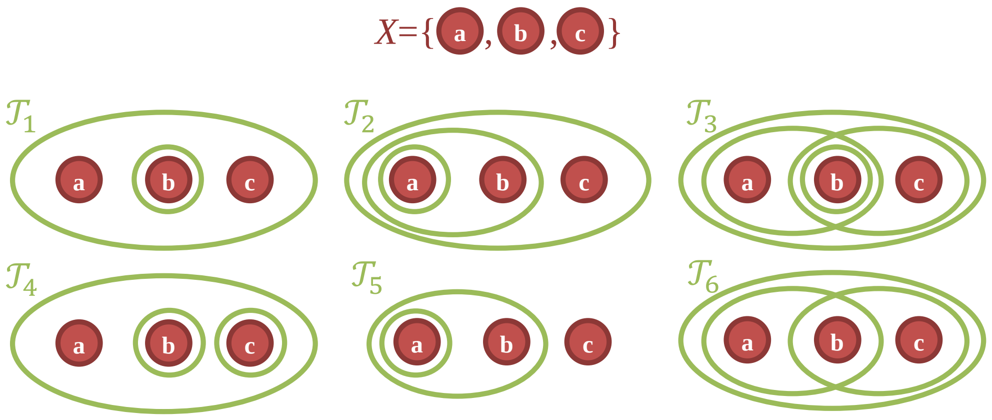


Fig. 8: Examples of topologies and non-topologies

Function $f: X \rightarrow Y$ between two *topological spaces* $(X, \mathcal{T}_X)$ and $(Y, \mathcal{T}_Y)$ is called a **homeomorphism** if it has the following properties:

- f is a bijection (one-to-one and onto);
- f is continuous;
- the inverse function f^{-1} exists (and f is an open mapping);

If *topological spaces* $(X, \mathcal{T}_X)$ and $(Y, \mathcal{T}_Y)$ are *homeomorphic* (if *homeomorphism* exists) then it is guaranteed that points "near" point $x \in X$ are mapped to points "near" point $y \in Y$ (e.g. in Fig. 9);

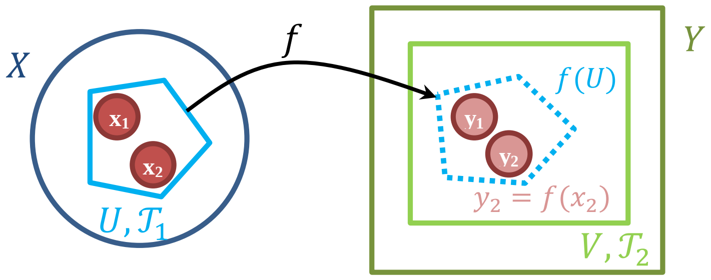
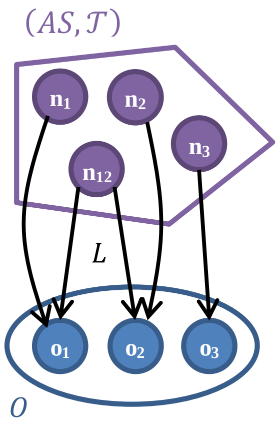


Fig. 9: Homeomorphism illustration

Addressing



Let us return to terms important for addressing. The **address** is a **topologically dependent** *name* (i.e., *address* contains leads about the position in topology). **Address space** *AS* is a set of *addresses* $AS = \{a | a$ is *address*$\}$ with a given scope. *Address space* is *topological space*, it is a *namespace* with a *topology* $\mathcal{T}$ imposed on it: $(AS, \mathcal{T}): AS \subseteq NS$.

We can perform same operations (e.g., *assign*, *bind*) and observe same properties (e.g., uniqueness) with *addresses* as with *names*. *Address locates* (a subset of) addressable *object*(s): **locate** is relation $L: AS \times \mathcal{P}(O)$. Instead of *identifying*, we are using term *locating* concerning *addresses*. However, both *identify* and *locate* are relations with same outcomes.

Resulting Properties

The *name* need not to be meaningful throughout the domain and need not be drawn from a uniform *namespace*, whereas the *address* must be meaningful and must be drawn from uniform (flat or hierarchical) *address space*. Flat *address space* has limitations; most notably no hierarchy leaves routing action without any help. Hierarchical *address space* has pros (reduction of routing table sizes) and also cons (what is topologically close may be far away on hierarchical tree branches, which leads to suboptimal routing). However, any structure/hierarchy in the *name* or *address* is intended to make some operation easier (i.e., search for an identifier in a directory).

The *address* is a *name*, but the *name* is not necessarily *address*. The *address* is bound either to name(s) or object(s) in order to *locate* it(/them). Therefore, the *address* is always a pointer in *topology* (e.g., position of the node in the graph, grid coordinates). The name is merely a label without any context to location.

The **route** is the specific information needed to forward a piece of information to its specified *address*. Routing action may require one or a series of steps in order to forward information to reach a destination. There should be mechanism mapping *address* into an appropriate *route*.

Address is **location dependent** if it encodes (even the part of) *topology* information (i.e., *address* string depends on where the address is present in the topology). *Address*, which is **route dependent**, encodes (even the part of) *route* information. Because there may be more than one *route* to a given location, we want *addresses* to be *location dependent* but *route independent*.

3.2 Theory

Employing knowledge from ISO/IEC 7498-3 [35], Saltzer's RFC 1498 [36] and Chapters 5 and 8 of Day's book [20], we will try to postulate some synthesis of the naming and addressing theory.

The *object address* is a *name* of the *object* to which it is *bound*. The *object* cannot be *located* without *identification*, nor can the *object* be *identified* without *localization*. Therefore, no reason exists, why to distinguish term *name* from *address* because *identifying* and *locating* the *object* are relations yielding same results. Hence, this means that *object name* and *object address* are same because they do not *identify* distinct *objects*. E.g., if "OBJ" is the *name,* then it is also its *address*, which help us to *identify/localize* an *object* in the scope of other objects. The previous statement is the final resolution of name-address dichotomy.

There are three objects that should be named in computer networks:

1) **services/applications/users** – Services are functions that are being used, e.g. service is Internet browsing. The application is using services, e.g. Internet browser. The user is a particular computer running Internet browser. Difference between service, application and user are in this sense non-essential, and we are going to use them indistinguishably within this subchapter;
2) **nodes** – Nodes are computers that run services. Some nodes are hosts (service consumers) while other nodes provide auxiliary functions to run services (e.g., routing and forwarding by routers);
3) **network attachments points** a.k.a. PoAs – PoAs are (Internet-connected) interfaces of a given node;

The natural way, how to relate to previous *objects*, is to use terms *application/service name, node address, network attachment point address*, even thou that we could use *application address* or *network attachment point name* in compliance with this theory.

Following three *bindings* exist between *objects* above:

1) **directory** – Directory is *service* to *node* mapping used to find service's location (i.e., communication endpoint);
2) **routes** – Route is a sequence of *node addresses* calculated by the routing algorithm; route interconnects a given pair of source and destination *nodes*;
3) **paths** – Path[28] is a *node* to PoA mapping of the nearest neighbor (i.e., next-hop); path interconnects PoAs of adjacent *nodes*.

Naming and addressing are free to use any form of identifier that seems helpful. It could be a binary or printable character string. The *namespace* and *address space* could be flat or hierarchical; the

[28] Term "path" here differs from path known from graph theory. Better would be to use "link", however, we follow original terms by Jerome Saltzer.

same *object* can even use different identifiers a.k.a. *aliases*, where some of them may be flat and others hierarchical.

Naming requirements (for more about them in frame of general networking, please see [31]) can be described in terms of *bindings* and *binding* changes among *objects* mentioned above:

- A given *service* may run on one or more *nodes*. Any *service* may need to move from one *node* to another without losing its identity;
- A given *node* may be connected to one or more PoAs. Any *node* may need to move from one PoA to another without losing its identity;
- A given pair of PoAs may be connected by one or more *paths*. Any of those *paths* may need to change without affecting the identity of the PoAs.

Each requirement contains some identity preservation, which is guaranteed when the *name* does not change during the moves – *object name* must be invariant when referring to some property of particular scope. This can be accomplished by maintaining a list of *bindings* between *services*, *nodes*, and PoAs. Basically, we *name* proper *objects* and then keep track of *bindings* between them.

To wit, *service/user names* do not change with location, *node names* do not change as PoA endpoints, and PoAs do not change as particular path endpoints. However, following rules do not mean that *names* should be assigned to a given *object* only once, and they cannot change after that. Essentially, *names* could be changed but this act must comply with previous requirements. Also, the *identity* of an *object* exists regardless of whether we can express it with some *name*.

If we want to send a packet to a given *service*, then following actions are done:

1) Find *nodes* on which the requested *service* operates. The task is *service name* resolution, which consists of *directory* search in order to discover a proper *binding* between *service* and *node*(s);
2) Find *routes* between source and destination *nodes* and pick the next-hop *node*, where the packet should be forwarded. This process is a.k.a. **routing**, where the initial result is *route* as the sequence of *node names*, and next result is next-hop *node name*;
3) Find PoAs of the next-hop *node* en route, i.e., perform *node name location* to reach *node*(s) found in the previous step;
4) Find *paths* between the current and the next-hop *node's* PoAs, i.e. discover the *binding* between the same PoAs pair and the *path*. This action is done by identifying a set of *paths* which leads among PoAs acquired in the previous step.

Each of previous steps might return either single or multiple alternatives. In the case of multiple returned *objects*, a choice must be made which of them to use. While these choices are distinct, they might interact – e.g., we may swap communication to a different *node* running the same *service* according to the *path* aptness.

We can easily satisfy basic *object*'s properties using this theory – what it is, where it is and which way it is. To wit, when speaking of network *applications*, the *service name* provides an answer to *what*, node and PoA names provide answer to *where*, routes and paths provide answer to *which way*. The difference between *node address* and PoA *address* allows us to create a logical over the physical *address space* relation. A network addressing system must support at least one level of indirection.

Resulting mode of this theory is illustrated in Fig. 10. Let us briefly inspect emerging properties of this model:

- *Directory* and *path* mappings are similar in a way that both of them track the *binding* of *objects* one hop away.
- Two *nodes* could be interconnected via multiple distinct *routes* (containing different interim nodes).
- Two adjacent *nodes* could be interconnected via multiple distinct *paths* (separate physical connections).
- The *route* could be viewed as a concatenation of *paths* in a relaxed context.

The application name should be *location independent*. Node address should be *location dependent* (the logical address). PoA names are *route dependent* (the physical address). PoA *address* should be *unambiguous* only within a particular scope, and PoA *addresses* need not to belong to the same *namespace* (e.g., Ethernet and FDDI addresses are from different *namespaces*).

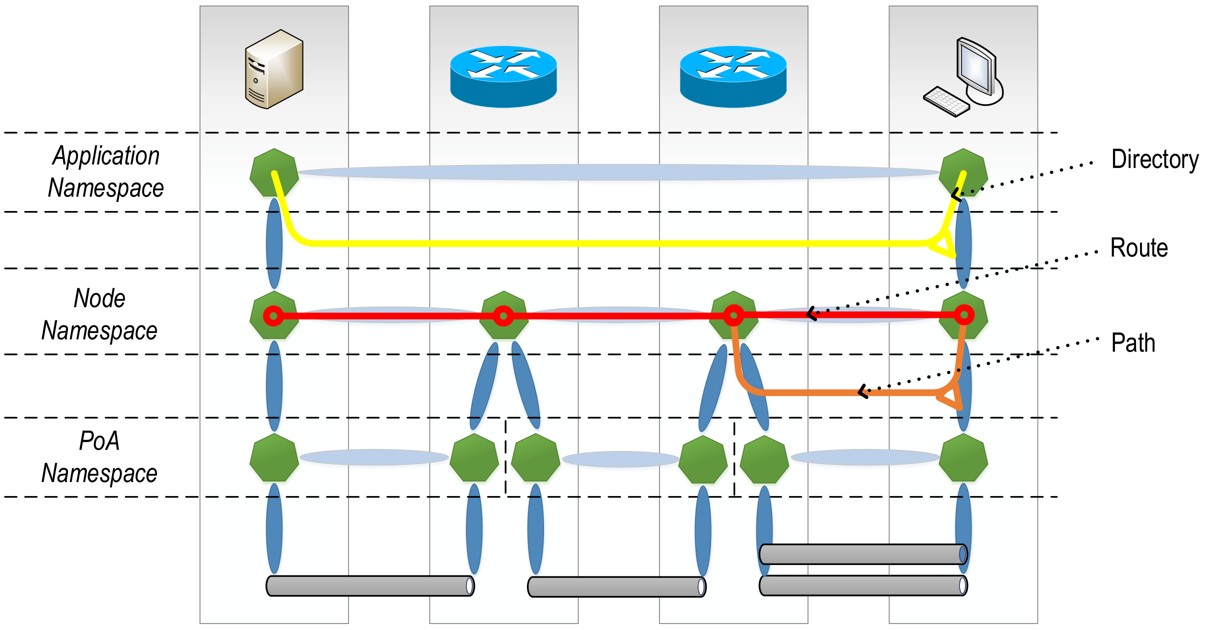


Fig. 10: Theoretical naming and addressing model for computer networks

Despite the fact that Saltzer's and Schoch's papers are more than 30 years old and extensively cited, very few have been done to integrate their ideas into computer networking praxis. To wit, at least two following fundamental requirements exist for a correct addressing and naming system: 1) recognition of objects – applications, nodes, and PoAs; 2) distinguishing changeable bindings – application to node, node to route, node to PoA, and PoA to path.

Unfortunately, IPv4 does not follow those two requirements at all! Current Internet architecture contains only PoAs and routes; it completely misses application and node names. IPv4 address ought to identify a node, but it retains semantic of interface address. Unfortunately, this makes multihoming impracticable because IPv4 address labels only node's PoA not a node itself. What is worst, IP address names the same thing as MAC address. Routes are then falsely bound to an IP address. Instead of the general directory, the Internet is stuck with well-known port numbers (SSH is on 22, Telnet on 23, SMTP on 25, HTTP on 80 and so on) and they are no more than a suffix to the network address. Basically, a node layer is missing. On Fig. 11 current broken model is depicted:

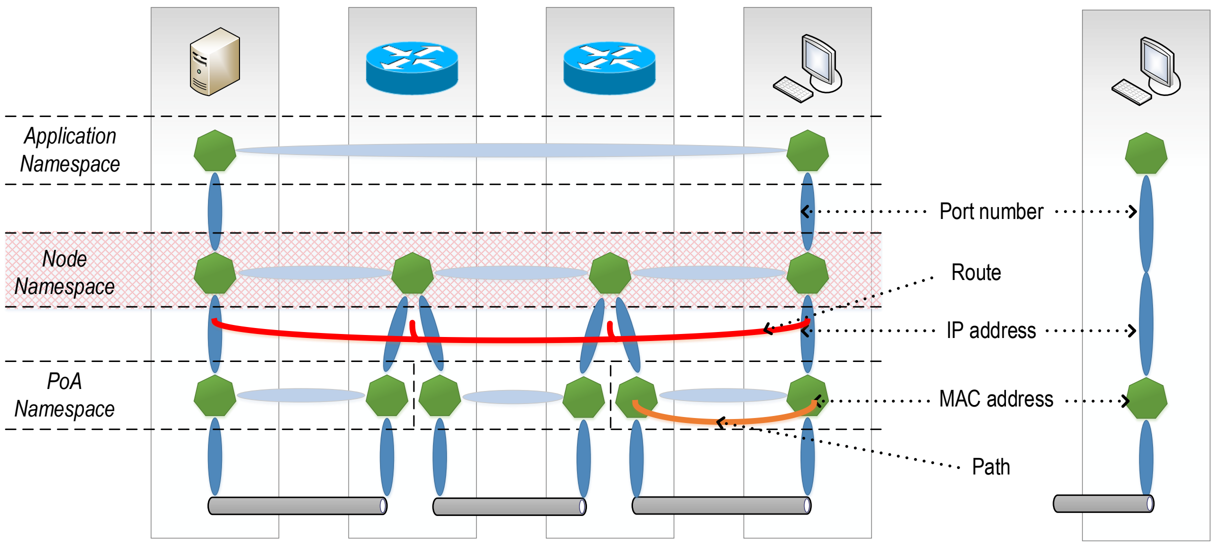


Fig. 11: Broken Internet naming and addressing model

Previous illustrates that major flaw exists in current TCP/IP naming architecture. Mostly because of this poor design, Internet suffers from issues described in Subchapter 2.1.

3.3 Possible Solution

This subchapter introduces theoretical properties of any solution based on Subchapter 2.1 analysis and RFC 6227 [7]. Moreover, it describes and compares features of existing candidates. Details of each candidate are out of the scope of this summary.

3.3.1 Ideal Solution Properties

One of the major goals for any upcoming change of the Internet architecture is to make the routing system scalable with respect to a number of prefixes, users and interconnections between autonomous systems.

As stated above overloading of IP address semantics causes collisions and limited flexibility. Hence, it is expected that a solution would decouple identifier namespace from location address space. Nevertheless, there are two approaches how separation should be performed: a) by splitting hosts, identifiers, and locators; b) by removing end-site prefixes from globally routable prefixes. The solution should contain the fix and should be compatible with either case. Ideally identifiers should be allocated at the birth of object, they never change, nor are they re-used. Hence, identifiers must be location-independent. Locators should point to device's position in the network, and they should change whenever the topology changes, thus locators must be *location dependent*.

The more scalable solution for multihoming is strongly desired to allow organizations multihome without adding pressure to DFZ routing tables.

As for mobility more efficient approach is wanted that allows mobile entity topological changes at a high rate. Hypothetically ideal solution should decouple mobility completely from routing.

TE is a necessity for a network operation of any organization. However, solution for inbound traffic engineering should pose no burden to the scaling of the routing system.

Renumbering is an inconvenience for either small or large scale networks. Even with the existence of working methodologies like RFC 4192 [37] how to renumber without the Flag Day it is still difficult to make this process cheap and smooth for any organization. Therefore, it is required that organizations could renumber their networks easily with as less disruption as possible.

Previous features refer to existing and above thoroughly described issues. Nevertheless, there are two more properties, which any solution should incorporate. The routing system is secured through additional protocol-specific mechanisms (i.e. mutual authentication of routing updates with the help of HMACs) that were introduced later during target routing protocol lifecycle. Hence, the solution must provide the same level of routing security, or better must be secure by design. Also, any solution must be deployable from technical and practical perspective – it must allow incremental deployment and provide necessary backward compatibility with nowadays employed services.

3.3.2 Existing Proposals

RFC 6115 [38] clearly states that: a) RRG has rough consensus on separating identity and location of devices but does not have consensus how to do it properly; b) RRG has consensus that multihoming and traffic engineering issues need to be solved in a scalable manner.

Theoretically, there are three ways how to decouple identity and locality:

- *Map-and-encap network-based architecture* – It evolves from Robert Hinden's ENCAPS protocol [39]. When a source sends the packet towards destination outside of source network, the packet must traverse through border router between two address spaces (locator space and identifier space). Here at first border router performs mapping of an identifier to appropriate locator ("map" phase). Then the packet is encapsulated using returned locator address ("encap" phase). Hence, map-and-encap principle wraps a new header (called *outer header*) using locator addresses around the original header (called *inner header*) with identifier addresses. When encapsulated packet reaches the destination network, the border router strips off the outer header and sends the original packet towards the receiver. Map-and-encap usually does not require changes to hosts or to the core routing infrastructure (that is DFZ). Unfortunately, with additional overlay encapsulation comes size overhead.

- *Rewriting hybrid network-based architecture* – Originally this principle comes from papers written by Robert Smart and David Clark 8+8 [40] and later by Mike O'Dell GSE [41]. It utilizes IPv6 so that in the upper part of IPv6 address PCI's fields is stored locator and in the lower part identifier. If a source sends packet outside its domain, border router takes addresses containing only identifiers and fills upper bits with appropriate locators. Then locators are removed from addresses upon reception by destination border router. Rewriting schemes may differ whether they perform either destination or both destination and source addresses rewrites;
- *Host-based architecture* – Decisions in this architecture are purely in hands of hosts. Thus, hosts prepare and fill all relevant PCI fields (including locators and identifiers) as the packet is being dispatched by the operating system. Interim devices like routers are usually transparent to this approach.

According to [42], possible solutions could be categorized into two classes that are not in the opposite. Over the years following terms were established to describe them:

- **Core-Edge Separation (CES)** – A subset address space (*edge*) corresponding to end site addresses is separated from the transit DFZ (*core*). This "edge" address space is then handled differently for routing. Subsequently DFZ routing table increases its site only a new ISP transit network instead of a new edge network. Some mapping system is needed to glue core and edge address spaces. CES is depicted schematically in Fig. 12 where it shows communication between *PC-A* and *PC-B* using (green) identifiers and (red) locators;
- **Core-Edge Elimination (CEE)** – The goal of CEE is to eliminate all PI and de-aggregated PA prefixes from the core. Hosts then use either PA addresses provided by ISPs or usually something different (not in IP address namespace) as an identifier. Some changes in host network behavior are necessary to deploy CEE. Illustrated in Fig. 13.

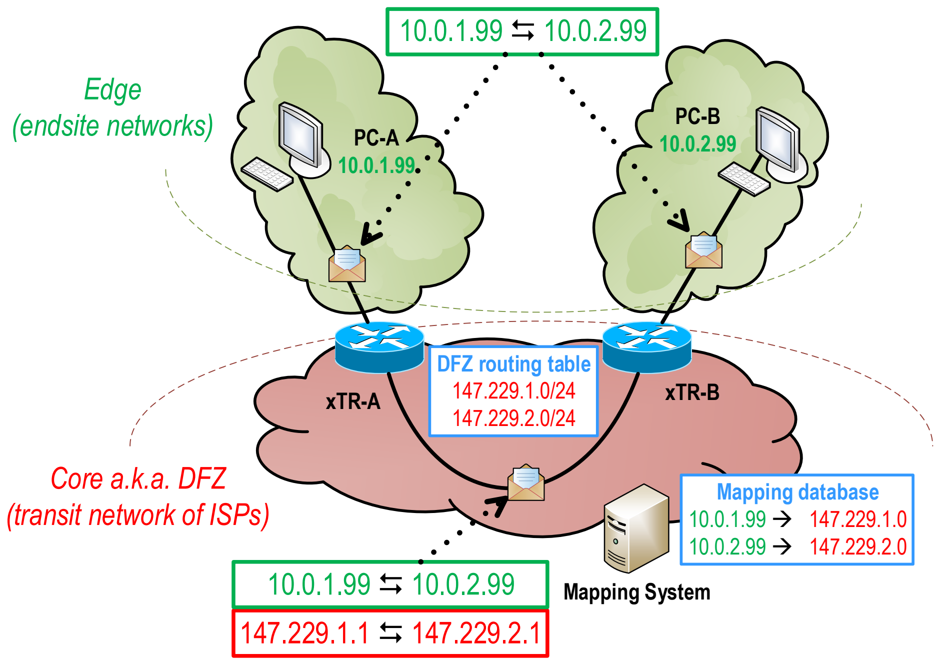


Fig. 12: Core-Edge Separation solution

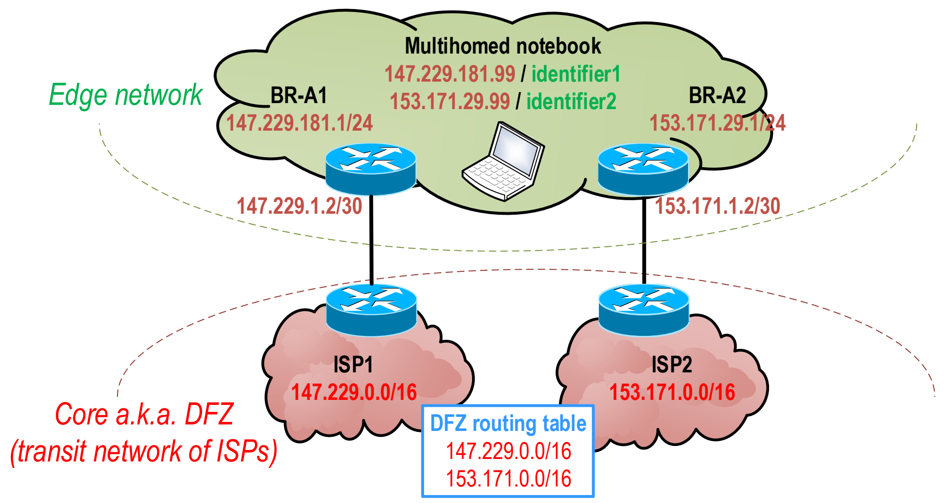


Fig. 13: Core-Edge Elimination solution

Down below in Tab. 1 is the summarized list of solution candidates.

Locator/Id Split Protocol (LISP)
LISP focuses on separation of locators and identifiers into two distinct address spaces using mapping and encapsulation on routers residing on the borders between those two spaces. Only locators are present in DFZ, thus are a possible subject of topological aggregation. With the separation of identifiers comes the ability to renumber cost effectively and get rid of more-specific prefixes in DFZ. With LISP, there is no need to change hosts or DFZ routers. LISP utilizes robust mapping system based on a pull model, where queries are data driven. However, it may introduce delay or even packet losses, when ID-to-loc mapping is being discovered. [43]
Host Identifier Protocol (HIP)
Network layer employs IP address as a locator, transport and application layer uses the identifier in the form of a cryptographic private-public key pair. Each host is responsible for generating this kind of pair. HIP makes use of DNS or distributed hash table (DHT) to obtain the identifier. [44]
Level 3 Multihoming Shim Protocol for IPv6 (Shim6)
Shim6 splits locator/id in a manner that IPv6 address field contains locator and extension header contains an identifier. Shim6 employs initial 4-way handshake with DNS lookup during which locator sets are exchanged. Keepalive mechanism tracks locator’s reachability. [45]
Routing Architecture for the Next Generation Internet (RANGI)
RANGI append one new layer between network and transport layer just as HIP. Hence, flows and connection are bound to host identifier instead of IP address that now serves as a locator. Unlike to HIP, RANGI host identifiers are hierarchical with an organized structure. [46]
Internet Vastly Improved Plumbing (Ivip)
Ivip works with map-and-encap principle as LISP. However, Ivip uses global mapping system instead of a hierarchical pull model. It maps only single locator to a given identifier and mappings are updated in real-time. Ivip employs direct IP-in-IP encapsulation. [47]
Hierarchical IPv4 Framework (hIPv4)
hIPv4 introduces an additional hierarchy of IPv4 address space by dividing it into area and endpoint locators. Both of them are inserted as optional fields into new shim header between network and transport layer. hIPv4 utilizes DNS for locator distribution. [48]
Name Overlay Service for Scalable Internet Routing (NOL)
NOL utilizes session layer and introduces new devices performing translation between public PA and private PI address namespace that prevent PI from entering DFZ. NOL leverages DNS to store name as a new kind of record. [49]
Global Locator, Local Locator, and Identifier Split (GLI-Split)
GLI-Split decouples addresses into global/local locators and static identifiers. It encodes two different namespaces (each one 64 bits or less) onto single IPv6 address. The communication with legacy Internet is without any proxies or stateful NAT. [50]

Tunneled Inter-Domain Routing (TIDR)
Loc/ID split is performed on BGP level as a new attribute. When a packet to identifier prefix is being routed, it is encapsulated into the tunnel. [51]
Identifier-Locator Network Protocol (ILNP)
ILNP decouples identity and locality inside IPv6 address field. Multiple locators might be used by a device simultaneously, whereas applications bind to a single identifier. ILNP needs DNS for backward/forward resolution of locators/identifiers to the domain name. [52]
Name-Based Sockets (NBS)
NBS are a new alternative for socket-based communication. Unlike nowadays BSD sockets that are bind to IP addresses, NBS are bind to domain names. Applications communicate using domain names where appropriate IP address selection is left on TCP/IP stack. [53]
A Practical Transit-Mapping Service (APT)
APT is a copy of LISP with operational restrictions that helps to more clear Loc/ID split design. APT uses periodical synchronization of the mapping system. Identifier to locator mappings are carried using new BGP attribute. [54]
Internet Routing Overlay Network with Routing and Addressing in Networks with Global Enterprise Recursion (IRON-RANGER)
IRON-RANGER utilizes own tunneling and path MTU discovery protocol called SEAL which redefines the semantics of some ICMP messages. IRON-RANGER is architecturally derived from ISATAP. [55]
Tunneling Route Reduction Protocol (TRRP)
TRRP interconnects border routers between core and edge using GRE. DNS lookup (above overloaded TXT resource record) helps to find tunnel endpoint. TRRP does not support multicast. [56]
Six/One Router (Six/One)
Six/One rewrites edge's local and core's remote addresses at the borders. Six/One takes advantage of special IPv6 extension header. [57]

Tab. 1: Candidates summary

3.3.3 Proposals Comparison

The following table Tab. 2 summarizes properties of each proposal above. Abbreviations used as columns names mean:

- *type* – Whether proposal employs map-and-encap ("M"), rewrite ("R"), host-based principle ("H") or it is something inherently different ("diff");
- *CE* – Whether proposal is Core-Edge Separation ("CES"), Core-Edge Elimination ("CEE") or generally different ("diff") solutions;
- *IPv = Internet Protocol version* – Which IP version does proposal supports ("v4/v6/v4v6");
- *RS = Routing Scalability* – Whether proposal reduces DFZ routing tables sizes ("yes/no");
- *DIL = Decoupling of Identification and Localization* – Whether proposal performs ("yes") locator/identifier split or not ("no");
- *MH = Multihoming* – Whether proposal supports better multihoming or not ("yes/no"), or it is supported conditionally together with utilization of multipath transport protocol ("cond");
- *Mob = Mobility* – Whether proposal supports seamless mobility or not ("yes/no"), or it is supported conditionally together with utilization of multipath transport protocol ("cond");
- *TE = Traffic Engineering* – Whether proposal contains TE by design or not ("yes/no"), or it is supported conditionally with utilization of multipath transport protocol ("cond");
- *Ren = Renumbering* – Whether proposal supports easier renumbering ("yes/no");
- *Dep = Deployability* – Whether proposal allows communication between upgraded and non-upgraded devices ("yes/no") or whether it is not applicable ("n/a").

Name	type	CE	IPv	RS	DIL	MH	Mob	TE	Ren	Dep
LISP	M	CES	v4v6	yes	yes	yes	yes	yes	yes	yes
HIP	H	CEE	v6	yes	yes	yes	yes	no	yes	no
SHIM6	H	CEE	v6	no	yes	yes	no	no	no	yes
RANGI	H	CEE	v6	yes	yes	yes	yes	yes	yes	yes
Ivip	M	CES	v4v6	yes	yes	yes	yes	yes	yes	yes
hIPv4	diff	diff	v4	yes	yes	cond	cond	cond	yes	no
NOL	R	diff	v4v6	yes	yes	yes	yes	yes	no	no
GLI-Split	R	CEE	v6	yes	yes	yes	yes	yes	yes	yes
TIDR	M	CES	v4v6	no	yes	yes	no	yes	yes	yes
ILNP	R	CEE	v6	yes	yes	yes	yes	yes	yes	yes
NBS	diff	CEE	v4v6	yes	yes	cond	cond	cond	no	no
APT	M	CES	v4v6	yes	yes	yes	yes	yes	yes	yes
IRON-RANGER	M	CES	v4v6	yes	yes	yes	yes	yes	yes	yes
TRRP	M	CES	v4v6	yes	no	yes	no	yes	no	yes
Six/One	R	CES	v6	yes	yes	yes	no	no	yes	yes
RINA	diff	diff	v4v6	yes	yes	yes	yes	yes	yes	yes

Tab. 2: Properties comparison of existing proposals

Main CES features are summarized in the following list and Fig. 14 below:

- Edge networks are separated from DFZ routing tables or are at least highly aggregated. Routing scalability is visible in direct proportion to how widely is CES solution adopted;
- CES benefits are available immediately to adopters – multihoming, inbound TE and if possible also mobility;
- Deployment of CES does not affect DFZ routers, but new devices on the border between core and edge are needed to interconnect this two address spaces together with mapping system;
- CES solutions do not require host stack, API or application changes;
- Tunneling and overlaying impose additional size overhead on fragments, thus introducing MTU concerns when employing CES.

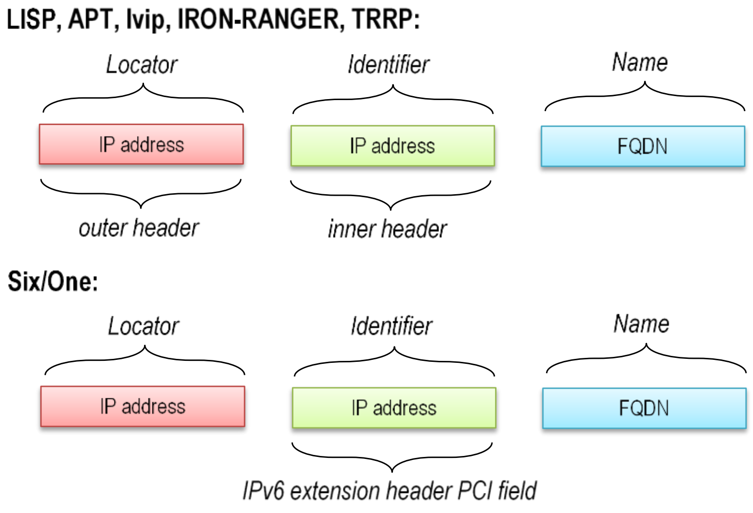


Fig. 14: CES types

Main CEE features are summarized in the list and Fig. 15 below:

- The most of CEE solutions separates locators and identifiers into two different *namespaces*.
- CEE benefits are visible and widely available to adopters only after majority of network migrate;
- Routing scalability is attained in a way that applications are no longer dependent on stable PI (or de-aggregated PA) addresses. Hence, PA addresses could be easily preferred and administratively more available than PI addresses.

- CEE host stack must determine which locator should use. Besides that, potential set of locators could be retrieved, thus implying resolving multihoming, inbound TE, and mobility issues;
- DFZ routers are not affected, and no additional tunneling devices are needed, however, a new infrastructure must be present to provide mapping between identifiers and locators;
- CEE solutions need host stack changes and applications augmentations;
- The most of CEE solutions do not support IPv4 and have some troubles with NAT so additionally clutches are needed.

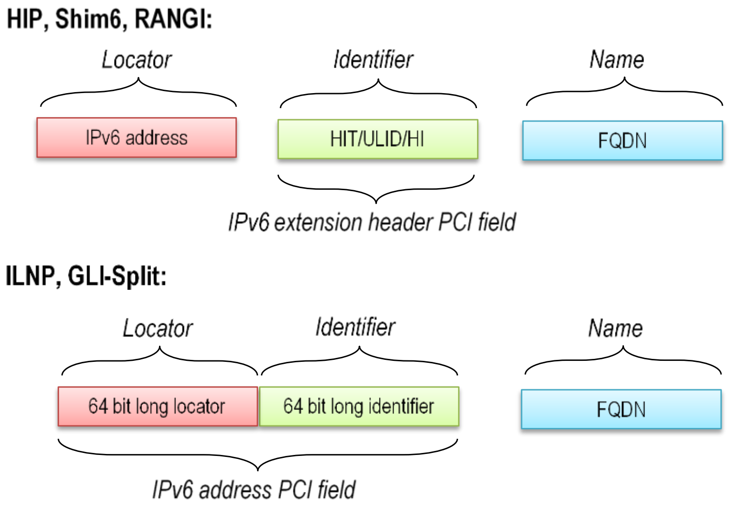


Fig. 15: CEE types

It is assumed that CES are easier for voluntarily adoption rather than CEE. On the one hand, the purpose of the routing system is to serve hosts. Hence, the goal is to make routing system more scalable with the help of CES solution that targets network, not hosts. On the other hand, CEE solutions are believed to lead to better final shape of the Internet, because of: a) routing should be as simple as possible without unnecessary tunneling clutches; b) utilization of IP address as identifier is a fundamentally wrong concept. One can say that CES is “network-centric” and CEE is “host-centric”. Unfortunately, no hybrid solution between CES and CEE does exist.

Both of them need a scalable mapping system. Nevertheless, CES mapping system is arguably more efficient because: a) CES lookups are needed only for initial communication towards a host inside edge network in opposite to CEE lookups that must be performed by senders and receivers for any newly established communications; b) CES mapping system is better designed for caching to alleviate unnecessary resolutions; c) it is unlikely that organizations already using PI addresses would downgrade for PA addresses.

3.4 Chapter Summary

This chapter offered theoretical background on naming, addressing, and routing issues. We postulated complete naming and addressing model based on a synthesis of respected works in this field. Before anything else, we outlined ideal solution properties and organized their goals according to the importance and beneficial effect. We enumerated existing feasible candidates and briefly mention their specifics. Then we compared and categorized all possible solutions.

The ideal solution should have following properties: a) provide complete naming architecture with one or more levels of addressing indirection, where employed addresses are *location dependent* but *route independent*; b) inherently support use-cases like network multihoming, device’s mobility, and owner regulated traffic engineering.

Drawing on overall results and findings, we decided to pursue LISP and RINA more closely to see whether they comply with postulated naming and addressing model and at the same time fit to achieve the most of the ideal solution goals.

4 Locator/ID Separation Protocol

꩜ –“*Perhaps it's impossible to wear an identity without becoming what you pretend to be.*” O.S.Card
꩜ *What is LISP? What components, messages and function does LISP employ?*
꩜ *Can we improve LISP’s operation?*

LISP is currently one of the most discussed CES solutions that could bring alleviation to “pain points” of nowadays Internet, such as mobility, multihoming, decoupling identity and locality. LISP works with map-and-encap principle benefiting from own mapping system to distribute information about identifier-locator pairs.

LISP development started after IAB Workshop in 2006, and it supposes to be the response dealing with major problems introduced in Subchapter 2.2. LISP should reduce DFZ routing table growth, stop prefix deaggregation, allow easier multihoming and mobility without the BGP and split locator and identifier namespaces. LISP should be deployed without any changes to hosts or DNS. It must support both IPv4 and IPv6 seamlessly. Moreover, it is agnostic to any network protocol (it could be used with future IPv7 or any new invention working on this layer). Transition mechanisms are part of LISP protocol standard. Thus, it supports communication with the legacy non-LISP world. Nevertheless, the enterprise is always skeptical and slow when adopting new technology. Hence, it is a great research challenge to investigate LISP features using modeling and simulation as the referential testbed tools producing meaningful outcomes.

In this chapter, we would like to dive into the LISP and explore its capabilities and limitations. The main goal of this chapter is: a) to provide an in-depth presentation of LISP; b) to illustrate known LISP issues; c) to propose improvements and implement them in the form of new simulation models for OMNeT++; and d) to evaluate the impact of suggested improvements.

4.1 Overview

Majority of this subchapter is based on RFC 6830-6834 [58], [59], [60], [61] that standardize LISP protocol and its interfaces as experimental.

The main idea behind LISP is to separate localization and identification. Following the example of GSM network could serve as an analogy for this. Cellphone identifier is a telephone number, and cell phone localizator is operator’s network, which connects the device. If somebody calls the number (“to identify”) then operator’s network searches for particular base transceiver station (“to localize”) with which cell phone is associated right now in order to establish the call. Whenever owner travels with cell phone abroad, cell phone changes also operator’s network (locator). However, callers are still using the same number (identifier) to reach owner despite the fact that locality has changed.

LISP accomplishes similar behavior by splitting the IP address into two *namespaces*:

- **Routing Locator (RLOC)** *namespace* where addresses fulfill their localization purposes by telling where is device connected to the network (red cloud on Fig. 16);
- **Endpoint Identifier (EID)** *namespace* where each device has a unique name that identifies it from each other (green cloud on Fig. 16).

Also a **non-LISP** *namespace* exists (and probably always will exist), where direct LISP communication is (even intentionally) not supported (blue cloud on Fig. 16). Apart from *namespaces* also exist: a) specialized routers performing map-and-encap that interconnects different *namespaces*; b) dedicated devices maintaining mapping system; and c) proxy routers allowing communication between LISP and the non-LISP world.

4.1.1 Tunneling

A LISP mapping system performs lookups to retrieve a set of RLOCs for a given EID. Tunnel routers between *namespaces* utilize these EID-to-RLOC mappings to perform map-and-encapsulation.

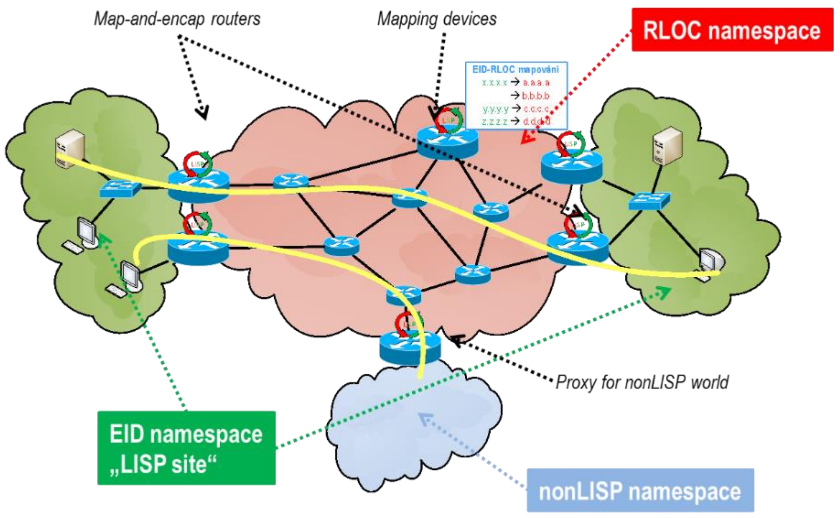


Fig. 16: Basic LISP scheme

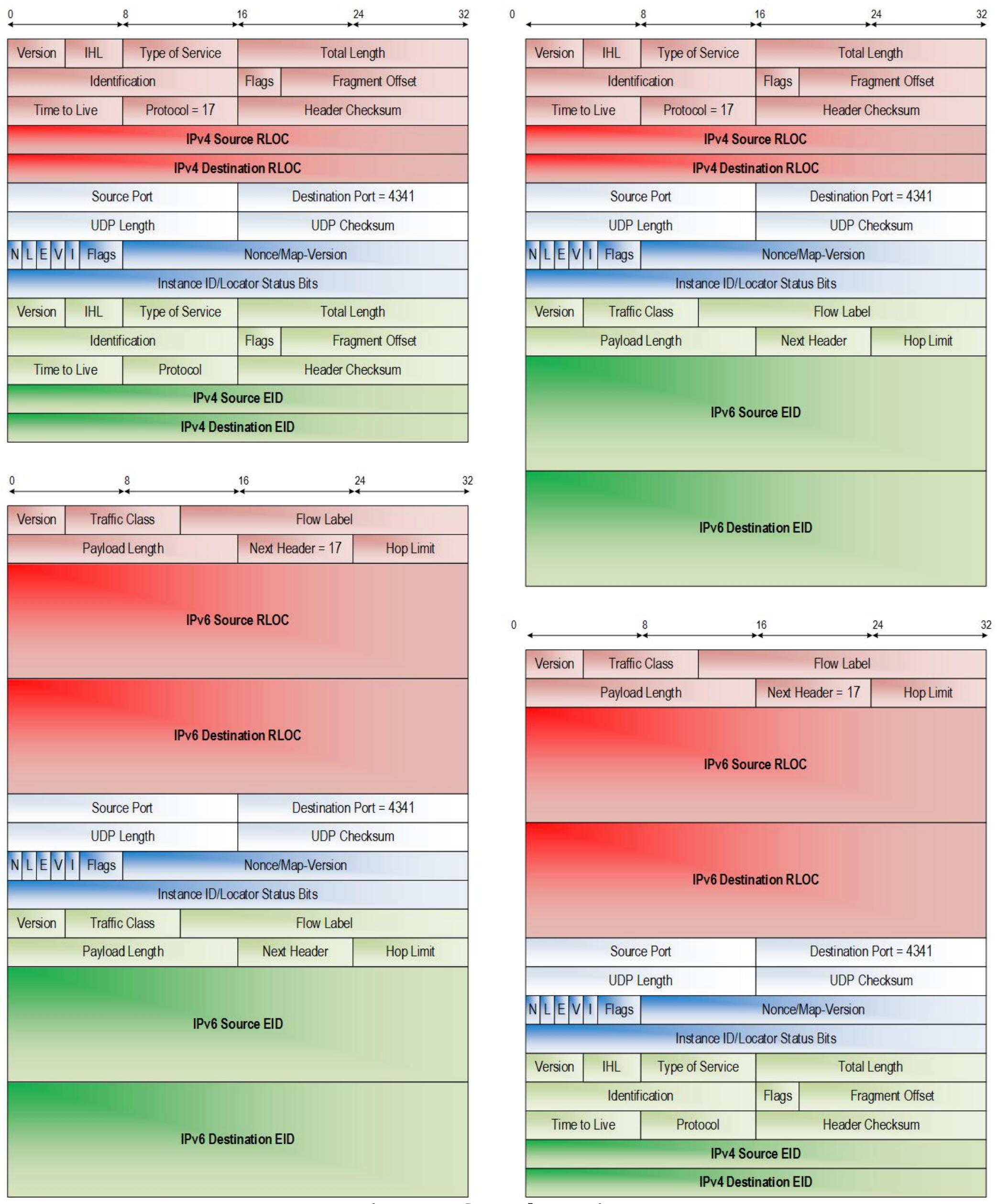

0 8 16 24 32
Version | IHL | Type of Service | Total Length
Identification | Flags | Fragment Offset
Time to Live | Protocol = 17 | Header Checksum
IPv4 Source RLOC
IPv4 Destination RLOC
Source Port | Destination Port = 4341
UDP Length | UDP Checksum
N | L | E | V | I | Flags | Nonce/Map-Version
Instance ID/Locator Status Bits
Version | IHL | Type of Service | Total Length
Identification | Flags | Fragment Offset
Time to Live | Protocol | Header Checksum
IPv4 Source EID
IPv4 Destination EID
0 8 16 24 32
Version | IHL | Type of Service | Total Length
Identification | Flags | Fragment Offset
Time to Live | Protocol = 17 | Header Checksum
IPv4 Source RLOC
IPv4 Destination RLOC
Source Port | Destination Port = 4341
UDP Length | UDP Checksum
N | L | E | V | I | Flags | Nonce/Map-Version
Instance ID/Locator Status Bits
Version | Traffic Class | Flow Label
Payload Length | Next Header | Hop Limit
IPv6 Source EID
IPv6 Destination EID
0 8 16 24 32
Version | Traffic Class | Flow Label
Payload Length | Next Header = 17 | Hop Limit
IPv6 Source RLOC
IPv6 Destination RLOC
Source Port | Destination Port = 4341
UDP Length | UDP Checksum
N | L | E | V | I | Flags | Nonce/Map-Version
Instance ID/Locator Status Bits
Version | Traffic Class | Flow Label
Payload Length | Next Header | Hop Limit
IPv6 Source EID
IPv6 Destination EID
0 8 16 24 32
Version | Traffic Class | Flow Label
Payload Length | Next Header = 17 | Hop Limit
IPv6 Source RLOC
IPv6 Destination RLOC
Source Port | Destination Port = 4341
UDP Length | UDP Checksum
N | L | E | V | I | Flags | Nonce/Map-Version
Instance ID/Locator Status Bits
Version | IHL | Type of Service | Total Length
Identification | Flags | Fragment Offset
Time to Live | Protocol | Header Checksum
IPv4 Source EID
IPv4 Destination EID

Fig. 17: LISP packet variants

The original (inner) header (with EIDs as addresses) is encapsulated by a new (outer) header (with RLOCs as addresses), which is appended when crossing borders from EID to RLOC *namespace*. Whenever a packet is crossing back from RLOC to EID *namespace*, the packet is decapsulated by stripping outer header off. LISP supports both IPv4 and IPv6. Moreover, LISP is agnostic to address family thus it can seamlessly work with any upcoming network protocol. Transition mechanisms are part of the protocol standard. Hence, LISP supports communication with the legacy non-LISP world. LISP places between inner and outer header additional PCI in the form of UDP header succeeded by LISP header. LISP uses reserved port numbers – 4341 for data and 4342 for signalization. Currently, any combination of IP headers is supported – IPv4 outer / IPv4 inner, IPv4 outer / IPv6 inner, IPv6 outer / IPv4 inner, IPv6 outer / IPv6 inner. However, the map-and-encap principle is so generic that LISP could inherently support any network layer protocol. Fig. 17 depicts all variants of LISP packets.

Basic components are **Ingress Tunnel Router** (ITR) and **Egress Tunnel Router** (ETR). Both are border devices between EID and RLOC space; the only difference is in which direction they operate. The single device could be either ITR-only or ETR-only or ITR and ETR at the same time (thus abbreviation **xTR**).

ITR is the exit point from EID space (a.k.a. **LISP site**) to RLOC space, which encapsulates the original packet. This process may consist of querying mapping system followed by updating local **map-cache** of recently used mappings. Map-cache improves the performance of the system (i.e., EID-to-RLOC mapping pairs are stored for a limited time to reduce signalization overhead).

ETR is the exit from RLOC space to EID space that decapsulates original header. Outer header, auxiliary UDP, and LISP headers are stripped off. ETR is also announcing all LISP sites (their EID addresses) and by which RLOCs they are accessible.

If we inspect structure of LISP packet somewhere in RLOC space then:

- Inner header source IP = sender’s EID address;
- Inner header destination IP = receiver’s EID address;
- Outer header source IP = ITR’s RLOC address;
- Outer header destination IP = ETR’s RLOC address.

4.1.2 Mapping System

Before moving to LISP mapping system concretely, let us discuss how those things are handled theoretically. Any Internet mapping system is nothing else than the huge distributed database. Simple mapping information is represented in a single database record.

LISP mapping system is primarily employing two components – **Map Resolver (MR)** and **Map Server (MS)**. Looking for EID-to-RLOC mapping is an analogous process as DNS name resolution (see Fig. 18). In the case of DNS, the host asks its DNS resolver (configured within OS) which IP address belongs to a given FQDN. DNS server responds with a cached answer or delegates the question recursively or iteratively to another DNS server according to the name hierarchy. In the case of LISP, querier is ITR that needs to find out which RLOCs could be used to reach a given EID. ITR has preconfigured MR, which is bothered each time mapping is needed.

Queries performing EID-to-RLOC mapping are data-driven. This behavior means that a new data transfer between LISP sites may require a mapping lookup, which causes that data dispatch is stopped until a mapping is retrieved. This behavior allows LISP to operate a decentralized database of EID-to-RLOC mappings. Replication of whole (potentially large-scale) database is unnecessary because mappings are accessed on-demand, just like as in DNS a host does not need to know complete domain database. Tunnel routers maintain map-cache of recently used mappings to improve the performance of the system.

Following list contains all LISP mapping signalization messages with their brief description. They are without inner header – just the outer header, followed by UDP header (with source and destination ports set on 4342), and followed by appropriate LISP message header.

- *LISP Map-Register* – Each ETR announces as authority one or more LISP site(s) to the MS with this message. Each registration contains authentication data and the list of mappings and their properties;

- *LISP Map-Notify* – UDP cannot guarantee message delivery. MS may optionally (when the proper bit is set) confirm reception of *LISP Map-Register* with this message;
- *LISP Map-Request* – ITR generates this request whenever it needs to discover current EID-to-RLOC mapping and sends it preconfigured MR;
- *LISP Map-Reply* – This is solicited a response from the mapping system to a previous request and contains all RLOCs to a certain EID together with their attributes. Each ITR has its map-cache where reply information is stored for a limited time and used locally to reduce signalization overhead of mapping system. Moreover, mapping system generates *LISP Negative Map-Reply* as a response whenever given identifier is not the EID, and thus proxy routing for non-native LISP communication must occur.

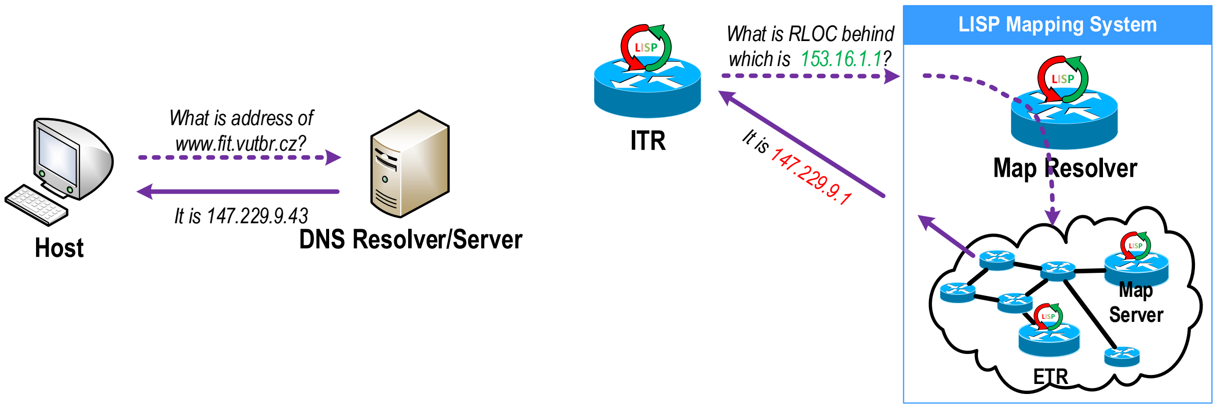


Fig. 18: Comparison between DNS and LISP mapping system

MR processes ITR's *LISP Map-Requests*. Either MR responds with *LISP Negative Map-Reply* if queried address is from a non-LISP world (not EID), or *LISP Map-Requests* is delegated further into a mapping system to appropriate MS.

Every MS maintains **mapping database** of LISP sites that are advertised by *LISP Map-Register* messages. If MS receives *LISP Map-Request* then: either a) MS responds directly to querying ITR (it is allowed to do that because MS has all the necessary information in its mapping database); or b) MS forwards request towards designated ETR that is successfully registered to MS for target EID.

Each RLOC is accompanied by two attributes – priority and weight. **Priority** (one-byte long value in the range from 0 to 255) expresses each RLOC preference. The locator with the lowest priority is preferred and is going to be used as the outer header address. Priority value 255 means that the locator must not be used for traffic forwarding. Incoming communication may be load-balanced based on the **weight** value (in the range from 0 to 100) between multiple RLOCs sharing the same priority. Zero weight means that RLOC usage for load-balancing depends on ITR preferences.

xTRs perform **RLOC probing** (checking of non-local locator liveness) in order to always use current information. RLOC probing is done with the help of special variant *LISP Map-Request* and *LISP Map-Reply* messages (with the appropriate bit set on). Let us called them *LISP Map-Request Probe* and *LISP Map-Reply Probe*.

4.1.3 Coexistence between LISP and Non-LISP

Flag Day is not an option in case of migration to LISP just as in the case of IPv6. Moreover, there will always be networks that do not intend to deploy LISP or where LISP deployment is not beneficiary or possible. Special devices are needed to interconnect LISP and a non-LISP world where IP address locality and identity are not decoupled. Communication between those two worlds differs according to the direction, how IP addresses are interpreted during routing procedure and what issues are connected with it:

- *non-LISP* ⇨ *LISP* – Hosts and routers do not know anything about loc/id split. Hence, EIDs are considered as ordinary addresses and natively routed to "EID network entry point";
- *LISP* ⇨ *non-LISP* – ITR must recognize that the destination address is not EID. Hence, there are no RLOCs associated with it. The packet is then delivered to "LISP world exit point".

Two approaches are proposed for LISP/non-LISP coexistence purposes: a) address translation; b) proxies providing ITR and ETR roles (both briefly documented bellow and in [59]).

No matter whether a) or b) is used, the both of them supports Day 1 benefits so that the number of adopters does not determine overall functionality and quality of LISP deployment. Therefore, site profits from LISP (i.e. easier mobility or multihoming) immediately after migration. Full control over inbound TE is the most noticeable adoption gain because of priority and weight attributes that are mandatory to follow by any LISP implementation. Compare LISP load-balancing (according to priority/weight – inherent parts of LISP protocol design) and BGP policies that should accomplish the same goal. Unfortunately, BPG policies cannot be enforced and are prone to reconfiguration when traversing ASes.

4.2 State-of-the-Art

The research community has limited options how to observe and expand LISP features in a safe environment of simulator where different scenarios could be easily scheduled and verified later.

One of a few attempts is CoreSim developed by Coras et al. [62]. It is written in Perl, and it allows predict ITR and MS behavior at a macro-scale level using traffic traces, BGP data, and latency estimations. However, CoreSim estimations use rather a general mathematical model taking into account only the distance [63]. Currently, limited LISP implementation exists authored by Hoefling et al. [64] to support LISP MobileNode NAT traversal [65]. However, it is intended for outdated INET-20100323 and OMNeT++ 4.0. Previously, LISP map-cache performance have been evaluated employing high-level simulation that is not taking into account protocol implementation specifics [66].

Among additional goals of this thesis is to provide the community with a variety of simulation models supporting up-to-date version of LISP protocol.

4.3 Contribution

LISP architectural implications are discussed in IETF draft [67] followed by companion paper [68]. Previous papers outline and discuss two major issues for LISP threatening its scalability – Site-Based State Synchronization Problem and Locator Path Liveness Problem.

Site-Based Synchronization Problem occurs whenever EID-to-RLOC mappings (including locator statuses) may need to be shared among nodes. Remember that LISP mapping queries are data-driven. There is no need to rediscover mapping for the same data traffic by one xTR if this mapping is already known to other site's xTRs. Sharing of mapping improves routing of packets in case of asymmetrical traffic flows. Imagine that traffic is leaving the site via two xTRs – one is actively dispatching all traffic, another is backing up its functionality. Map-cache on active xTR is populated with records whereabouts map-cache on backup has no mapping state. Whenever traffic shifts from active path to backup path, former backup xTR experiences map-cache misses

Locator Path Liveness Problem is formulated by a question whether given set of source locators and a set of destination locators, can bi-directional connectivity be determined between the $\langle \mathrm{srcRLOC}, \mathrm{dstRLOC} \rangle$ address pairs? Locator Path Liveness Problem is present not only in LISP but its variants also apply to other candidates like HIP, SHIM6 or IRON-RANGER. In the case of LISP, if ITR chooses destination RLOC, which is not reachable, then traffic is discarded somewhere along the path towards destination LISP site.

This subchapter introduces two proposed improvements targeting some of the issues from previously mentioned papers that increase LISP performance – map-cache synchronization and merged RLOC probing. In order to evaluate contribution, we developed brand new OMNeT++ simulation modules for LISP and also for Virtual Router Redundancy Protocol that is being deployed simultaneously on ITR.

4.3.1 Map-Cache Synchronization

Assume multiple redundant routers are acting as first hops in the high-availability scenario like in Fig. 19. Those routers are simultaneously clustered into VRRP groups and act as LISP's xTRs – they run LISP and VRRP at the same time.

The performance of map-and-encap depends on the fact whether xTR's map-cache contains valid EID-to-RLOC mapping or not. Dispatched data traffic drives map-cache record creation. If map-cache misses the mapping, then, a mapping system needs to be asked, and initiating data traffic is meantime dropped. This fact is illustrated in Fig. 19 for EID address y.y.y.y. On the one hand, packets (with y.y.y.y as destination) can traverse *ITR1* without any problem (locator c.c.c.c is present in map-cache). On the other hand, same packets are discarded on *ITR2*, which misses the mapping. Packet dropping is a valid step as long as the mapping is not discovered because map-and-encap cannot occur without proper information. The rationale behind this behavior is the same as in the case of ARP throttling [11], where any triggering traffic should be discarded to protect control-plane processing and prevent superfluously recurrent mapping system queries.

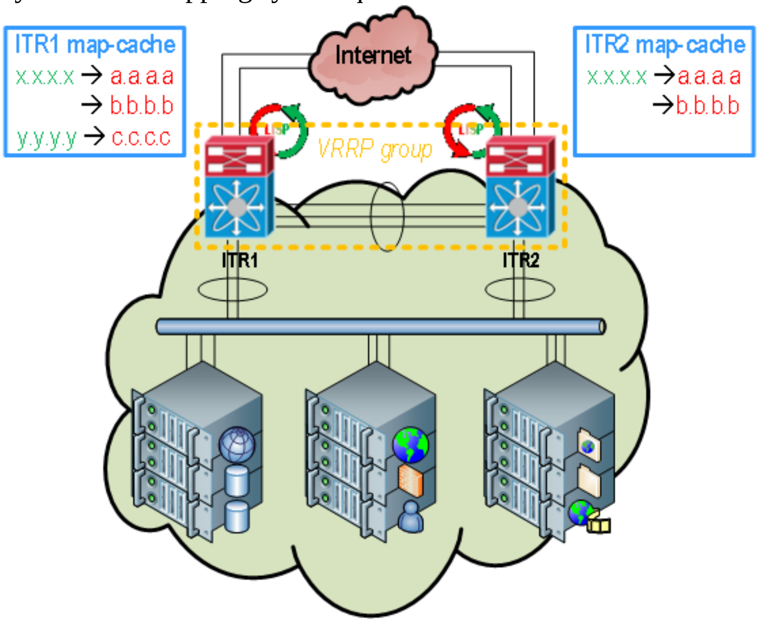


Fig. 19: Site-Based State Synchronization Problem illustration

Each xTR has its map-cache, and its content may differ even within the same LISP site because different traffic may initialize various map-cache entries. Hence, xTRs can easily experience severe packet drops and LISP control message storms due to the map-cache misses when Master change occurs within VRRP group.

Previous is known as **Site-Based State Synchronization Problem**. If we have two or more redundant xTRs, then we want to reduce packet drops as much as possible in case there is a traffic shift from an active to a backup device. xTR outage leads to the off-site signalization storm (lots of LISP Map-Request/Reply messages being exchanged) and dispatching delay for ordinary traffic.

This problem is described as the one of LISP weak-points in [69] and theoretically investigated in [70]. The viable solution would be to provide map-cache content synchronization that should minimize map-cache misses upon failure. Inspired by that, we present our solution addressing this problem.

We have decided to implement it as a technique maintaining synchronized map-caches within a predefined **synchronization set (SS)** of ITRs. Any solicited *LISP Map-Reply* triggers synchronization process among SS members.

SS members are identified and reached using the IP address. Following strategies might be used when choosing appropriate SS member address:

- SS address comes from non-LISP world – Either IP address should be loopback or address of dedicated interconnection shared by all SS members. In the first case, unique device loopbacks need to employ additional routing. In the second case, the additional port for the dedicated connection is seldom available. Also, tracking of SS member needs additional LISP control plane updates;

- SS address comes from LISP world:
    - SS address is RLOC – SS membership is bound to the operability of a given RLOC interface, but this has negative implications for the situation, where xTR has more than one RLOC available. Although, it is easy to track SS member status using return value of RLOC probing;
    - SS address is EID – The best option reflecting LISP's ideology. EID as SS address should be reachable via direct routing (xTRs share common EID segment) or unless all RLOCs to this EID are down (which could be also used to track peer synchronization status).

Each record in the map-cache is equipped with a **time-to-live (TTL)** parameter. TTL expresses how long the record is considered to be valid and usable for map-and-encap. By default, every record uses the same initial TTL value. Map-caches within SS must maintain the same TTL on shared records; otherwise a loss of synchronization might occur (on some ITRs, identical records could expire because of no demand for traffic).

Either SS membership may be completely stateless, or SS member may maintain a state of its synchronization peers. The stateful approach allows sending of partial synchronization updates. We have implemented two modes of synchronization reflecting previous observation:

1) *Naïve* – The whole content of map-cache is transferred to SS. All mappings are then updated according to the new content and TTLs are reset. This approach works fine, but it obviously introduces significant transfer overheads;
2) *Smart* – Only record that caused synchronization is transferred. However, peer synchronization status have to be employed to deal with the situation when SS member goes back up and completely lacks any mapping. At that time, a whole set of map-cache content must be sent (not just a partial update). Moreover, we bound this mode with the following policy. When TTL expires, the ITR must check record usage during the last minute (one minute should be a period long enough to detect ongoing communication). If the mapping has not been used (based on the last lookup time of cache record), then it is removed from the cache. Otherwise, its state is refreshed by query followed by synchronization.

Both approaches guarantee that devices within SS could forward rerouted LISP data traffic without packet loss or interruption because they share the same content as ITR's map-cache of malfunctioned former Master.

Synchronization itself is done with the help of two new LISP messages – one carries synchronization data, another optionally acknowledges successful synchronization:

- *LISP CacheSync* – It contains map-cache records, which are being synchronized, and authentication data, which protect SS members from spoofed messages;
- *LISP CacheSync Ack(nowledgement)* – Because LISP leverages UDP, it cannot guarantee message delivery. However, we decided to employ the same principle as for *LISP Map-Register* and *LISP Map-Notify*. Hence, *LISP CacheSync* delivery may be optionally confirmed by echoing back *LISP CacheSync Ack* message.

Message structure of *LISP CacheSync* is depicted in Fig. 21 and *LISP CacheSync Ack* in Fig. 22. Notable differences when comparing to *LISP CacheSync/(Ack)* with the structure of *LISP Map-Register/Notify* are:

- Both messages also include new Type values – *LISP CacheSync* is 5, *LISP CacheSync Acknowledge* is 6;
- *LISP CacheSync* header contains C flag. When C flag is set on, then synchronization acknowledgment is requested by a sender. Receiver (i.e., SS member) must reply with *LISP CacheSync Ack* containing all the map-cache records that have been successfully processed. *LISP CacheSync* message is resent after the acknowledgment awaiting timeout (by default with cumulative value $2^{\mathrm{numOfRetries}}$);

- There is no need for A flag in Cache Record and L and p flags in RLoc (for details about flag meanings, please see [58]);
- As in the case of *LISP Map-Register/Notify*, *LISP CacheSync/(Ack)* mandatorily contain nonce and authentication using HMAC to avoid spoofing of false unsolicited cache synchronization information.

The diagram in Fig. 20 depicts FSM implementing map-cache synchronization where transitions are denoted with "input / action" labels. Our solution provides a clean-slate way how to alter the content of the map-cache reliably. Nevertheless, others might try to leverage options already available in LISP. Unfortunately, each one has some disadvantage.

The first approach is to alter existing *LISP Map-Requests* by forcing included map-reply record field to contain more than one record. However, this approach is unreliable because it lacks acknowledgment scheme and cannot solve all following wrong goings. What if receiver side does not recognize this option inside *LISP Map-Request*? What if *LISP Map-Request* did not reach receiver? What if the receiver wants to process only part of synchronization information? What if SS-members need to synchronize map-cache when the condition for sending *LISP Map-Request* is not met?

The second approach is that LISP already contains an on-demand renewal of mapping information called **Solicit-Map-Request (SMR)**. SMR is a mechanism how ETRs may rate-limit requests and notify ITRs about mapping change. When mapping changes, ETR starts to send *LISP Map-Request* (with the SMR-bit set on) messages to ITRs with which it recently exchanged data. Then, ITR generates SMR-invoked *LISP Map-Request* to discover new mapping. If we want to use SMR to push new mappings into ITR's map-cache, then the best way seems to be extending the functionality of MR (see [70]). However, this approach yields significant off-site signalization overhead.

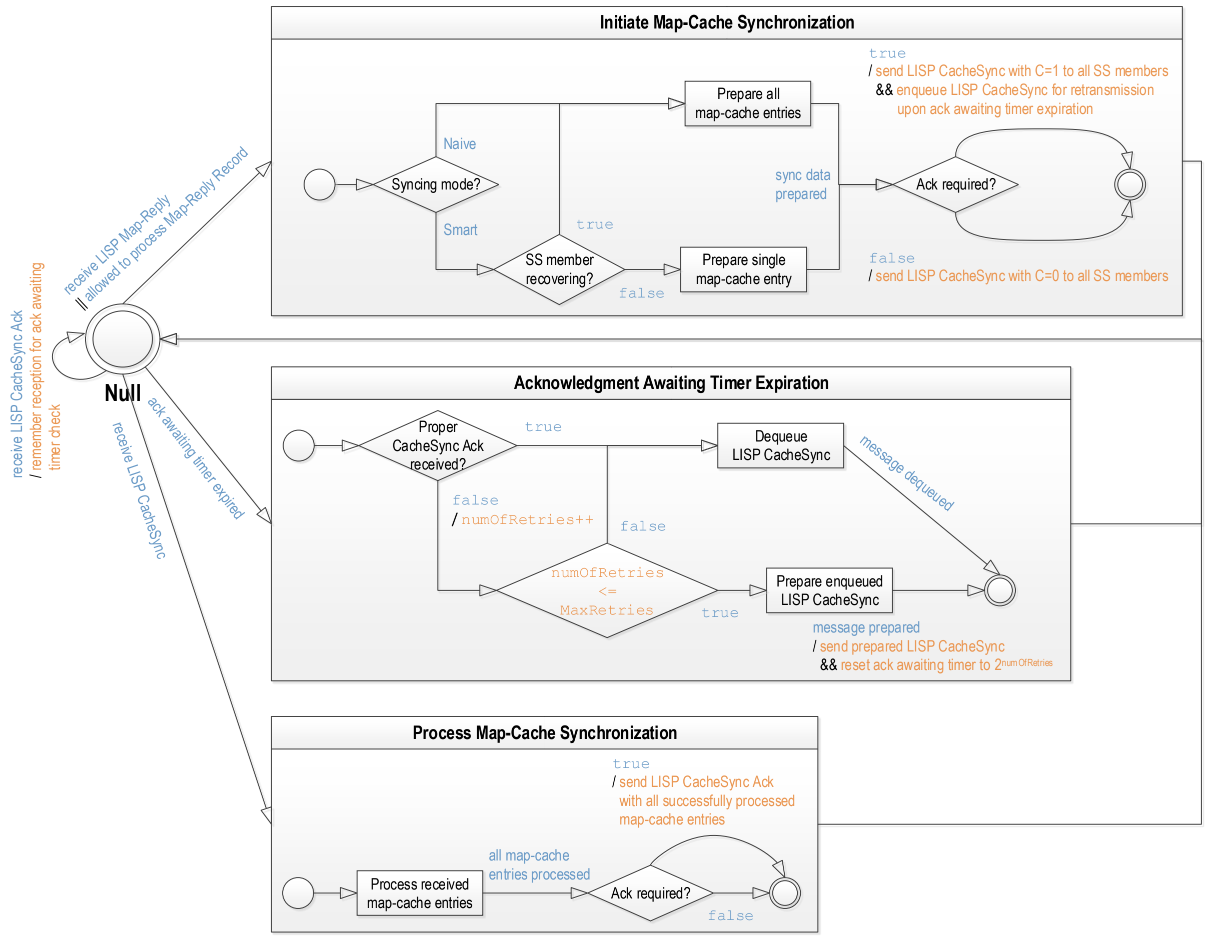

Initiate Map-Cache Synchronization
Syncing mode?
Naive
Smart
Prepare all map-cache entries
SS member recovering?
true
false
Prepare single map-cache entry
sync data prepared
Ack required?
true
/ send LISP CacheSync with C=1 to all SS members
&& enqueue LISP CacheSync for retransmission upon ack awaiting timer expiration
false
/ send LISP CacheSync with C=0 to all SS members
Null
receive LISP Map-Reply
// allowed to process Map-Reply Record
receive LISP CacheSync Ack
/ remember reception for ack awaiting timer check
ack awaiting timer expired
receive LISP CacheSync
Acknowledgment Awaiting Timer Expiration
Proper CacheSync Ack received?
true
Dequeue LISP CacheSync
message dequeued
false
/ numOfRetries++
false
numOfRetries <= MaxRetries
true
Prepare enqueued LISP CacheSync
message prepared
/ send prepared LISP CacheSync
&& reset ack awaiting timer to 2^numOfRetries
Process Map-Cache Synchronization
Process received map-cache entries
all map-cache entries processed
Ack required?
true
/ send LISP CacheSync Ack with all successfully processed map-cache entries
false

Fig. 20: Map-cache synchronization operation

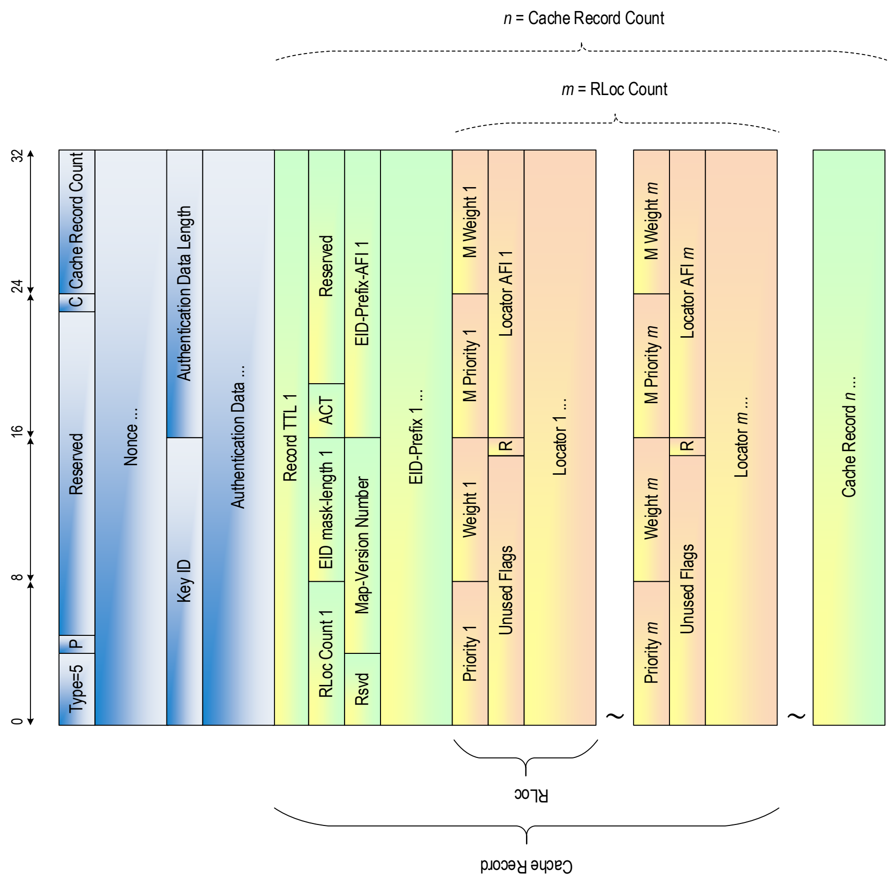


Fig. 21: LISP CacheSync message format

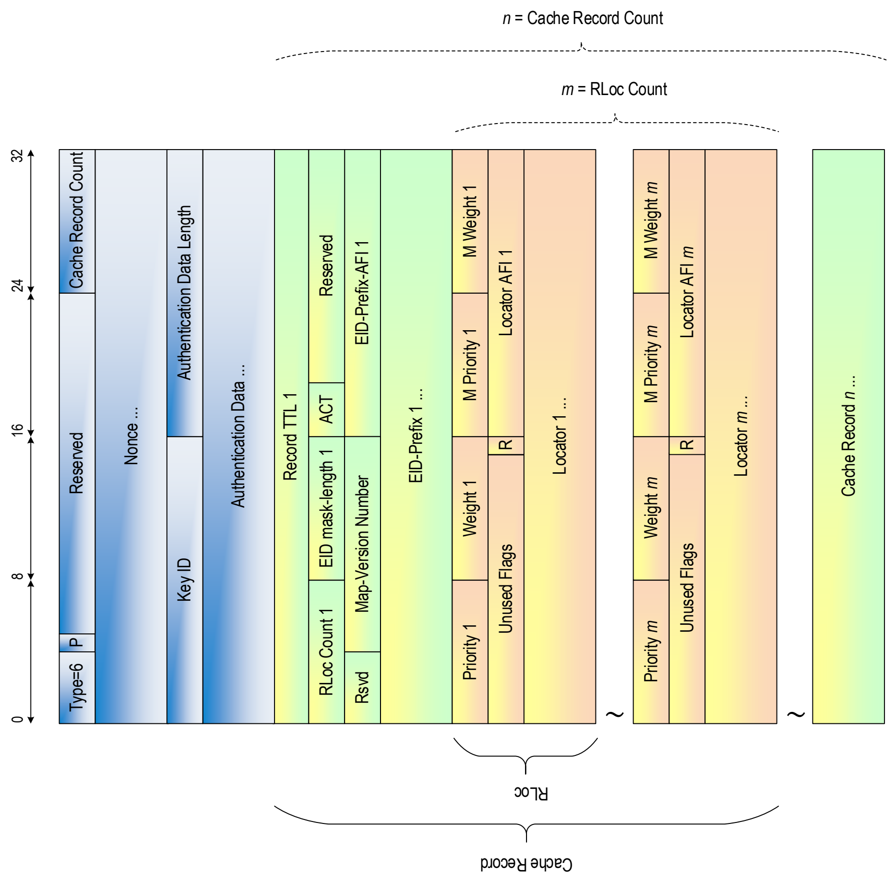


Fig. 22: LISP CacheSync Acknowledgment message format

4.3.2 Merged RLOC Probing

Locator Path Liveness Problem concerns whether a destination locator is reachable via particular source locator or, in other words, whether bi-directional connectivity exists between a given pair of locators. Problem relevant to LISP is depicted in Fig. 23 where *xTR-A1* asks for *Site B* locators. In this case, two locators are available (1.0.0.1 and 2.0.0.1). *xTR-A1* chooses the second one as a destination address for packets. If the link between *ISP1* and *ISP2* goes (un)intentionally down, 2.0.0.1 is not reachable anymore, and *xTR-A1* must somehow find out this fact.

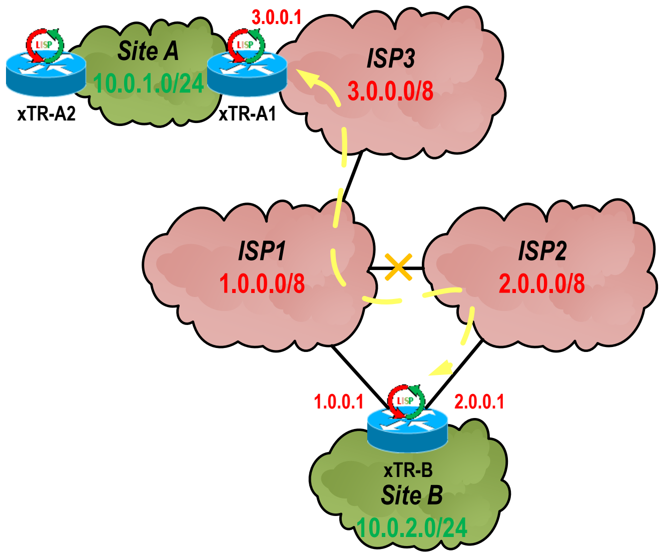


Fig. 23: Locator Path Liveness Problem illustration

Locator Path Liveness detection (checking whether RLOC is reachable or not) does not scale very well in large networks because the reachability of every destination locator must be probed against every source locator of a given device. Complexity of such a task is generally $O(n \times m)$, where n is a number of source and m a number of destination locators. However, instead of brute-force probing some hints might be used to mitigate (but not to avoid) such complexity, e.g. piggybacking, timeouts, existence of underlying routing, positive feedback from protocol control messages or other protocols.

To make Locator Path Liveness Problem even more complicated, let us imagine a situation when LISP site has two or more ITRs with different destination locator reachability. One ITR has connectivity, and another has not (e.g. *xTR-A1* and *xTR-A2* on Fig. 23). Hence, all packets processed by that ITR are going to be discarded somewhere in the network. Unfortunately, neither IGP responsible for routing the packet to faulty ITR nor hosts have capabilities to detect this issue from their internal point of view.

In order to find a remedy for this problem, we focused on the behavior of Cisco referential implementations and their RLOC-probing algorithm checking locator reachability. ITR is probing assigned locators for each configured EID. This behavior is in compliance with [58] but it leads to repeated check of the same locator multiple times, which represents scalability issue in larger networks.

We decided to decrease protocol overhead by merging EIDs to check locator liveness with a single RLOC probe that we call **merged RLOC probing**.

The *simple* but rather a trivial approach would be to make the following assumption: "If the same locator is reachable for one EID then it would also be reachable for other EID." Hence, the router can generate only single RLOC probe during one liveness checking period. If it receives positive *LISP Map-Reply Probe*, it may consider probed locator as alive for all EIDs in map-cache that are using it. More *sophisticated* approach is to:

1) On sender, check liveness of a given locator with a single *LISP Map-Request Probe* containing one or more query records. Each query record specifies cached EID that uses probed RLOC;
2) On receiver, respond with *LISP Map-Reply Probe* that includes locator status updates for all queried EIDs contained in request (or only subset of those EIDs that are in up state);
3) Back on the sender, refresh locator status of relevant EIDs in map-cache according to answer(s) in reply.

Above described mechanism is compatible with RFC description and does not need any protocol extensions. Yet, it preserves the accuracy of Cisco's RLOC probing algorithm but with only single RLOC probe exchanged. We have integrated all above described algorithms – *Cisco's*, *Simple* and *Sophisticated* – in our LISP simulation module.

4.3.3 Results

This section presents results of evaluation of newly implemented mechanisms. Each measured phenomenon has its subsection with dedicated network graph and scenario. The goal of this subchapter is to show: a) the impact of synchronization on a packet drop rate (and a number of map-cache misses) and to enumerate the burden of deploying it on control plane; and b) the impact of merged RLOC probing on control plane processing.

Impact of Map-Cache Synchronization

We prepared simulation network that contains a LISP site (network EID 192.168.1.0/24 reachable via two RLOCs 11.0.0.1 and 12.0.0.1) with two routers (*xTR1* and *xTR2*), which provide highly-available VRRP default gateway (192.168.1.254) for two hosts interconnected by switch *SW*. *Host1* and *Host2* are pinging IPv4 EIDs (172.16.[0-19].0/24) randomly thus generating traffic that triggers LISP mapping system queries. All routing is done statically. Hence, there is no need to employ routing protocol on *Core* router. We prepared special xTR called *xTR_Responder1* that: a) registers destination EIDs to *MRMS*; and b) responds to hosts ICMP messages. The whole network graph is depicted in Fig. 24. Also this scenario (named "LispSyncTest") is located in `/examples/ansa/lispSyncTest` folder of available source codes.

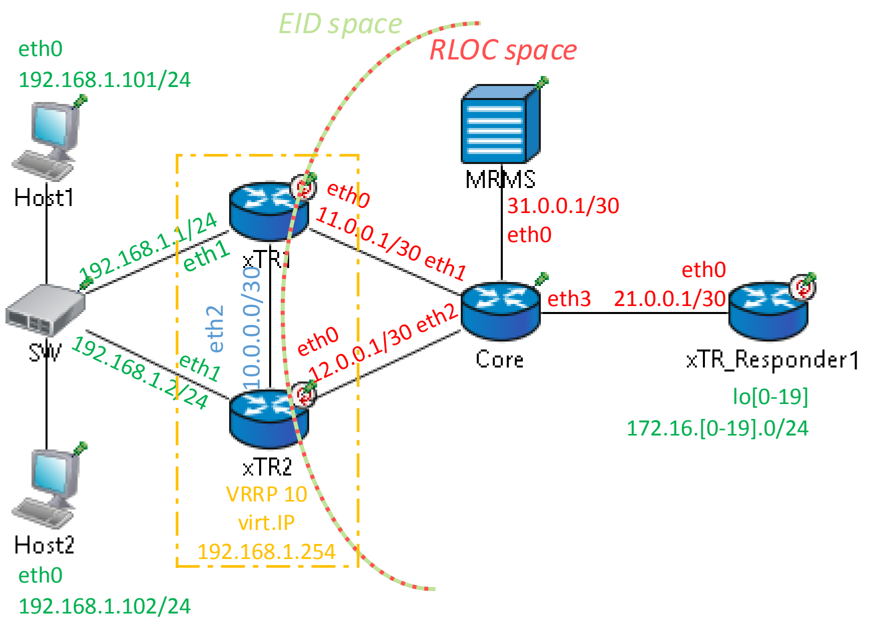


Fig. 24: LISP testing network for Map-Cache synchronization

The testing scenario is focused on cache misses due to the missing mapping rather than expired ones because of default TTL value (1 day). Five minutes time slot with the single VRRP Master outage is the simplest illustration of how to compare the impact of map-cache synchronization. During the outage, all xTR1's interfaces shut down (i.e., they are physically disconnected from the network). Yet, the xTR1's control plane is operational (generating scheduled LISP messages, which are not delivered).

We scheduled following phases for the test run focusing on map-cache synchronization:

#1) At first, all xTRs register their EIDs. In the case of *xTR_Responder1*, EID space is modeled with the help of loopback interfaces – twenty of them ranging with addresses from 172.16.0.0/24 to 172.16.19.0/24 reachable via single RLOC 21.0.0.1. In case of xTR1 and xTR2, EID 192.168.1.0/24 is reachable via two RLOCs 11.0.0.1 and 12.0.0.1;

#2) *xTR1* and *xTR2* form VRRP group with VID 10 and virtual address 192.168.1.254, which is used by *Host1* and *Host2* as default-gateway. *xTR1* is Master because of higher priority (*xTR1* has 150, *xTR2* only 100) as long as it is operational.
#3) *Host1* starts pinging ten random EIDs in the range from 172.16.0.0/24 to 172.16.9.0/24. Because EIDs are chosen randomly, they may be duplicate. Each first ICMP packet causes mapping query and is dropped.
#4) Then right before a new LISP registration (at `t=119s`), *xTR1* failure occurs. Hosts traffic is diverted to a new VRRP Master, which is *xTR2*.
#5) After phase 4), also *Host2* starts to ping ten random EIDs from 172.16.10.0/24 to 172.16.19.0/24. Same duplicity rule as in 3) applies.
#6) *xTR1* recovers from the outage at `t=235s` and once again all hosts traffic goes through it.

Depending on the map-cache synchronization type, additional map-cache misses might occur. *xTR1* and *xTR2* synchronized themselves via their RLOCs (11.0.0.1 for *xTR1* and 12.0.0.1 for *xTR2*).

The scenario has been tested with three simulation configurations, which we can divide according to the used map-cache synchronization technique: α) no synchronization at all (default LISP behavior); β) *naïve* mode; and γ) *smart* mode. Impact on map-cache is summarized in Tab. 3 for all previously mentioned different configuration runs. Fewer map-cache misses are considered better.

We do not employ LISP synchronization acknowledgment scheme for β/γ-runs, the impact of acks is analyzed later. The scenario offers testing of all three kinds of addressed for SS member identification – e.g., nonLISP with 10.0.0.0/30; RLOC with 11.0.0.1 and 12.0.0.1; and EID with 192.168.1.1 and 192.168.1.2) with same results. Nevertheless, we use EIDs as the most feasible options.

Before interpreting results, please note that *Host1* randomly (using same random generator seeds) chose eight different EIDs, *Host2* six EIDs, fourteen distinct ping destinations in the summary.

Phase	**α cache misses**		**β cache misses**		**γ cache misses**	
	xTR1	*xTR2*	*xTR1*	*xTR2*	*xTR1*	*xTR2*
#3	8	0	8	0	8	0
#5	0	14	0	6	0	6
#6	14	0	0	0	0	0
Total	22	14	8	6	8	6

Tab. 3: Count of map-cache misses under different configurations in scenario with one outage

Without any synchronization, traffic diversion to a new VRRP Master always causes misses due to unknown mappings. We can see it in phases #5 and #6 for α-run when the router starts to dispatch LISP data with the empty map-cache.

If synchronization is employed, then, only new destinations lead to map-cache miss. This is because a new VRRP Master already has mappings discovered by neighbor xTR. Hence, there is a difference in phase #5 for α-run (empty cache) and β/γ-runs (cache in sync with SS member). The difference (36 cache misses versus 14) would be even more significant in the case of multiple VRRP Master outages. Please note that every map-cache miss is also connected with the data packet drop.

In order to compare synchronization modes, we conducted measurement taking into account all LISP control messages processed by `LISPCore` module, namely their packet sizes. We assume that larger size is always a greater burden for router's control plane processing. Fig. 25 shows results (α-run = blue crosses, β-run = green triangles, γ-run = red circles), where each symbol represents one LISP control message.

We can see that *smart* outperforms *naïve* because it is less intensive while only single mapping is transferred during synchronization, not a whole map-cache. Moreover, both synchronization modes are better than no synchronization on protocol overhead because they decrease the number of mapping queries (i.e., exchanged messages count). The difference is not so significant on Fig. 25, especially between *naïve* and no sync mode. However, it is getting more obvious as the number of VRRP outages increases. Following table and figure prove this claim for the same network but with two *xTR1* outages – basically phases #4 and #6 repeat twice.

Phase	**α cache misses**		**β cache misses**		**γ cache misses**	
	xTR1	*xTR2*	*xTR1*	*xTR2*	*xTR1*	*xTR2*
#3a	8	0	8	0	8	0
#5a	0	14	0	6	0	6
#6a	14	0	0	0	0	0
#5b	0	0	0	0	0	0
#6b	14	0	0	0	0	0
Total	36	14	8	6	8	6

Tab. 4: Count of map-cache misses under different configurations in scenario with two outages

Repetition of phases 4), 5) and 6) is denoted in Tab. 4 with letters: "a" for the first outage; and "b" for the second outage. In Tab. 4, we can observe that a total number of cache misses for α-run has increased by 14. *xTR1* had gone down (losing its map-cache content), then went back (repopulating map-cache once again with 14 EIDs) and then this cycle repeats once again. For β-run and γ-run, additional outages pose no change, because *xTR1* completely synchronizes itself with *xTR2* (*xTR2* sends the whole map-cache as soon as it detects the status of the one of *xTR1*'s RLOCs up), when it is once again operational. Fig. 26 shows an increase in a number of processed LISP control message for no synchronization, where impacts of other synchronization techniques remain same.

LISP synchronization acknowledgment mechanism poses an additional control plane burden. In order to evaluate acknowledgment impact, we conducted measurement on the same network with two outages. The results in a number of processed LISP control messages bytes are depicted in Fig. 27 and can be compared with Fig. 25.

It is apparent that protocol overhead on the number of messages has increased. In the case of no synchronization, it slightly outperforms *naïve* mode by a total size of processed bytes. However, the *smart* mode still has the best characteristic even with enabled acknowledgments. Once again, we can expect that additional outages or more EID ping destinations would influence results in favor of β/γ-runs over α-run.

To summarize the evaluation of map-cache synchronization technique, we provide Tab. 5, which shows α/β/γ-run (i.e., none, *naïve* and *smart* sync) statistics for different scenarios (i.e., one/two/three outage(s) with or without acknowledgment). xTR1's statistic numbers are depicted with following column meanings: "miss" as the number of map-cache miss occurrence; "cnt" as the total count of LISP control plane messages sent and received; "size" as processed messages count by LISP control plane measured in total byte size. We added to Tab. 5 also same statistics section for the scenario with three outages in order to analyze trends even thou that it is not described via dedicated table and graph above. Results show a linear growth in complexity.

single xTR1 outage scenario									**single xTR1 outage with sync ack scenario**								
α			**β**			**γ**			**α**			**β**			**γ**		
miss	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**
22	81	4 458	8	62	4 328	8	62	3 796	22	81	4 458	8	71	5 458	8	71	4 394
two xTR1 outages scenario									**two xTR1 outages with sync ack scenario**								
α			**β**			**γ**			**α**			**β**			**γ**		
miss	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**
36	109	5 718	8	63	4 614	8	63	4 082	36	109	5 718	8	73	6 030	8	73	4 966
three xTR1 outages scenario									**three xTR1 outages with sync ack scenario**								
α			**β**			**γ**			**α**			**β**			**γ**		
miss	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**	**miss**	**cnt**	**size**
50	137	6 978	8	64	4 900	8	64	4 368	50	137	6 978	8	75	6 602	8	75	5 538

Tab. 5: xTR1's statistics for different map-cache synchronization scenarios

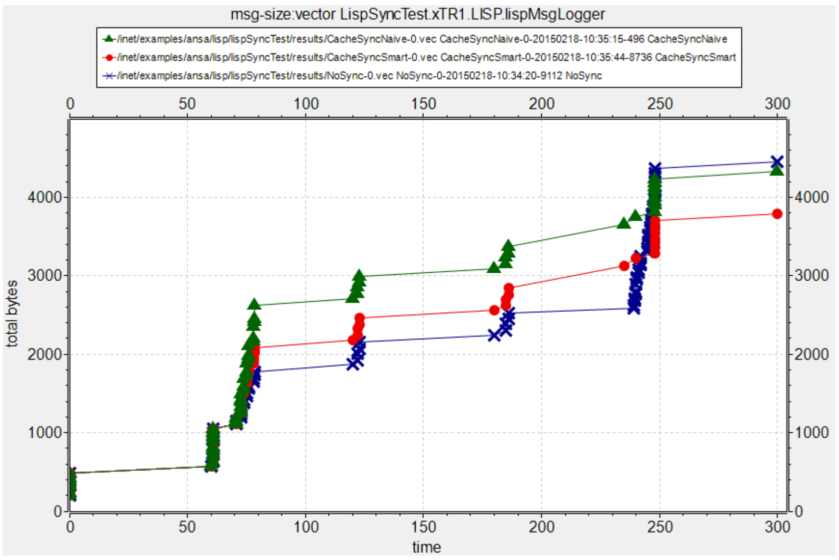


Fig. 25: xTR1's LISP control messages occurrence and total processed byte size in scenario with single outage

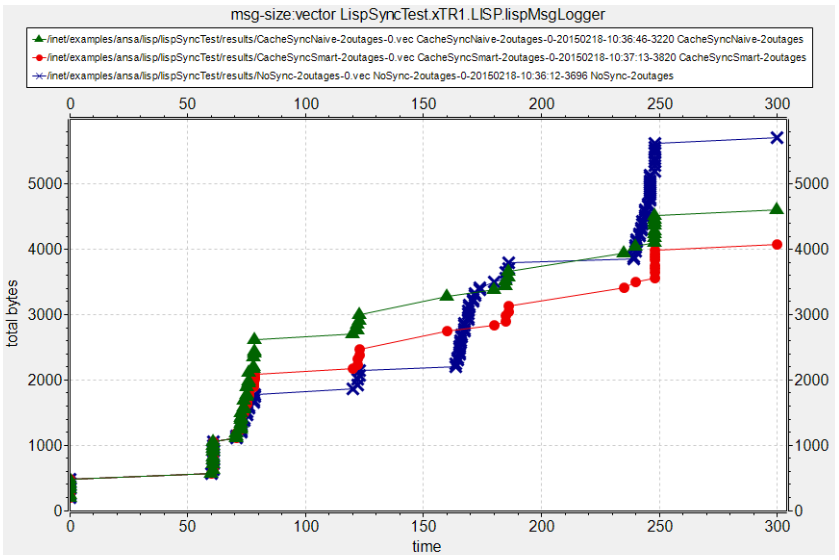


Fig. 26: xTR1's LISP control messages occurrence and total processed byte size in scenario with two outages

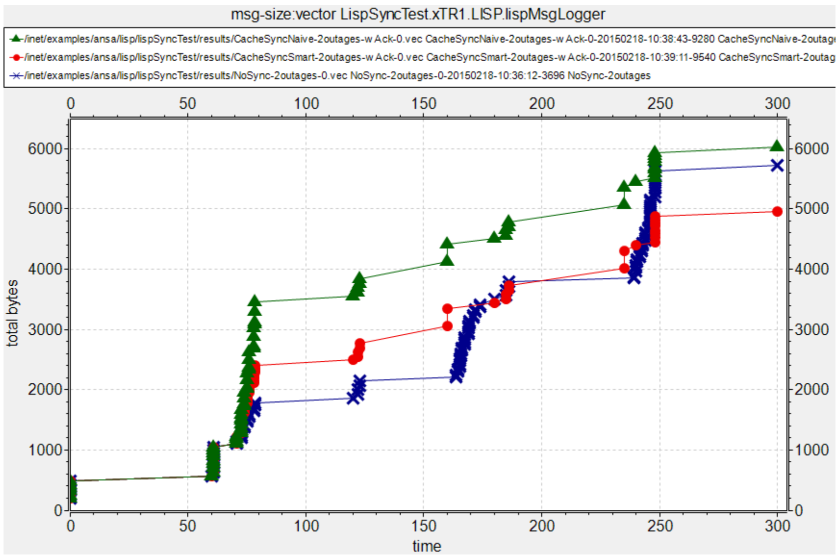


Fig. 27: xTR1's LISP control messages occurrence and total processed byte size in scenario with two outages + ack

Impact of Merged RLOC Probing

We took the previous network and adjusted it. Currently, it contains a LISP site with just one *xTR* router and one end-device called *Host1*. More important are LISP sites that are reachable via *xTR_Responder1* and *xTR_Responder2*. We simulate multiple EID networks reachable via the same xTRs with the help of loopback interfaces. Each xTR_Responder has forty loopbacks with EID addresses in the range of 172.16.[0-39].0/24. Each EID is being registered towards *MRMS* as reachable via *xTR_Responder1*'s RLOC 21.0.0.1 and *xTR_Responder2*'s RLOC 22.0.0.1. VRRP functionality on *xTR* is disabled because it is not needed for this scenario. *Host1* might randomly generate ICMP traffic towards destination EIDs, but this is not necessary for merged RLOC probing analysis. All communicating parties are interconnected via *Core* employing static routing configuration. The whole network graph is depicted in Fig. 28. Also this scenario (named "LispProbeTest") is located in `/examples/ansa/lispProbeTest` folder of available source codes.

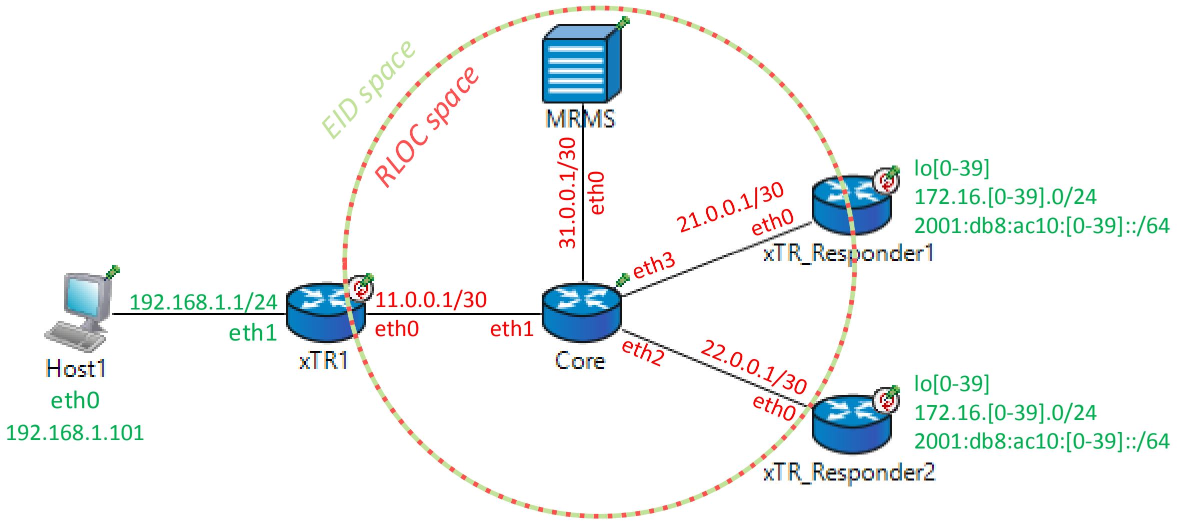


Fig. 28: LISP testing network for merged RLOC probing

RLOC probing starts immediately after LISP routing control plane is initialized. Following phases occur no matter on used RLOC probing algorithm:

- Probing xTR sends *LISP Map-Request Probe* to RLOC address for a given set of EIDs;
- Probed xTR responds with *LISP Map-Reply Probe* announcing that RLOC is up;
- In case that *LISP Map-Request Probe* was not replied, probing xTR repeats the probe at time $t_{next} = t_{last} + 2^{numOfRetries}$, where t_{last} is the time last probe was sent and $numOfRetries$ is a number of retry attempts to send this probe. By default, after three unsuccessful *LISP Map-Request Probe*, RLOC is marked as down and the next probe is scheduled after 60 seconds.

Optional phase 3) behavior is solely based on Cisco implementation observations. Also Cisco's LISP implementation has some other specifics: a) postponed start of first EID registration ($t + 60$ seconds since control plane initialization); b) postponed start of RLOC probing for IPv6 RLOCs ($t + 30$ since the first IPv4 probe). We have integrated this behavior into the LISP simulator. However, we are not employing it in order to provide better readability of this scenario's results.

These phases repeat by default every minute to keep RLOC reachability up-to-date. This interval could be decremented to a lower value, but protocol overhead increases in an inverse relationship.

Measurement is focused on a number of *LISP Map-Request/Reply Probes* exchanged between *xTR_Responder1* and *xTR_Responder2* and the amount of corresponding bytes processed by *xTR_Responder1*'s LISP control plane. We assume that five minutes simulation time is a period long enough to show the trend of each RLOC probing algorithm. During this period, five RLOC probe batches occur. Except mandatory EID registrations, no other LISP control traffic is spoiling the results.

We have conducted two simulation scenarios in order to observe complexity trends. The first one is for the network with forty different EIDs (twenty IPv4 172.16.[0-19].0/24 and twenty IPv6 2001:db8:ac10:[0-19]::/64) on xTR_Responders reachable via RLOCs 21.0.0.1 and 22.0.0.1, the second with eighty different EIDs (forty IPv4 172.16.[0-39].0/24 and forty IPv6 2001:db8:ac10:[0-39]::/64).

All three algorithms are evaluated separately as different configuration simulation runs - Cisco's default algorithm as δ-run, *simple* as ε-run and *sophisticated* as λ-run algorithm variants of merged RLOC probing.

40 EIDs scenario						80 EIDs scenario					
δ		ε		λ		δ		ε		λ	
cnt	size	cnt	size	cnt	size	cnt	size	cnt	size	cnt	size
805	55 500	25	8 520	25	28 530	1 605	110 900	25	15 920	25	56 330

Tab. 6: xTR_Responder1's statistics for different RLOC probing algorithm scenarios

Total count of sent and received LISP control messages are shown in Tab. 6. Columns have following meaning: "cnt" as the total count of LISP control plane messages sent and received; "size" as the amount processed messages by LISP control plane measured in total byte size.

Apart from five *LISP Map-Register*, *xTR_Responder1* five times: a) sends *LISP Map-Request Probe* and receives *LISP Map-Reply Probe*; b) receives *xTR_Responder2*'s probes and responds to them with replies. It is apparent that a count of exchanged messages is drastically lower when using any merged RLOC probing algorithm. Cisco's algorithm generates RLOC probe for each EID-to-RLOC mapping, which means forty/eighty *LISP Map-Request Probe* and forty/eighty *LISP Map-Reply Probe* messages per single phases #1 and #2 occurrences. Opposite to that any merged RLOC algorithm exchanges only single *LISP Map-Request/Reply Probe* pair between xTR_Responders.

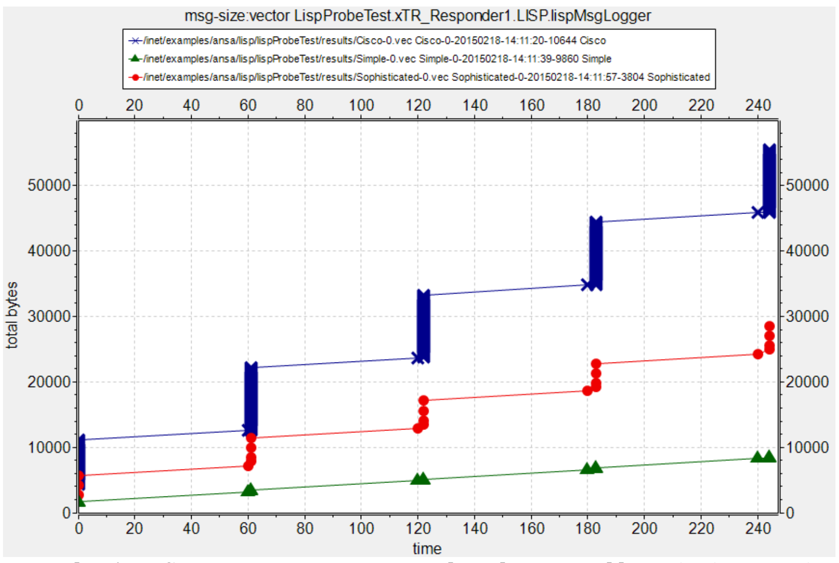


Fig. 29: xTR_Responder1's LISP messages occurrence and total processed byte size in scenario with forty EIDs

In Fig. 29, we can see that ε-run has better protocol overhead measured in the total amount of bytes processed by *xTR_Responder1*. This is because each probe carries only single EID chosen in a round-robin fashion, where successful reception of *LISP Map-Reply Probe* refreshes RLOC state for all EIDs that are using it. In the case of the *sophisticated* algorithm, all relevant EIDs are packed in single probe thus (significantly) increasing its size (but still half of Cisco's total processed byte size). On the other hand *simple* merged RLOC probing algorithm might seem to be too simple and lacking of accuracy if we want the use-case where the same RLOC is up for some EIDs, and down for another EIDs. In that case, *sophisticated* variant offers the same functionality but with better granularity.

4.4 Chapter Summary

This chapter described in great detail all routing aspects of LISP. In the first subchapter, we started with a basic overview of LISP functionality and its main components. We focused on the distributed mapping system of LISP including how map-cache content impacts LISP routing performance. We outlined LISP signalization messages together with their semantics. We discussed ways how LISP coexists with traditional TCP/IP networks and what are transition possibilities and deployment options.

The last subchapter described one of the main contributions of this thesis. Two major issues are introduced that limit LISP operation – Site-based Synchronization Problem and Locator Path Liveness Problem. Furthermore, we proposed concrete map-cache synchronization techniques and merger RLOC probing algorithms, which should reduce protocol overhead and increase LISP routing performance. Hence, these improvements should at least partially deal with problems above. Moreover, we developed and implemented brand new simulation modules of LISP (and as a byproduct also VRRP) intended for OMNeT++. Employing these modules, we tested and successfully proved the effectiveness of proposed improvements.

If we want to qualify and quantify impact of our propositions the following items hold:

- Both *naïve* and *smart* map-cache synchronization modes significantly reduce (theoretically to zero) map-cache misses for sites with multiple ITRs;
- *Smart* mode outperforms *naïve* mode in protocol overhead (in number of processed bytes):
    - having approx. 11% lower overhead for scenarios without acknowledgment and both are better than no synchronization;
    - having approx. 17% lower overhead for scenarios with acknowledgment, where *smart* is always better than no synchronization, and *naïve* gets better with more outages;
- Merged RLOC probing decreases radically protocol overhead (in processed bytes count) of locator liveness checking:
    - the *simple* algorithm reduces overhead by approx. 85%;
    - the *sophisticated* algorithm reduces overhead by approx. 50%;

5 Recursive Internet Architecture

> ֍ –"*In order to understand recursion, one must first understand recursion.*" Anonymous
> ֍ *What is RINA and what are its most distinctive features?*
> ֍ *Can we prove RINA's feasibility as the clean-slate architecture?*

RINA is the clean-slate architecture aimed to change the whole Internet unlike just temporary fixes for current status quo. RINA concept is based on John Day's thoughts, lectures and book [20] regarding ISO/OSI initiative failure, TCP/IP development, commercial adoption of the Internet and other technical/political events in Internet history.

This chapter familiarizes the reader with RINA basics. Based on our experience, we must admit that "mental-shift" from nowadays networking towards RINA is not easy at all. Hence, the reader is advised to seek further in related references when confused.

Among main goals of this chapter are the following items: a) to introduce RINA as a new networking paradigm; b) to provide an in-depth explanation of RINA's operation; c) to revisit and improve some of RINA specifications; d) to develop the first RINA simulator as a new educational and research tool.

5.1 Overview

This subchapter introduces theoretical background. However, explanation of the whole Recursive Internet Architecture is far beyond the scope of this thesis. Hence, only parts relevant to the current RINASim functionality are captured. Synthesis of RINA information provided below comes from the following sources: [71], [72], [73], [74] and [75].

5.1.1 Nature of Applications and Application Protocols

Is *application* a part of IPC environment or not? The set of Internet applications was rather simplistic before WWW – one application with a single instance using only one protocol. Hence, there is nearly no distinction between an application and its networking part. However, the web completely changed this situation – one application protocol may be used by more than one application and also one application may have many application protocols.
Following terms are recognized in the frame of RINA, and their relationship is depicted in Fig. 30:

- **Application Process (AP)** – Program instantiation to accomplish some purpose;
- **Application Entity (AE)** – AE is the part of AP, which represents application protocol and application aspects concerned with communication.

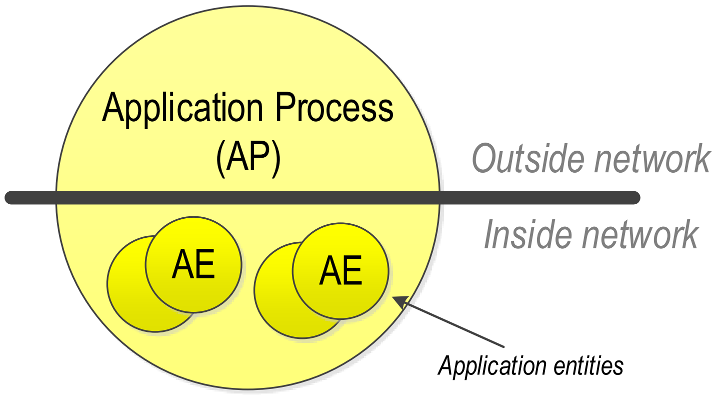


Fig. 30: Application Protocol and Application Entities relationship

There may be multiple instances of the Application Process in the same system. AP may have multiple AEs, each one may process different application protocol. There also may be more than one instance of each AE type within a single AP.

All application protocols are stateless; the state is and should be maintained in the application. Thus, all **application protocols** modify shared state external to the protocol itself on various objects (e.g. data, file, HW peripherals). Because of that, there is only one application protocol that contains trivial operations (e.g., read/write, start/stop). **Data transfer protocols** modify state internal to the protocol, the only external effect is the delivery of SDUs.

5.1.2 Core Terms

The data transport and internetworking tasks together (generally known as *networking*) constitute **inter-process communication (IPC)**. IPC between two APs on the same operating system needs to locate processes, evaluate permission, pass data, schedule tasks and manage memory. IPC between two APs on different systems works similarly plus adding functionality to overcome the lack of shared memory.

In traditional networking stack, the layer provides a service to the layer immediately above it. As RINA name suggests, recursion and repeating of patterns are the main feature of the whole architecture. Layer recursion became more popular even in TCP/IP with technologies like Virtual Private Networks (VPNs) or overlay networks (e.g., OTV[29]). Recursion is a natural thing whenever we need to affect the scope of communicating parties. However, so far it was just recursion of repeating functions in existing layers. RINA is based on following core ideas:

— *"Networking is interprocess communication…and IPC only!"* [76]

— *"Application Processes communicate via a service provided by a distributed application that provides IPC. The application processes that make up this Distributed IPC Facility provide a protocol that implements an IPC mechanism, and a protocol for managing distributed IPC (routing, security and other management tasks)."* [77]

In ISO/OSI or TCP/IP, there is a set of layers each with completely different functions. RINA, on the other hand, yields idea of the single generic layer with fixed mechanisms but configurable policies. This layer is in RINA called **Distributed IPC Facility (DIF)** – a set of cooperating APs providing IPC. There is not a fixed number of DIFs in RINA; we can stack them according to application or network needs. From the DIF point of view actual stack depth is irrelevant, DIF must know only (N+1)-layer above and (N-1)-layer below. DIF stacking partitions network into smaller, thus, more manageable parts.

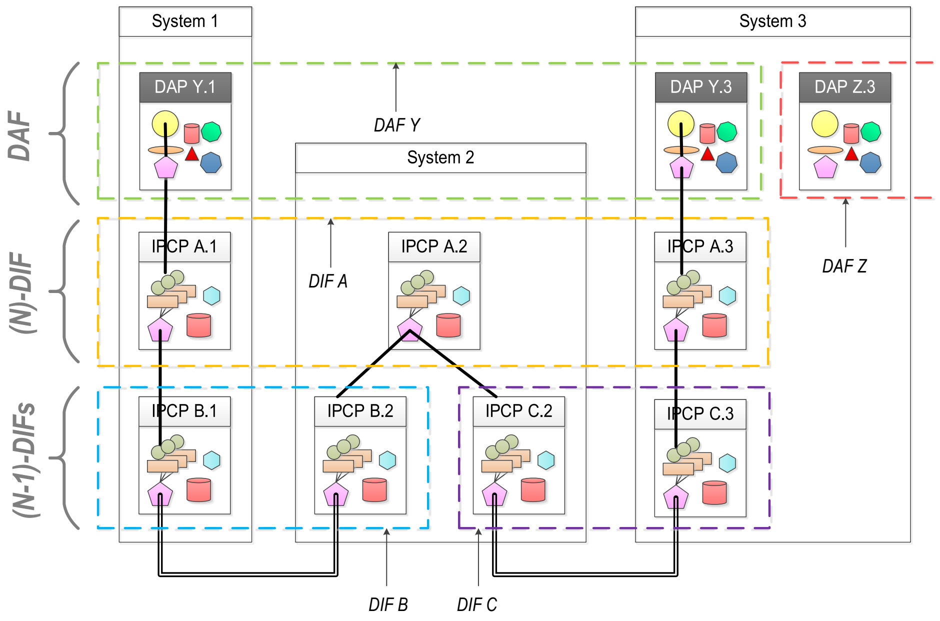


Fig. 31: DIF, DAF, DAP and IPCP illustration

[29] Overlay Transport Virtualization (OTV). For more, see http://www.cisco.com/c/en/us/solutions/data-center-virtualization/overlay-transport-virtualization-otv/index.html

The concept of RINA layer could be further generalized to **Distributed Application Facility (DAF)** – a set of cooperating APs in one or more computing systems, which exchange information using IPC and maintain shared state. A DIF is a DAF that does only IPC. **Distributed Application Process (DAP)** is a member of a DAF. **IPC Process (IPCP)** is special AP within DIF delivering inter-process communication. IPCP is an instantiation of DIF membership; computing system can perform IPC with other DIF members via its IPC process within this DIF. An IPCP is specialized DAP. The relationship between all newly defined terms is depicted in Fig. 31.

DIF limits and encloses cooperating processes in the one scope. However, its functionality is more general and versatile apart from rigid TCP/IP layers with dedicated functionality (i.e., data-link layer for adjacent node communication, a transport layer for reliable data transfer between applications). DIF provides IPC to either another DIF or to DAF. Therefore, DIF uses a single application protocol with generic primitive operations to support inter-DIF communication.

5.1.3 Connection-oriented vs. Connectionless

The clash between connection-oriented and connectionless approaches (that also corrupted ISO/OSI tendencies) is from RINA perspective quite easy to settle. Connection-oriented and connectionless communication are both just functions of the layer that should not be visible to applications. Both approaches are equal, and it depends on application requirements which one to use. On the one hand, connectionless is characterized by the maximal dissemination of the state information and dynamic resource allocation. On the other hand, connection-oriented limits the dissemination and tends toward static resource allocation. The first one is good for low volume stochastic traffic. The second one is useful for scenarios with deterministic traffic flows.

If the applications request the allocation of communication resources, then layer determines what mechanisms and policies to use. Allocation is accompanied by access rights and description of QoS demands (e.g., what minimum bandwidth or delay is needed for correct operation of application).

5.1.4 Delta-t Synchronization

All properly designed data transfer protocols are soft-state. There is no need for explicit state synchronization (hard-state) and tools like SYNs and FINs are unnecessary.

Initial synchronization of communicating parties is done with the help of **Delta-t** protocol (see [78] and [79]). Delta-t was developed by Richard Watson, who proposed time-based synchronization technique. He proved that conditions for distributed synchronization were met if the following three timers are realized: a) **Maximum Packet Lifetime** (**MPL**); b) Maximum time to attempt retransmission a.k.a. maximum period during sender is holding PDU for retransmission while waiting for a positive acknowledgment (a.k.a. **R-timer**); c) Maximum time before Acknowledgement (a.k.a. **A-timer**).

Delta-t assumes that all connections exist all the time. Synchronization state is maintained only during the activity, but after 2-3 MPL periods without any traffic it may be discarded which effectively resets the connection. Because of that, there are no hard-state (with explicit synchronization) protocols only soft-state ones. Delta-t postulates that port allocation and synchronization are distinct.

5.1.5 Separation of Mechanism and Policy

Just to remind the reader that mechanism is fixed, the policy is flexible part of any IPC. In the same subchapter, the most common mechanisms have been cataloged using ontology. Nevertheless, this mechanism list is not final and even to several mechanisms exist dozens/hundreds of different policies, how exactly are these mechanisms implemented and enforced.

If we focus only on mechanisms connected with data transfer, then we can clearly separate them into two groups:

- **tightly-bound** that must be associated with every PDU, which handle fundamental aspects of data transfers (e.g., the sequence number of every PDU, integrity check using hashes associated with the PDU content);

- **loosely-bound** that may be associated with some data transfer PDUs, which provide additional features (namely reliability and flow control).

Both groups are coupled through state-vector maintained separately per flow; every active flow has its state-vector holding state information. For instance, the behavior of retransmission and flow control can be heavily influenced by chosen policies and they can be used independently on each other. This implies that only single generic data transfer protocol based on Delta-t is needed, which may be governed by different transfer control policies. This data transfer protocol modifies state internal to its PM, where application protocol (carried inside) modifies state external to PM.

5.1.6 Naming and Addressing

Application Process communicates in order to share state. In 5.1.1, we mentioned that AP consists of AEs. We need to differentiate between different APs and also different AEs within the same AP. Thus, RINA is using **Application Process Name (APN)** as globally unambiguous, location-independent, system-dependent name. **Application Process Instance Identifier (API-id)** differentiates between multiple instances of the same AP in the system. **Application Entity Instance Identifier (AEI-id)**, which is unambiguous for a single AP, helps us to identify different AE instances of same **Application Entity Name (AEN)** within AP. **Application Naming Information (ANI)** references a complete set of identifiers to name particular application; it consists of four-tuple APN, API-id, AEN, and AEI-id. The only required part of ANI is APN; others are optional. **Distributed Application Name (DAN)** is globally unambiguous name for a set of system-independent APs.

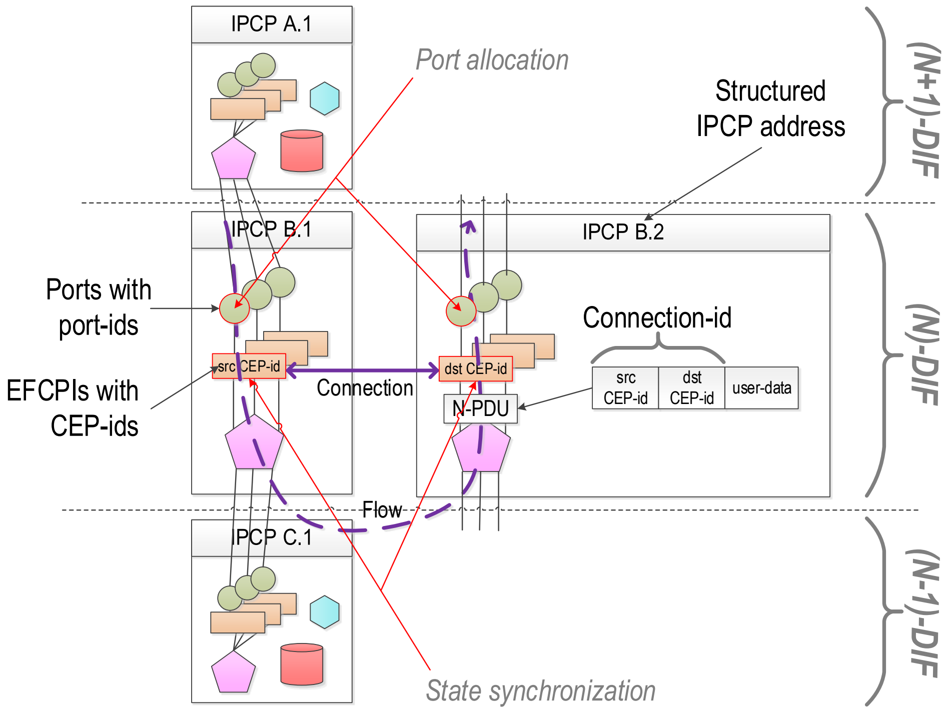


Fig. 32: IPCP local identifiers overview

IPC Process has APN to identify it among other DIF members. An RINA **address** is a synonym for IPCP's APN with a scope limited to the layer and structured to facilitate forwarding. APN is useful for management purposes but not for forwarding. Address structure may be *topologically dependent* (indicating the nearness of IPCPs). <u>APN and address are simply two different means to locate an object in different context.</u> There are two local identifiers important for IPCP functionality – port-id and connection-endpoint-id. **Port-id** binds this (N)-IPCP and (N+1)-IPCP/AP; both of them use the same port-id when passing messages. Port-id is returned as a handle to the communication allocator and is unambiguous within a computing system. **Connection-endpoint-id (CEP-id)** identifies a shared state of one communication endpoint. Since there may be more than one flow between the same IPCP pair, it is necessary to distinguish them. For this purpose, **Connection-id** is formed by combining source and destination CEP-ids with QoS requirements descriptor. CEP-id is unambiguous within IPCP and Connection-id is unambiguous between a given pair of IPCPs. Fig. 32 depicts all relevant identifiers between two IPCPs.

Watson's delta-t implies port-id and CEP-id in order to help separate port allocation and synchronization. RINA's **connection** is a shared state between N-PMs – ends identified by CEP-ids. RINA's **flow** is when connection ends are bound to ports identified by port-ids. The lifetimes of flow and its connection(s) are independent of each other.

The relationship between node and PoA is relative – node address is (N)-address, and its PoA is (N-1)-address. Routes are sequences of (N)-addresses, where (N)-layer routes based on this addresses (not according to (N-1)-addresses). Hence, the layer itself should assign addresses because it understands address structure.

5.2 RINA Components

To understand RINA architecture means to understand each of its elements. This subchapter starts with a description of high-level RINA network nodes and then goes deeper and outlines various IPC Management and IPCP components.

5.2.1 Nodes

There are only three basic kinds of nodes in RINA network (illustrated in Fig. 33). Each kind represents computing system running RINA:

- **Hosts** – end-devices for IPC containing AEs in the top layer; they employ two or more DIF levels;
- **Interior routers** – interim devices, which are interconnecting (N)-DIF neighbors via multiple (N-1)-DIFs; they employ two or more DIF levels;
- **Border routers** – interim devices, which are interconnecting (N)-DIF neighbors via (N-1)-DIFs, where some of (N-1)-DIFs are reachable only through (N-2)-DIFs; they employ three or more DIF levels.

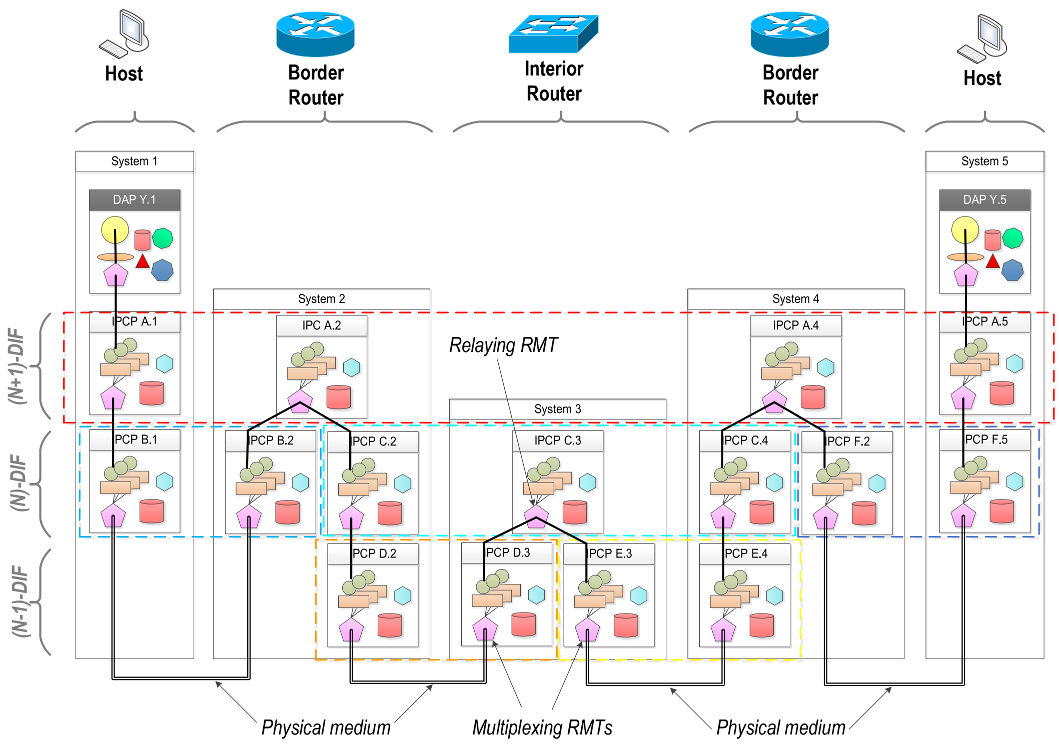


Fig. 33: Example of RINA network with three levels of DIFs and different nodes

As seen in Fig. 33, the main difference between node kinds is in an overall number of DIF levels present in a computing system. Due to the limited number of network interface cards (NIC), Hosts usually have a single 0-DIF (connected to the physical medium) and a few 1-DIFs leveraging on this lowest level DIF. Interior routers have potentially a lot of 0-DIFs (for each interface) but only a few relaying 1-DIFs. Border routers also perform relaying but serve as gateways between those (N-1)-IPCs, which are not connected directly. Thus, (N-2)-DIF is needed to reach physical medium.

5.2.2 IPC Management Components

IPC Management is an integral part of any DAP responsible for managing supporting DIFs and providing their services to participating APs.

Only IPC Resource Manager and DIF Allocator interface are exclusive to IPC Management, other components are also present in IPC Process and described later.

DIF Allocator

The primary task of **DIF Allocator (DA)** is to return a list of DIFs where destination application may be found given ANI and access control information. Additional and more complex DA description is available in [80]. DA contains and works with multiple mapping tables to provide its services:

- **Naming information table** – provides association between APN and its synonyms;
- **Search table** – provides mapping between requested APN and the list of DAs where to search for it next;
- **Neighbor table** – maintains a list of adjacent peers when trying to reach other DAs;
- **Directory** – contains records mapping APNs with access rights to the list of supporting DIFs including DIF's name, access control information and provided QoS.

IPC Resource Manager

IPC Resource Manager (IRM) (see specification [81]) as its name suggests manages DAF resources. This involves multiple different tasks:

- IRM processes *allocate* calls by delegating them to appropriate local IPCPs in relevant DIFs;
- IRM manages DA queries and acts upon their responses. When the DA response contains more than one DIF, IRM chooses which DIF to use;
- IRM manages the use of flows between AEs and DIFs. IRM may choose to multiplex a single or multiple AE flows into a single/multiple flows to a set of DIFs;
- IRM initiates joining or creating DAF and/or DIF. IRM acts upon the DAF, or DIF lost (e.g., sending notifications or perform subsequent actions).

5.2.3 IPC Process Components

IPC Process is instance within DIF, which allows the computing system to do IPC with other DIF members. Each IPC process performs (secure/reliable) data transport, (authenticated) enrollment, (de)allocation of resources, routing, management and more. Functions could be categorized under one of following categories: a) data transfer; b) data transfer control; and c) IPC management. Each category with different processing timescale and complexity – a) is simplest and performed the most often, c) the least often but the functionality is rather complex.

IPC provides API to a DIF/DAF above, which requested its service. Basic **IPC API** offers four operations: *allocate* (allocates communication resources); *deallocate* (releases previously allocated resources); *send* (passes SDU to IPC) and *receive* (retrieves SDU from IPC). Calls may be further subdifferentiated as *allocate request*, *allocate response*, *deallocate submit* and *deallocate deliver*.

Graphical representation of IPC Process and its most important components is depicted in Fig. 34. A brief description of each component and their functionality is provided below figure. Some components outlined below also contain policy descriptions. Those policies are mentioned because they are relevant to our contribution.

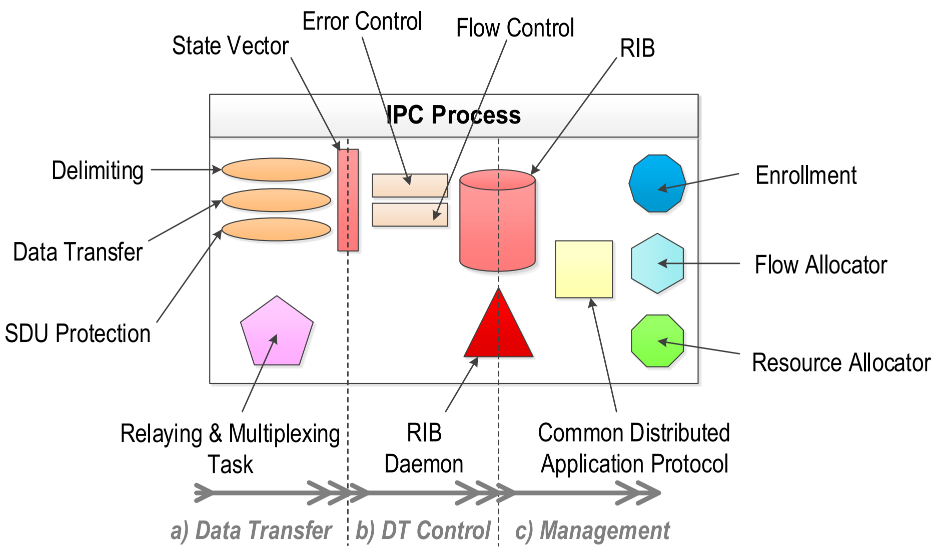


Fig. 34: IPC Process components

Enrollment

Enrollment takes place whenever IPCP joins existing DIF. IPCP newcomer creates a connection with other IPCP (which is already a member) allocating (N-1)-flow. Enrollment occurs after successful connection establishment. Enrollment procedure of a new member should be dependent on a connection use-case. For instance, there may be a different exchange of messages for: a) the new member joining DIF for the first time; b) the IPCP that had been already a member of DIF and right now is rejoining. The new member either tells or gets its address to/from a DIF. Enrollment procedure is codified in [82].

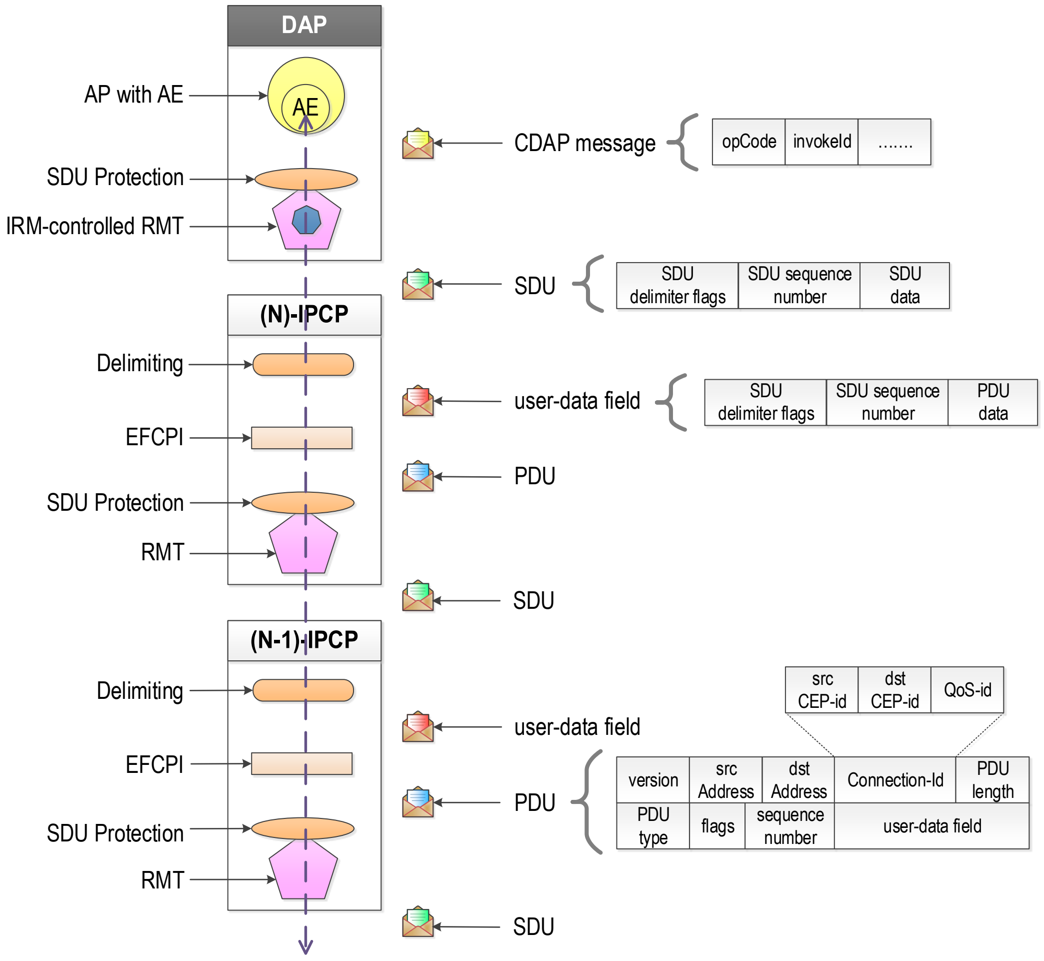


Fig. 35: Message passing between RINA components

Delimiting

SDU in RINA is a contiguous chunk of data. IPC might fragment SDU (when passing it down) or combine user-data (when passing it up). Hence, the operation performed by **Delimiting** module (for specification see [83] and [84]) is to delimit SDU into/from PDU's user-data preserving its identity. Employed mechanism indicates the beginning and/or the end of SDUs. Either internal (special pattern) or external (SDU length in PCI) delimiting could be used.

Encapsulation/Decapsulation of data messages happens in RINA components lying in the data path. Fig. 35 depicts this process DIF/DAF together with messages nomenclature.

Data Transfer with Error/Flow Control

Error and Flow Control Protocol (EFCP) is split into two independent PMs coupled and coordinated through a state vector. As EFCP name suggests, EFCP guarantees data transfer and data control. Full EFCP functionality is described in [85]. However, these specifications are currently being revisited.

Data Transfer Protocol (DTP) implements mechanisms tightly coupled with transported SDUs, e.g., fragmentation, reassembly, sequencing. DTP PM operates on a data PDU's PCI with fields requiring minimal processing – source/destination addresses, QoS requirements, Connection-id, optionally sequence number or checksum. DTP carries user-data.

Data Transfer Control Protocol (DTCP) implements mechanisms that are loosely coupled with transported SDUs, e.g., (re)transmission control using various acknowledgment schemes and flow control with data-rate limiting. DTCP functionality is based on Watson's Delta-t and DTCP PM processes control PDUs. DTCP provides error and flow control over user-data.

There is **EFCP instance (EFCPI)** module per every active flow. EFCPI consists of DTP and DTCP submodules. DTCP policies are driven by the quality of service demands. DTCP submodule is unnecessary for flows that do not need it, i.e., flows without any requirements for reliability. Control traffic stays out of the main data transfer.

Relaying and Multiplexing Task

Relaying and Multiplexing Task (RMT) modules have two main responsibilities – relaying and multiplexing as characterized in [86]. The goal of multiplexing is to pass PDUs from EFCPIs and RIB Daemon to appropriate (N-1)-flows and reverse of that. Relaying handles incoming PDUs from (N-1)-ports that are not directed to its IPCP and forwards them to other (N-1)-ports using the information provided by its forwarding policy.

RMT instances in hosts and bottom layers of routers usually perform just the multiplexing task, while RMTs in top layers of interior/border routers do both multiplexing and relaying. In addition to that, RMTs in top layers of border routers perform flow aggregation. Primary RMT functions are demonstrated in Fig. 33.

Each (N-1)-port handled by RMT has its set of input and output buffers. The number of buffers, their monitoring, their scheduling discipline and classification of traffic into distinct buffers are all matter of policies.

RMT is a straightforward high-speed component. As such, most of its management (state configuration, forwarding policy input, buffer allocation, and data rate regulation) is handled by the Resource Allocator, which makes the decisions based on observed IPC process performance.

Each IPC process has to solve the forwarding problem: given a set of EFCP PDUs and (N-1)-flows leading to various destinations, to which flow should be each PDU forwarded? In RINA, the decision is handled by the RMT and its *PDUForwardingPolicy*. The *PDUForwardingPolicy* may consist of looking up the PDU's destination in its forwarding table (resembling the forwarding mechanism in traditional TCP/IP routers), but it is not a requirement; other experimental forwarding paradigms (such as forwarding based on topological addressing) may not require a forwarding table at all. When in need of deciding for an output (N-1)-port for a PDU, the *PDUForwardingPolicy* is given the PDU's PCI and then it returns a set of (N-1)-ports to which the PDU has to be sent. This provides enough granularity to implement multiple communication schemes apart from unicast (such as multicast or load-balancing) because the decision is left to the *PDUForwardingPolicy*. E.g., a simple forwarding policy would return a single (N-1)-port based on PDU's destination address and QoS-id, whereas in case of a load-spreading policy and multiple (N-1)-ports leading to the same destination, the policy could split traffic by PDUs' flow-ids and always return a single (N-1)-port from the set.

SDU Protection

SDU Protection is the last part of the IPCP data path, before an SDU is handed over to an underlying DIF. It is responsible for protecting SDUs from untrusted (N-1)-DIFs by providing mechanisms for lifetime limiting, error checking, data integrity protection and data encryption. It also provides mechanisms for data compression and a potential placeholder for other two-way manipulations.

SDU Protection handles each (N-1)-flow separately due to different levels of trust. This gives SDU Protection the ability to skip some mechanisms in favor of performance for trusted networks while still being protected from untrusted networks. Therefore, SDU Protection employs various policies, e.g: a) *NullSDUProtection* that performs no transformations; b) *BasicSDUProtection* that applies life time limiting and error checking; c) *CryptographicSDUProtection* that extends the *BasicSDUProtection* by adding cryptographic encryption of data and an integrity check using a cryptographic hash of the content.

Flow Allocator

Flow Allocator (FA) processes *allocate*/*deallocate* IPC API calls and further management of all IPCP's flows. FA instantiates a Flow Allocator Instance to manage each flow; FA is controller/container for all Flow Allocator Instances.

Flow Allocator Instance (FAI) is created upon *allocate request* call, and it manages a given flow for its whole lifetime. FAI handles creating/deleting EFCPI(s) while managing a single flow's connection. FAI returns port-id to the allocation requestor upon successful allocation as a referencing handle. FAI participates only on port allocation, not on synchronization, which is the responsibility of EFCPI. The FAI maintains a mapping between flow's local port-id and connection's local CEP-id.

FA contains **Namespace Management (NSM)** interface for assigning and resolving names (including synonyms) within DIF. This activity involves maintaining the table with entries that map requested ANI to IPCP's address.

Flow object contains all information necessary to manage any given flow between communicating parties. It is carried inside *create/delete flow request/response* messages controlling FA and FAI operation. Flow object contains: source and destination ANI, source and destination port-ids, connection-id, source and destination address, QoS requirements, a set of policies, access control information, hop-count, current and maximal retries of *create flow requests*.

Flow allocation processes for (N)-DIF between two APs on different systems is depicted in Fig. 36. It assumes that relevant (N-1)-flows have been already allocated using the same principle as the one being described but on different DIF's rank.

#1) *AP1* issues *allocate request* that is delivered to IPCP *A.1*. If it is valid and well-formed, then it spawns FAI to manage requested flow. FAI resolves *AP3*'s APN to one of DIF *A* addresses (*A.3*). It instantiates EFCPI (with CEP-id) and creates bindings between EFCPI and RMT. *Create flow request* is sent as the last step;

#2) *Create flow request* arrives at "System 2". IPCP *A.2*'s FA processes the request and discuss NMS. It discovers that request is not intended for any local AP. FA looks up the destination discovering that *A.3* should be a next-hop. FA forwards the request to "System 3";

#3) The request arrives at IPCP *A.3*. Over there, FA determines by querying NMS that *create flow request* destination address is its address. Thus, destination AP resides on this system. FAI is spawned and determine whether the request can be accommodated. If not then negative *create flow response* is sent back to the requestor. Otherwise, FAI notifies destination AP with *allocate request*;

#4) If destination AP accepts or rejects the request then either positive or negative *allocate response* is returned to FAI. Based on the response, FAI binds port-id, instantiates EFCPI, creates bindings. Flow object is updated (with local port-id and CEP-id) and sent back as positive/negative *create flow response*. Response is just relayed (not processed) on interior routers (IPCP *A.2*);

#5) Originating *A.1*'s FAI receives *create flow response* and updates relevant flow object. If the response is positive, then, FAI notifies source AP with positive *allocate response* and APs may commence data transfer. If the response is negative, then FAI invokes retry policy to correct flow creation or deal appropriately with failure (i.e., passing negative *allocate response*).

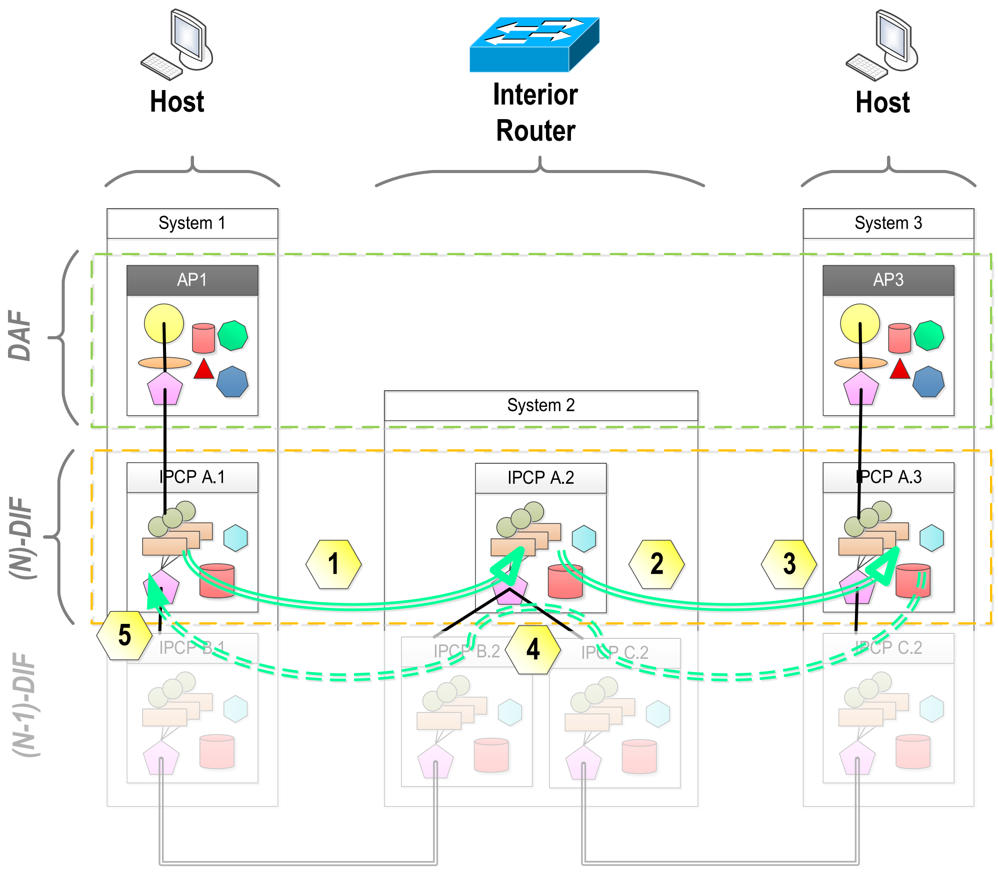


Fig. 36: Flow allocation process

Original specification [87] were refined as the subject of this thesis contribution. Detail description of flow allocation and deallocation is provided in Figures Fig. 37, Fig. 38, Fig. 39 and Fig. 40. Transitions are denoted with "input / action" labels. FA and FAI maintain state for any given flow and refuse inappropriate transitions (e.g., initiating deallocation before the allocation is successful). These transitions are omitted for clarity. There are four different FSMs. Fig. 37 depicts FA operation reacting upon notification from RIBd. Fig. 38 and Fig. 39 show flow allocation procedure for initiating and responding FAIs. Fig. 40 illustrates flow's lifecycle after successful allocation, and it is mutual for both initiating and responding FAIs.

NewFlowRequstPolicy is invoked after FAI's instantiation. Policy subtasks involve both 1) evaluation of access control rights; and 2) translation of QoS requirements specified in *allocate request* to appropriate RA's QoS-cubes. *AllocateRetryPolicy* occurs whenever initiating FAI receives negative *create flow response*. This policy allows FAI to reformulate the request and/or to recover properly from failure. *AllocateNotifyPolicy* controls a proper time when source AP is going to be notified of the result of allocation by initiating FAI. It may be either when EFCPI is created, or when allocation is confirmed by destination or any other notification strategy may be employed. *SeqRollOverPolicy* is invoked simultaneously by both initiating and responding FAIs whenever PDU's sequence number threshold is reached. The policy usually spawns new EFCPIs and changes bindings.

Resource Allocator

If a DIF has to support different qualities of service, then different flows will have to be allocated to different policies and traffic for them treated differently. **Resource Allocator (RA)** delineated in [88] is a component accomplishing this goal by handling management of various IPCP resources, namely it:

- controls creating/deleting and enlarging/shrinking of RMT queues;
- modifies EFCPI's DTCP policy parameters;
- controls creating/deleting of (N-1)-flows and their assignment to proper RMT queue(s);
- manages QoS classes and their assignment to RMT queue(s);
- manages routing information affecting RMT's relaying or initiates congestion control.

RA maintains a catalog of meters and dials by monitoring various management resources. Each catalog item can be manipulated and shared with other IPC processes within DIF.

Generating information necessary for *PDUForwardingPolicy* is one of the tasks of RA, namely its subcomponent called **PDU Forwarding Table Generator**. For this purpose, RA uses pieces of information provided by other sources, most notably the *RoutingPolicy*.

The *RoutingPolicy* exchanges information with other IPCPs in the DIF in order to generate a next-hop table for each PDU (usually based on the destination address and the id of the QoS class the PDU belongs to). The next-hop table is then converted into a **PDU Forwarding Table** with input from the PDU Forwarding Table Generator, by selecting an N-1 flow for each "next-hop". *RoutingPolicy* may resemble distance vector and link-state routing protocols used in today's Internet, but the current research is also aimed at other paradigms such as topological/hierarchical routing, greedy routing or MANET-like routing.

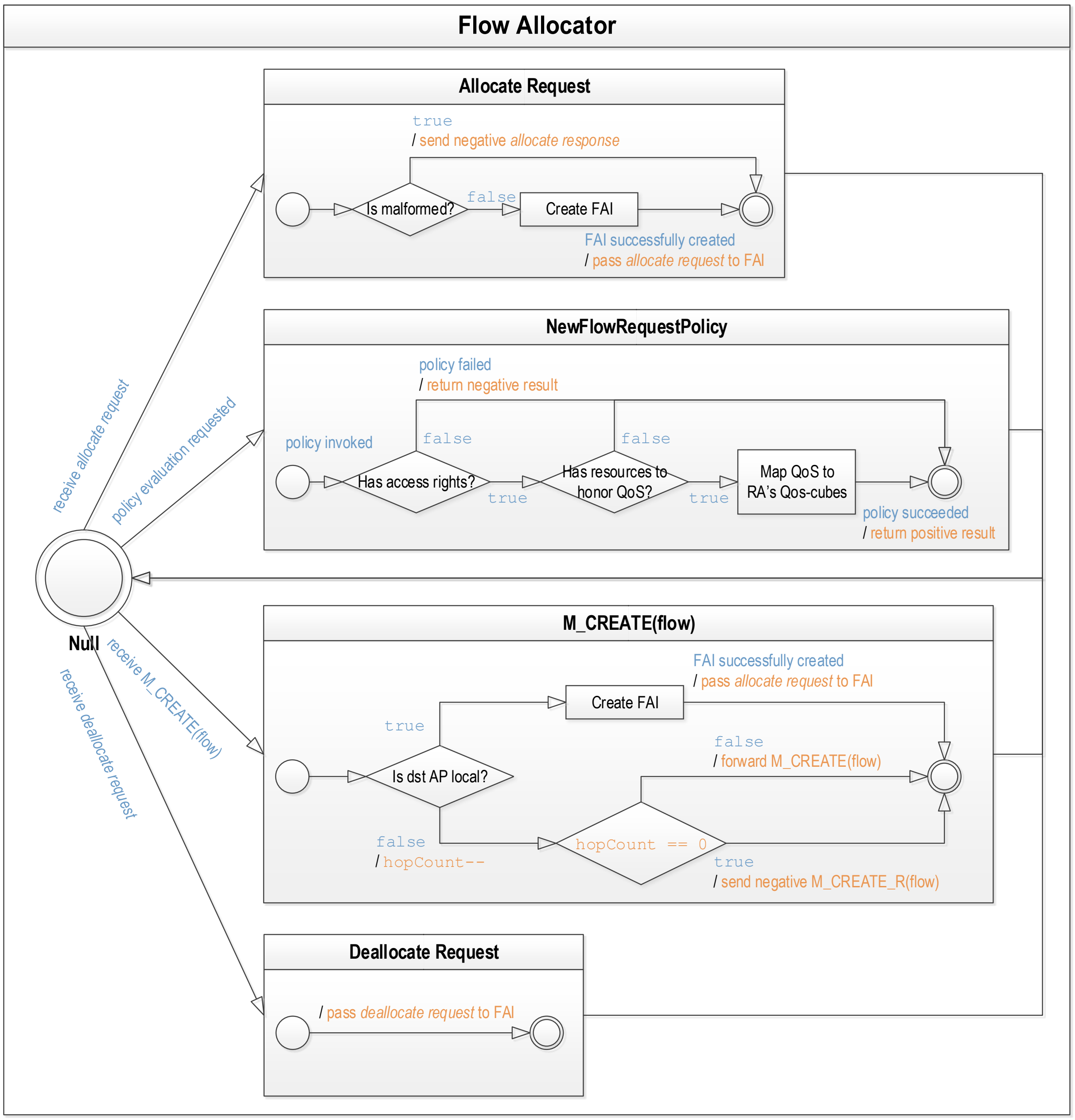


Fig. 37: Flow Allocator operation

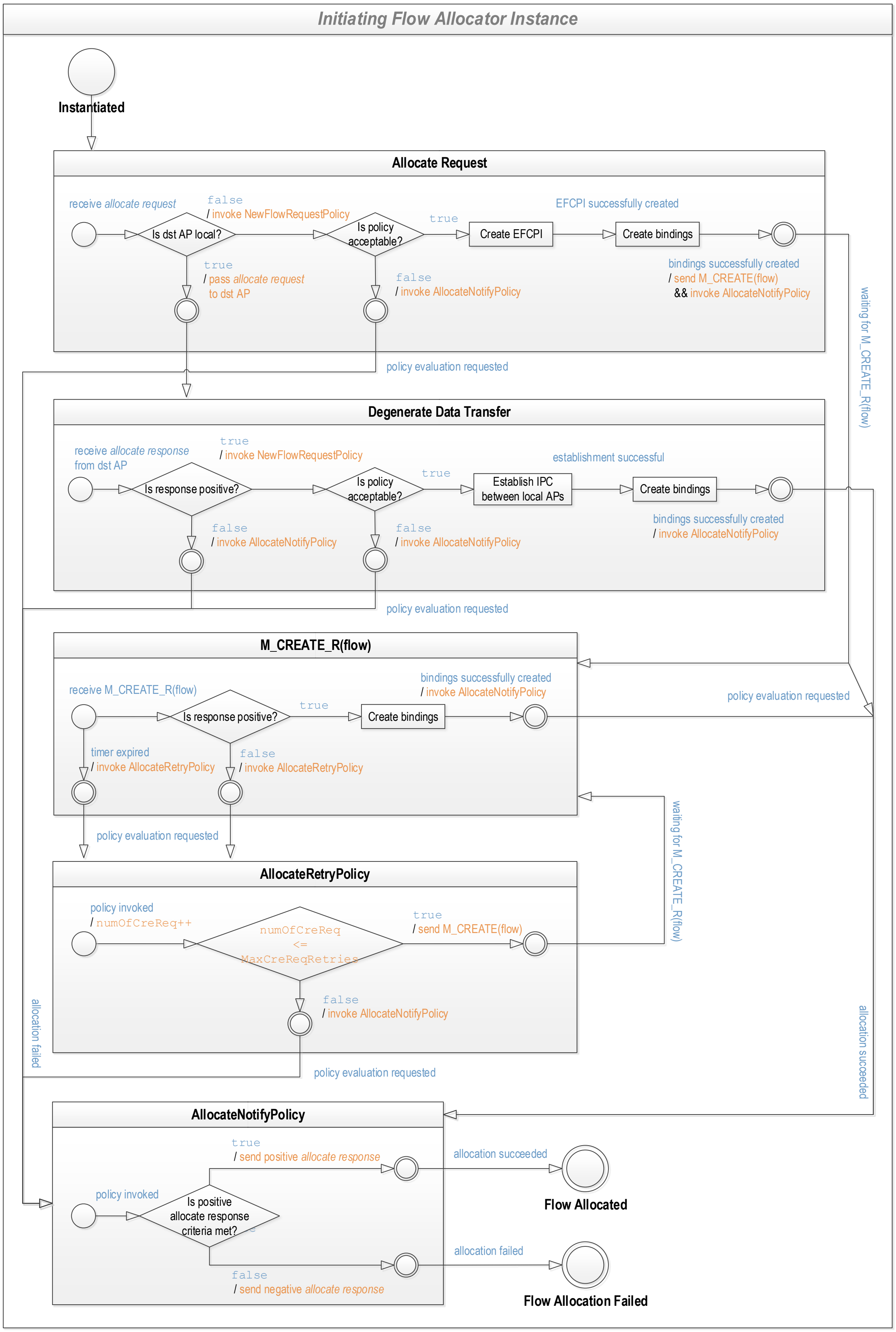

Initiating Flow Allocator Instance
Instantiated
Allocate Request
receive allocate request
false
/ invoke NewFlowRequestPolicy
Is dst AP local?
true
Is policy acceptable?
Create EFCPI
EFCPI successfully created
Create bindings
true
/ pass allocate request to dst AP
false
/ invoke AllocateNotifyPolicy
bindings successfully created
/ send M_CREATE(flow)
&& invoke AllocateNotifyPolicy
waiting for M_CREATE_R(flow)
policy evaluation requested
Degenerate Data Transfer
receive allocate response from dst AP
true
/ invoke NewFlowRequestPolicy
Is response positive?
Is policy acceptable?
true
Establish IPC between local APs
establishment successful
Create bindings
false
/ invoke AllocateNotifyPolicy
false
/ invoke AllocateNotifyPolicy
bindings successfully created
/ invoke AllocateNotifyPolicy
policy evaluation requested
M_CREATE_R(flow)
bindings successfully created
/ invoke AllocateNotifyPolicy
receive M_CREATE_R(flow)
Is response positive?
true
Create bindings
policy evaluation requested
timer expired
/ invoke AllocateRetryPolicy
false
/ invoke AllocateRetryPolicy
policy evaluation requested
waiting for M_CREATE_R(flow)
AllocateRetryPolicy
policy invoked
/ numOfCreReq++
numOfCreReq <= MaxCreReqRetries
true
/ send M_CREATE(flow)
false
/ invoke AllocateNotifyPolicy
allocation failed
policy evaluation requested
allocation succeeded
AllocateNotifyPolicy
true
/ send positive allocate response
policy invoked
Is positive allocate response criteria met?
false
/ send negative allocate response
allocation succeeded
Flow Allocated
allocation failed
Flow Allocation Failed

Fig. 38: Flow Allocator Instance operation of initiating IPCP

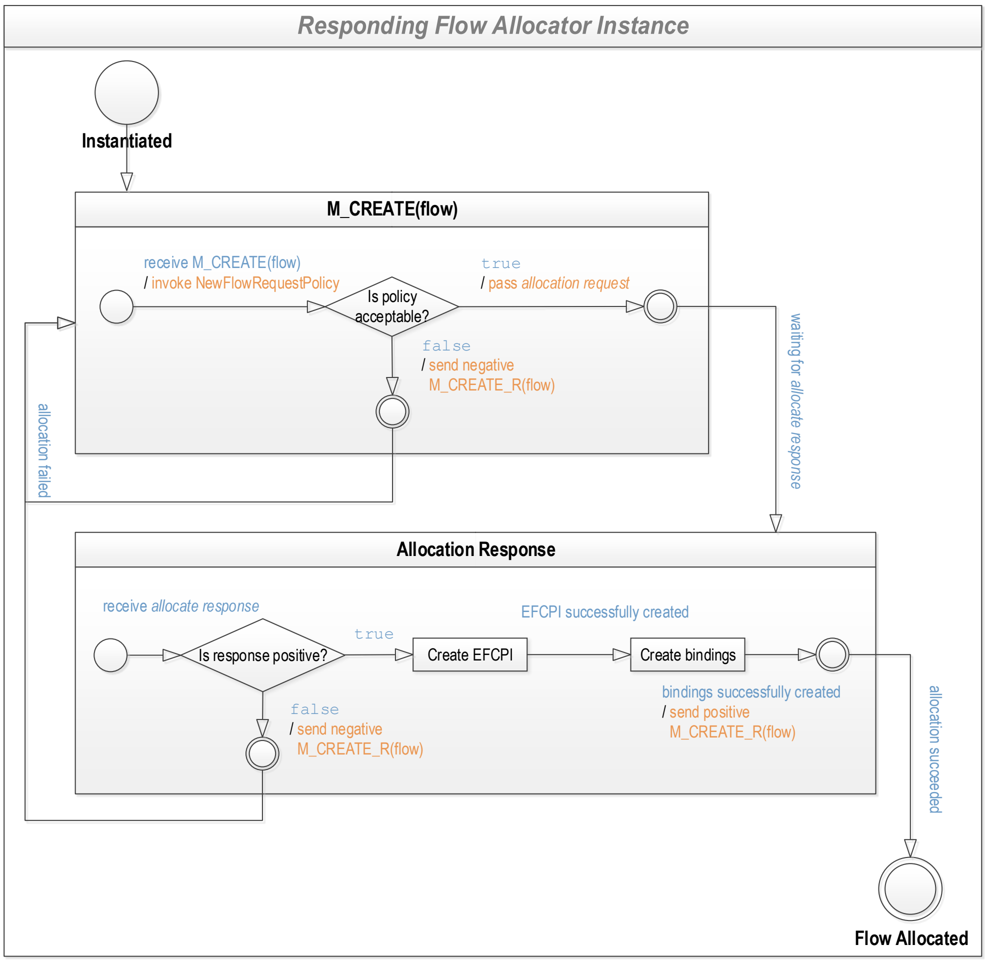

Responding Flow Allocator Instance
Instantiated
M_CREATE(flow)
receive M_CREATE(flow)
/ invoke NewFlowRequestPolicy
Is policy acceptable?
true
/ pass allocation request
false
/ send negative M_CREATE_R(flow)
allocation failed
waiting for allocate response
Allocation Response
receive allocate response
Is response positive?
true
Create EFCPI
EFCPI successfully created
Create bindings
bindings successfully created
/ send positive M_CREATE_R(flow)
false
/ send negative M_CREATE_R(flow)
allocation succeeded
Flow Allocated

Fig. 39: Flow Allocator Instance operation of responding IPCP before the flow was allocated

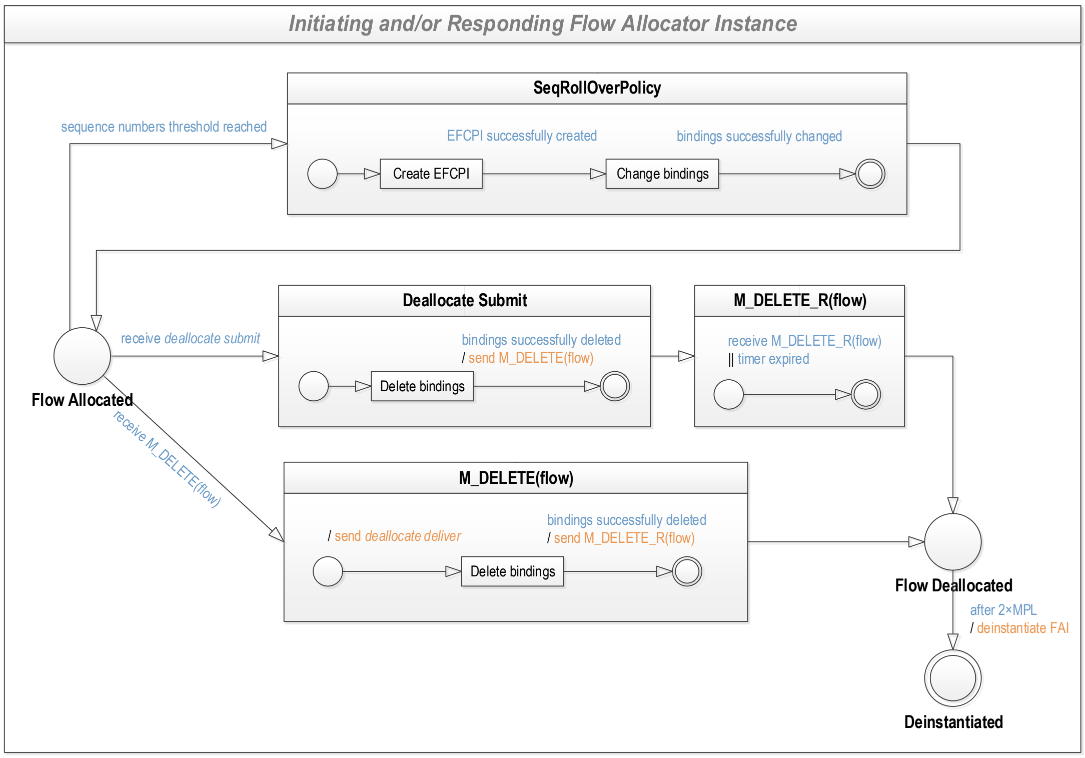


Fig. 40: Flow Allocator Instance operation after the flow was allocated

RIB Daemon

All information maintained by IPC tasks such as FA, RA, and others is available and updated through **RIB Daemon (RIBd)** described in [89] and [90]. Information exchange is necessary to coordinate the distributed IPC. Different update strategies for different kinds of information may be used to synchronize state between different DIF member subsets.

Resource Information Base (RIB) is a logical database of information accessible via RIB Daemon. By logical database, we mean that some of RIB information may be stored in the dedicated database and the rest in IPCP components. Periodic or solicited events can cause RIB to be queried/updated by IPCP peers via management CDAP messages. RIBd provides an API to perform an operation on both local and remote RIB.

Common Distributed Application Protocol

Subsection 5.1.1 postulates that there is only a single application protocol required and this is the **Common Distributed Application Protocol (CDAP)**. DIFs use CDAP for all non-data communication (i.e., IPC management such as maintaining RIB, controlling flow allocation, joining a DIF). DAFs may not use CDAP for backward compatibility. However, CDAP expressiveness should allow the transition of legacy protocols. CDAP is based and patterned on two existing protocols – ACSE (see [91] and [92]) for the establishment phase, CMIP [93] for the data transfer phase.

CDAP subpart for data transfer is object-oriented (with built-in scope and filter support) protocol offering six primitive operations: *create*; *delete*; *read* (i.e., get value); *write* (i.e., put or set value); *start* (i.e., execute action) and *stop* (i.e., suspend action). The collection of objects is dependent on used AE, which provides access rights to them.

CDAP has modular structure composed of three submodules to provide flexibility:

- The common application connection establishment (CACE) submodule;
- The authentication (Auth) submodule provides authentication of the communication endpoints. A range of submodules will be available to support different kinds (e.g., none authentication, shared password, certificates) of authentication policies employing different cryptographic tools (e.g., a-/symmetric ciphers for confidentiality, MAC codes for integrity);
- The CDAP submodule.

CDAP offers following eighteen message types summarized in Tab. 7 [94]:

Opcode	Description
M_CONNECT	Initiate a connection from a source application to a destination application
M_CONNECT_R	Response to *M_CONNECT* carries information or an error indication
M_RELEASE	Orderly close of a connection
M_RELEASE_R	Response to *M_RELEASE* carries final resolution of close operation
M_CREATE	Create an application object
M_CREATE_R	Response to *M_CREATE* carries result of creating request, including identification of the created object
M_DELETE	Delete a specified application object
M_DELETE_R	Response to *M_DELETE* carries result of deletion attempt
M_READ	Read the value of a specified application object
M_READ_R	Response to *M_READ* carries part or all of object value or error indication
M_CANCELREAD	Cancel a prior read issued using *M_READ*.
M_CANCELREAD_R	Response to *M_CANCELREAD* indicates outcome of cancelation
M_WRITE	Write a specified value to a specified application object
M_WRITE_R	Response to *M_WRITE* carries result of write operation
M_START	Start the operation of a specified application object, used when the object has operational and non-operational states
M_START_R	Response to *M_START* indicates the result of the operation
M_STOP	Stop the operation of a specified application object
M_STOP_R	Response to *M_STOP* indicates the result of the operation

Tab. 7: CDAP message types

Connection management between two applications is divided into two traditional phases – establishment and data transfer. An AP issues *allocate request* to underlying DIF's IPCPC specifying the destination APN and QoS requirements. If the allocation is successful, IPCP returns port-id to be used as a handle for all communication leveraging this flow. When the previous phase is completed, CACE sends a *M_CONNECT* message to start authentication using Auth submodule. Additional message exchange might follow in order to support different authentication mechanisms. If it is successful then the connection is established and CDAP transits to data transfer phase.

Another contribution is further refinement of CACE specifications [95]. Detail description of CDAP operation is provided in Figures Fig. 41, Fig. 42 and Fig. 43. Once again transitions are denoted with "input / action" labels. There are three different FSMs. Fig. 41 depicts establishment phase on initiating the process. Fig. 42 shows the same but from the perspective of the responding process. Fig. 43 outlines data transfer phase for both initiator and responder once they successfully reach "Established". For the sake of readability, only correct transitions are shown. Incorrect transitions upon receiving unexpected CDAP message terminate from any state in "Error" marked as "wrong input". Both initiator and responder might "indicate deallocation", thus entering "Deallocating" state at any given moment.

Depending on whether (N-1)-flow should be preserved or not, the transition from "Deallocating" (based on `keepFlow` boolean) may delete any state associated with connection and transit to the "Null" state.

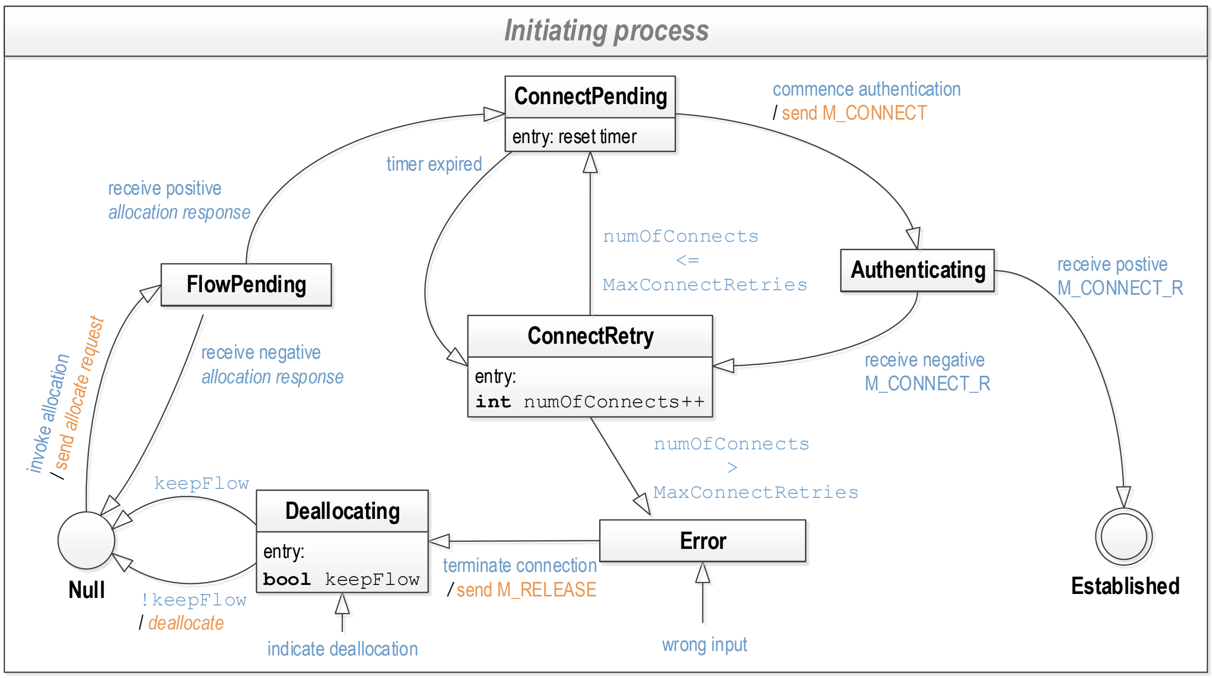


Fig. 41: Establishment phase on initiating process

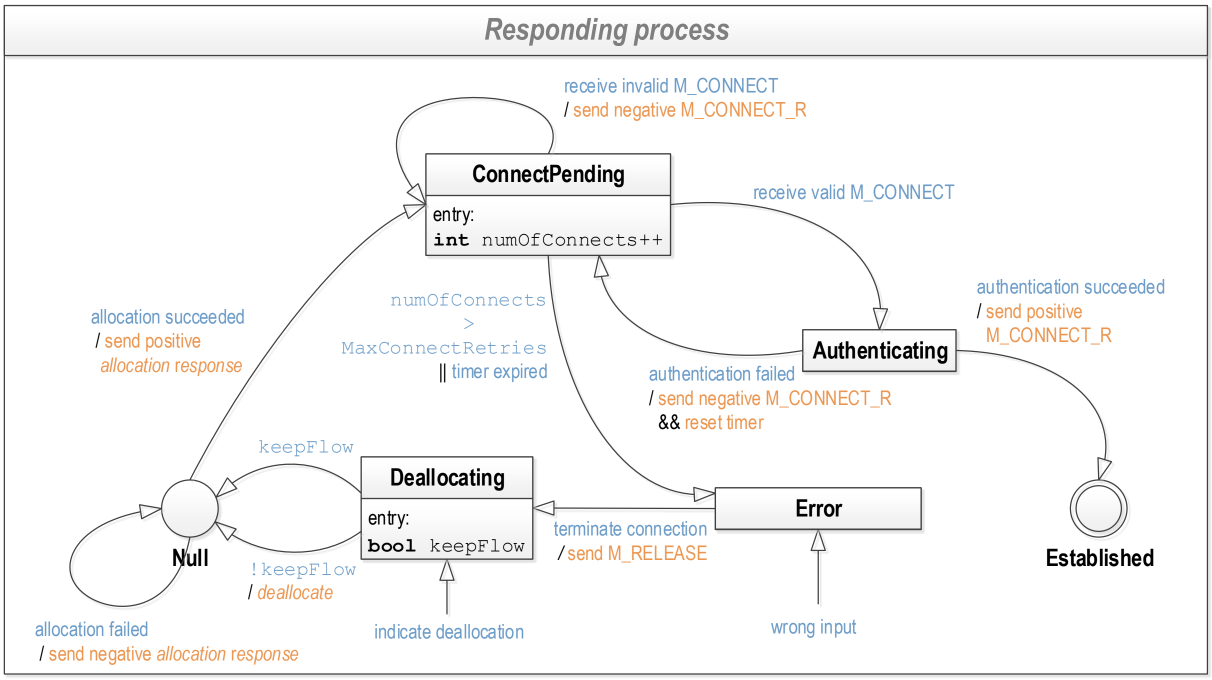


Fig. 42: Establishment phase on responding process

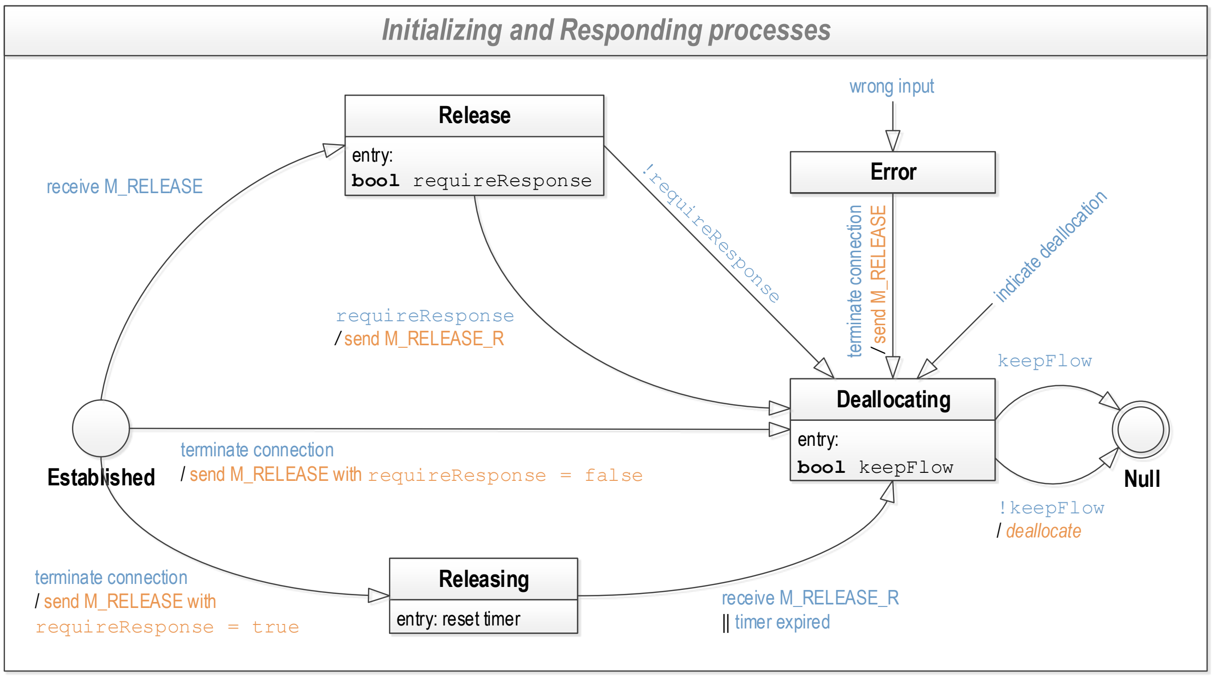


Fig. 43: Data transfer phase on initiating/responding process

5.3 Contribution

Simulation often serves for validating and verifying new technologies, which do not have a yet implementation. The simulation also finds weak points and drawbacks during test runs and subsequently allows one to enhance development process based on feedbacks. Hence, the implementation of the **RINA Simulator (RINASim)** is a natural step to support ongoing research and development of the Recursive Internet Architecture.

We are developing the RINASim in the frame of European project PRISTINE. RINASim is a stand-alone framework for OMNeT++ discrete event simulator environment. RINASim is coded from scratch and independent on another library. The main purpose is to offer the community with reliable and the most up-to-date tool (in the sense of RINA specification compliance) for simulating RINA-based computer networks. Thanks to the OMNeT++'s built-in result analysis and graphical simulation output, RINASim may be used not only for research but also as an educational tool.

This subchapter introduces RINASim installation guideline, development design and description of components interactions. Moreover, it illustrates RINA principles and RINASim functionality on one of the basic examples. Subchapter contains only the most relevant information due to the limited space, for more, please see PRISTINE deliverable 2.4 [96].

5.3.1 Installation

RINASim is developed in OMNeT++ 4.6, but its source codes are fully backward compatible with older OMNeT++ versions that support C+11 language standard and GCC 4.9.2 compiler. All source codes (including master and other thematic branches) are publicly available on the project's GitHub repository [97]. Apart from this official channel, RINASim stable release snapshots are periodically published on Open Source Project repository [98].

RINASim installation is a straightforward process with two phases: 1) importing the project into OMNeT++ IDE; 2) compiling the project, which creates one static library (`librinasimcore` containing simulation core) and one dynamic library (`librinasim` also containing various policies linked together with core).

5.3.2 Design

This subsection provides a general overview of RINASim components design, which includes high-level abstract models of computing systems (like hosts and routers) and also their low-level submodules (like IPCP). In general, a structure of RINASim models follows the structure proposed in the RINA specification. This intentional correspondence enables anyone understanding the RINA specifications to easily orient in RINASim too. Though this structure does not always stand for the most natural representation of RINA concepts in simulation models, it provides a framework for evaluating properties of the architecture and to identify missing or inaccurate information in the original specification. During the design of simulation models, we were able to identify several places where specifications should be refined to provide complete and unambiguous information. Following lines reflect RINASim design relevant to a date of this thesis.

Computing System Modules

RINASim offers a variety of high-level models simulating the behavior of independent computing system. These models can be employed to set quickly up simulation experiments. Through parameterization and extension, it is possible to test different deployments and settings. Based on the RINA specifications, we can distinguish between the following node types:

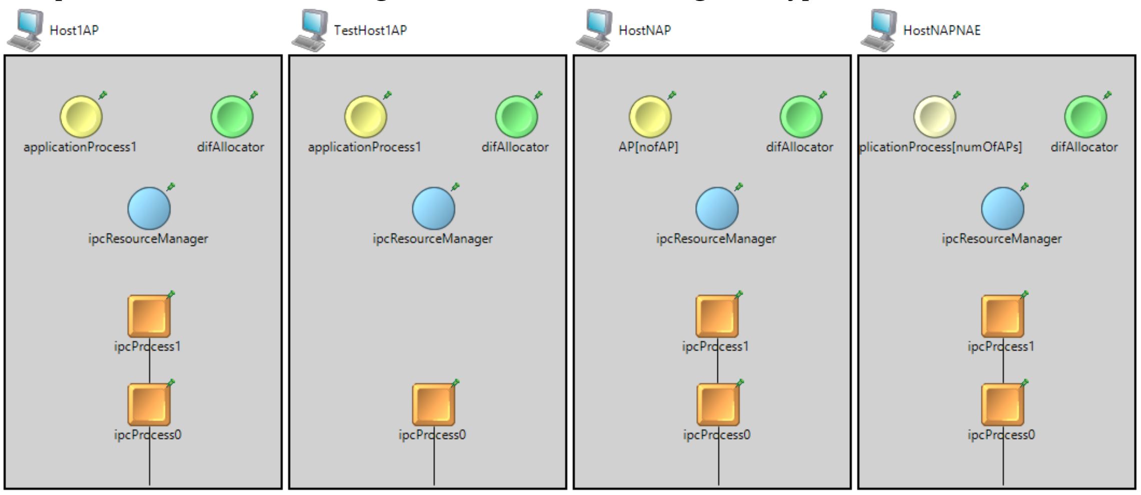


Fig. 44: Host nodes structure examples

- Host nodes, which represent devices or systems that run distributed applications. These nodes implement the full RINA stack and, also, contains an application process(es). AP instances are configured to communicate with each other to simulate the behavior of an arbitrary RINA application. Currently, there are several predefined host nodes depending on a number of APs and AEs. Fig. 44 illustrates some of host nodes internal structure. The most of depicted hosts contain two IPCPs, which models usual end-system with a single NIC. The host may contain only single IPCPs, which would allow IPC with only one directly connected neighbor. Alternatively, host may contain more than two IPCPs; (0)-rank IPCPs represent multiple NICs, and (1+)-rank IPCPs represent different DIFs host memberships;

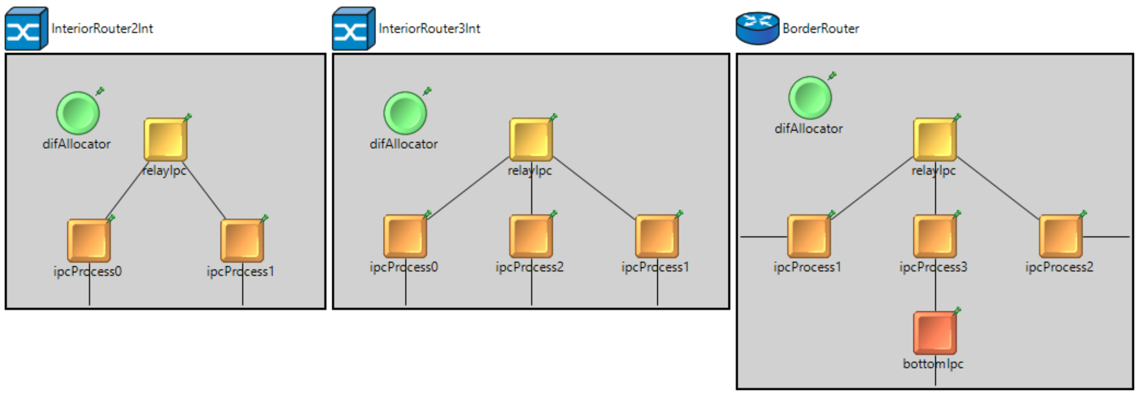


Fig. 45: Router nodes structure examples

- Routers (intermediate nodes), which can be either interior or border. A router is a device that interconnects different underlying DIFs and often does not run user applications. Just as in RINA specification, there are either interior or border routers depending on DIF stack depth (influenced partially also by a number of interfaces). Fig. 45 illustrates two interior routers and one border router simulation models.

Of course, there are many more possible combinations of host and router configurations than the ones currently defined in RINASim. However, the aim of providing predefined node models is not to cover all of the possible combinations but rather to offer the most used ones enabling to set quickly up simulation scenarios. Defining new node or router with required structure is not a complicated task. Nevertheless, the present collection of prepared models seems to be enough.

Policies

RINA specifications present the proposed network architecture as a generic framework, where mechanisms are intended to perform basic common functionality and policies are defined to select the most appropriate implementation of variable functionality. Rather than providing an exhaustive implementation of policies for each parameterized function, RINASim provides interfaces that are used by the core implementation to call functions defined by the selected policies.

The RINASim policy framework is based on OMNeT++ NED module interfaces [99], which helps to minimize the need for modifying existing C++/NED source codes. Instead of placing a simple module with a policy implementation inside the simulation network graph, a placeholder interface module is used. This design allows the potentially unlimited amount of user policy implementations to be defined and easily switchable via the configuration files (by setting a proper parameter of the encompassing module). Each policy consists of an NED module interface and a base C++ class. Fig. 47 shows an example of policy module interfaces (modules with “Policy” suffix in names) with loaded policies (blue labels above them).

DAF Modules

DAF components can be divided into three submodules: a) Application Processes (containing one or more Application Entities), which represents IPC endpoints; b) IPC Resource Manager, which interconnects APs and available IPCPs; c) DIF Allocator, which helps during APN discovery and management process. Components relationship and internal structure (described below) are depicted in Fig. 46.

The applicationProcess module contains applicationEntity submodules for each flow representing the connection between two applications. applicationEntity handles enforcing access control (by evaluating flow allocation requests), flow management and governing application protocol. Each applicationEntity contains iae (submodule interface, which allows pluggable change of application protocols) and the commonDistributedApplicationProtocol submodule that sends and receives messages on behalf of applicationEntity.

The commonDistributedApplicationProtocol submodule provides a simple object-based protocol for distributed applications. Currently, it is the part of RIBd and AE. CDAP is modeled as a compound module consisting of five main submodules:

- cace – Common Application Connection Establishment protocol instance processing *M_CONNECT* and *M_RELEASE* requests and responses;
- auth – providing authentication services during connection initialization); cdap (providing usual CDAP message exchange;
- cdapSplitter – delivering messages to appropriate upper submodules;
- cdapMsgLog – logger for an accounting of processed messages.

The difAllocator module handles locating a destination application based on its name. DA is a component of the DAP's IPC Management that takes ANI and access control information and returns a list of DIF-names through which the requested application is available. Moreover, the difAllocator module provides statically configured knowledge about simulation network graph. The difAllocator modules consists of five auxiliary submodules that maintain state information and help to deliver DA services:

- da – core functionality;
- namingInformation – mapping between APN synonyms;
- directory – mapping between APN and DIF-names;
- searchTable – mapping between APN and peer DA instance where to continue search;
- neighborTable – mapping between peer DA and neighboring DA instances.

The ipcResourceManager module currently queries DA module to find suitable IPCP and relays communication between AE and IPCP. The ipcResourceManager consists of two submodules:

- irm – acting as a broker between APs and IPCPs when handling the flow (de)allocation calls;
- connectionTable – maintaining state information for a given flows.

DIF Modules

All currently implemented DIF components are enclosed to the IPCProcess container module (instantiation of IPCP). The IPCProcess contains following submodules, and overall structure is shown in Fig. 47:

- Enrollment, which governs enrollment of IPCP into DIF;
- Flow Allocator, which processes flow (de)allocation;
- EFCP, which provides data transfer services optionally with transfer control;
- Relaying and Multiplexing module, which handles incoming and outgoing PDUs;
- Resource Allocator, which monitors resources namely (N-1)-flows and available QoS;
- RIBDaemon, which is in charge of processing management messages;
- Routing policy, which maintains PDU forwarding rules.

The enrollment module is in charge of enrollment procedure, which occurs upon successful connection establishment between IPCPs. It consists of core functionality submodule and table (enrollmentTable) maintaining connection state of each enrollment FSM.

The flowAllocator module handles (de)allocation request and response calls from the IRM, RIBDaemon or AE. The flowAllocator module consists of three submodules (and currently three supported policy interfaces):

- `fa` – core functionality involving instantiation of FAIs;
- `nFlowTable` – mapping between (N)-flow and bound FAI;
- `fai_<portId>_<CEPid>` – managing a whole flow lifecycle.

The Error and Flow Control Protocol is modeled as one compound module. This module dynamically spawns `efcpi_<CEPid>` (EFCP instance) and `delimiting` submodules per one flow. There is also the `efcpTable` module maintaining bindings between Delimiting and EFCPI. Apart from that, the `MockEFCPI` processes management PDUs sent/received by local RIBDaemon. Each EFCPI contains the `dtp` submodule (providing data transfer services), the `dtpState` submodule (maintaining state-vector) and a few policies related to DTP functionality. Optionally, EFCPI may also contain the `dtcp` submodule and several DTCP policies, whenever transfer control is requested for a communication (i.e., due to the reliable transmission demand).

The `relayAndMux` module represents a stateless function that takes incoming PDUs and relay them within current IPC or pass them to an outgoing port. In particular the RMT takes PDUs from (N-1)-ports, consults their address fields and perform one of the following actions: a) relay PDU between (N-1)-ports; b) pass PDU to EFCPI; and c) multiplex PDU from EFCPI to (N-1)-port.

The `relayAndMux` consists of multiple simple modules of various types, some of them are static, and some of them are instantiated dynamically at runtime. Among dynamically created modules are RMT ports (representing (N-1)-flow communication endpoints) and associated input/output queues. Among static submodules are:

- `rmt` – core functionality;
- `allocator` – managing addition, removal and reconfiguration of RMT queues and ports;
- `pduForwardingPolicy` – mapping table of destination addresses and QoS-ids to output ports that is used by the relaying functionality of the RMT;
- other policy module interfaces monitoring queue lengths and scheduling PDU departures.

The `resourceAllocator` monitors the operation of the IPCP and makes adjustments to its operation to keep it within the specified operational range. Its forwarding and queuing functionality are customizable by policies. The `resourceAllocator` consists of multiple simple modules of various types, namely:

- `ra` – core functionality that manages connections to other local IPCPs with the help of `nm1FlowTable` submodule;
- `pduFwdGenerator` – uses custom policies to manage `pduForwardingPolicy` entries;
- other policies executed upon RMT queue allocation.

The `ribDeamon` is the IPCP's management heart. It receives/sends CDAP management messages and notifies other submodules about management changes. RINASim's RIBDaemon consists of three submodules:

- `ribd` – core functionality mainly listening to calls from other DIF components and notifying them upon CDAP message reception;
- `commonDistributedApplicationProtocol` – same submodule as in case of DAF components description;
- `ribdSplitter` – splitter is delegating CDAP management messages to/from the `mockEFCPI` or appropriate EFCPIs.

The `routingPolicy` module is used by `pduFwdGenerator` to populate correctly/update the `pduForwardingPolicy`.

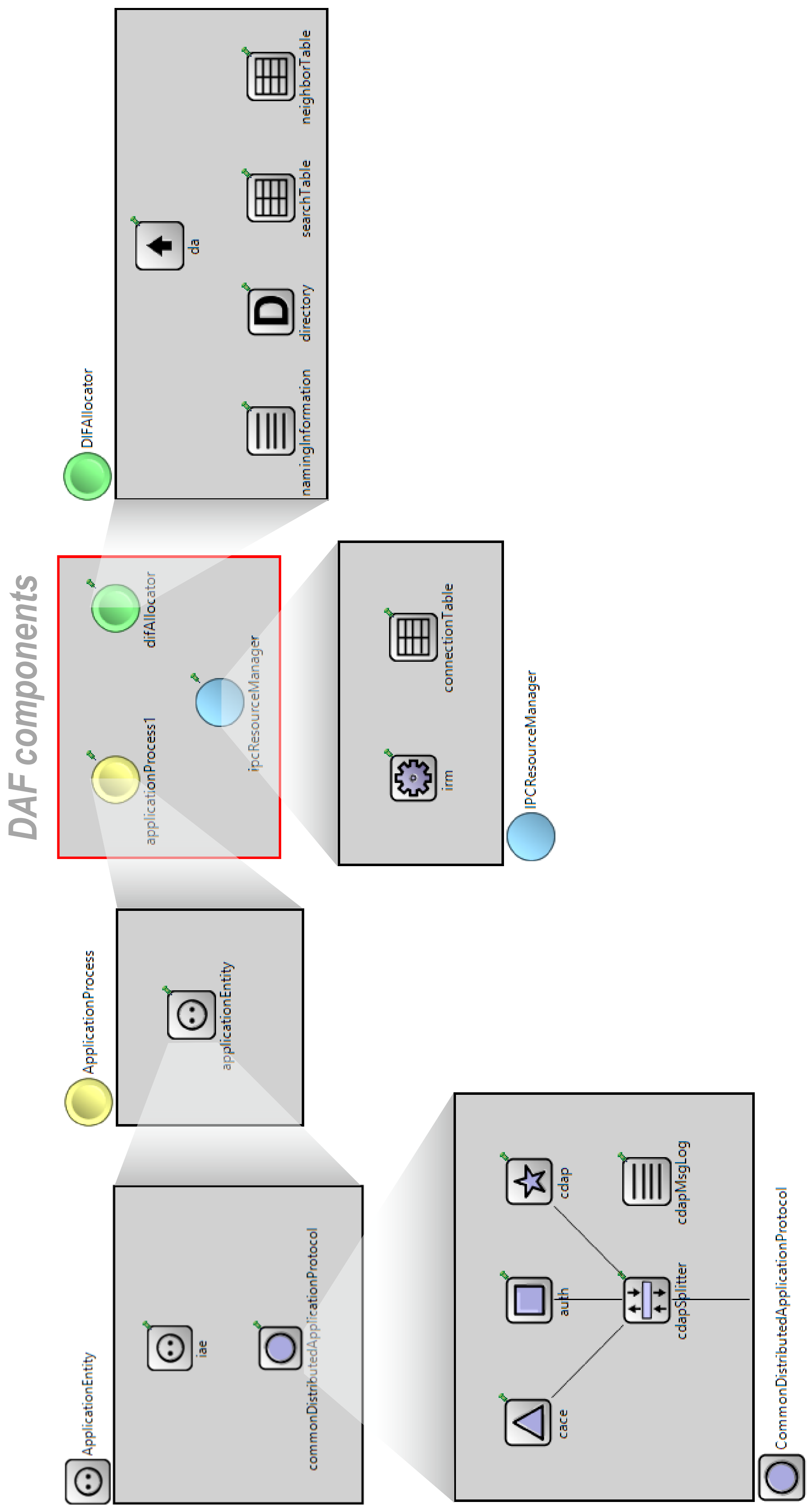


Fig. 46: DAF components for RINASim

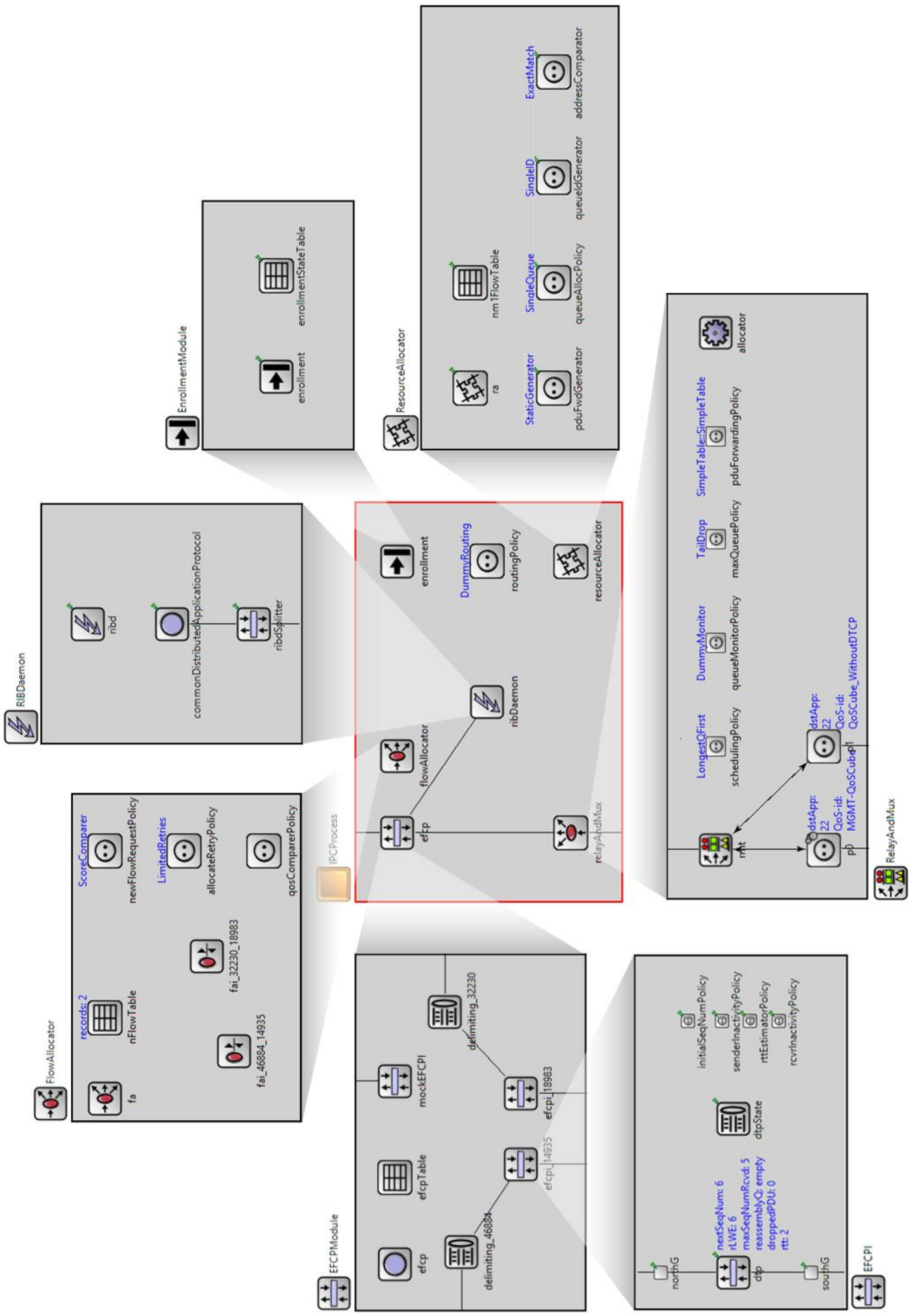


Fig. 47: IPCP's DIF components for RINASim

5.4 Chapter Summary

In this chapter, we described core RINA principles. We tried to summarize RINA theory in the text that lacks any usage of the term without previous thorough definition / context explanation because we know, how hard the “mental shift” from TCP/IP concepts towards RINA is.

The second subchapter went into more details about various RINA components. It started with a description of different kinds of high-level RINA nodes including hosts, interior routers, and border routers. Subsequently, we dived deep into low-level RINA components that are being used by DIF and DAF. Besides that as the research contribution, we thoroughly analyzed and enhanced (particularizing functional descriptions and equipping them with FSMs) RINA specifications namely for FA and CACEP operation.

The last subchapter described RINASim including installation guideline, design notes, and demonstration. RINASim philosophy benefits from clever OMNeT++ module interfacing in order to allow flexible change of used policies. Moreover, Subchapter 5.3 ending contained a thorough illustration of RINA principles using RINASim demo scenario. Demonstration description should show the impact of recursion and help others to understand enrollment and flow (de)allocation procedures in praxis. Moreover, demonstration setup may be employed as the template when creating new scenarios.

We have designed and implemented RINASim as the first full-scale RINA simulator containing a wide gamut of functions that are extensible and replaceable. RINASim reliably proves following RINA properties: isolation of *namespaces* and address spaces across DIFs; enrollment and flow allocation recursion and their impact; routing based on available resources reflecting QoS attributes; easy application protocol prototyping when employing CDAP messages (and action primitives they substitute); and others. Hence, RINA offers by design complete naming scheme and fulfills most of the ideal solution properties as described in Chapter 3.

The main contribution of this chapter is RINASim as a tool that helps: 1) researchers to prototype and test new policies and mechanisms in native and full-compliant RINA environment; 2) others to visualize and understand RINA principles.

6 Conclusion

–"*A story has no beginning or end: arbitrarily one chooses that moment of experience from which to look back or from which to look ahead.*" Graham Green

What has been done and accomplished in frame of this dissertation thesis?

What are the important results?

We pursue a difficult and complex task to define and to discuss elementary naming, addressing and routing principles of computer networks.

The thesis begins with an overview of networking fundamentals and points out design issues of traditional TCP/IP stack that are becoming more apparent as more users and devices are accessing the Internet each day. We tried to qualify causes and quantify their (future) impact (when following current trends). The Internet developed incrementally throughout previous 40 years. However, Internet struggles to redesign its communication schemes after the adoption of TCP/IP and its global expansion.

We collected and studied relevant papers and works written on the topic of naming, addressing and routing. We formulated low-level foundations using formal math apparatus. We compiled encompassing high-level theory and checked its compliance among existing addressing and naming techniques. This work allowed us to reevaluate problems of current Internet in the new light, which confirmed that abovementioned problems of TCP/IP are consequences of incomplete architecture that lacks necessary levels of indirection. We investigated properties of existing candidates, which aspire to deal with this situation. We decided to follow LISP and RINA further with our research efforts.

We thoroughly analyzed LISP use-cases and protocol details (namely the split of locator *address space* and identifier *namespace*). We were able to identify and investigate certain shortcomings of LISP design. Based on that, we developed improvements to LISP operations and verified them using discrete event simulator. We implemented the first low-level LISP simulation modules and successfully checked their compliance with the referential Cisco implementation in the real network. The principle of our LISP research is included in papers [100], [101] and [102].

We conducted a similar analysis of RINA and its properties that aim to the clean-slate design of not only naming and addressing but also other aspects of computer networking. We revisited all available RINA specifications and try to improve their clarity, particularly parts describing enrollment and flow (de)allocation procedures. Subsequently, we designed and implemented the first RINA discrete event simulator called RINASim, which provides a standalone framework with full-fledged RINA simulation modules for OMNeT++. The core contribution of our RINA research has been published as an independent framework in [103] and explained in PRISTINE Deliverable 2.4 [96] and Deliverable 2.6 [104].

Following two subchapters outline some conclusions and results of our research efforts involving Locator/Id Separation Protocol and Recursive InterNetwork Architecture.

6.1 Summary about LISP

Precise LISP (and VRRP) simulation modules for OMNeT++, which are used as the basis for ongoing research, represent the main code contribution. Based on well-known designed issues (see [108]), we investigated, proposed, implemented and tested two improvements – map-cache synchronization and merged RLOC probing. Our map-cache synchronization techniques minimize map-cache misses, thus significantly decreasing packet loss. Furthermore, employing our merged RLOC probing algorithms has an outstanding impact on LISP protocol overhead comparing to simple RLOC probing per every EID.

Despite the accomplished achievements in LISP operation tuning, LISP is unfortunately not an ultimate solution for current Internet troubles. It breaks several RFC 1958 concepts, and some problems were revealed during its worldwide deployment (RFC 7215 [105]). Moreover, LISP deployment needs additional configuration effort to secure LISP against possible attacks and threats (see [106]).

Basically, any solution decoupling locator and identifier has to deal with Locator Path Liveness problem, and any non-host-based loc/id split has to cope with Site-based State Synchronization problem. Their impact can be diminished (with for instance map-cache synchronization described above) but not completely treated. Hence, neither LISP nor any CES/CEE proposal reviewed in Chapter 3.3 is the desired solution.

Another and probably the most serious rebuke of any hybrid or network-based loc/id split is when a packet is traversing locator *namespace* then the routing is performed according to the locator, not an identifier. Previous is strictly in contradiction to the theory reviewed in Chapter 3, and implications are thoroughly investigated in [68]. LISP suffers from three major problems:

1) Routing should be done based on *node names* (see Saltzer's [36]). However, "routes" in nowadays Internet use PoAs. Therefore, all IP "routing" is based on false premises and would always be *route dependent* (which is unwanted based on knowledge in Subchapter 3.2). Routing should be performed based on identifier not locator (otherwise, it leads to Locator Path Liveness problem);
2) Locator and identifier are not *bound* to the same *object* – locator address is an address of the interim device (which performs header alternation relevant to loc/id split) not the end-device of communication;
3) All identifiers are used in some sense also for locating. An *object* cannot be *located* without *identifying* it and vice versa (see Saltzer's [32]). There could neither be identification without localization, nor localization without identification. Thus, there should be no semantic distinction between identifier and locator on the Internet but yet there is.

Therefore, LISP does not provide proper naming and addressing concept, nor it is even scalable routing solution for TCP/IP architecture.

6.2 Summary about RINA

RINA as the new (and complete) clean-slate architecture tries to touch and codify every part of communication within computer networks. Therefore, RINA's knowledge base spans from high-level reference model description to low-level characterization of each component functionality. Pouzin Society [107] is a formal body in charge of maintaining specifications with FIT-BUT as one of its members. In the theoretical part of this dissertation, we revisited and extended parts of RINA specifications concerning flow allocation and connection establishment procedure. We supplemented them with FSMs illustrating FA and CACE operations.

RINASim is the main contribution, and RINASim's development process helped to clarify and progress some RINA specifications. As the RINASim's chief designers and implementers, we authored FA, DA, AE, RIBd and RA simulation modules in the frame of this thesis.

RINA is still young in its technological readiness level. Hence, some of RINA's concepts were doubtful whether they will work or not. Following RINA features would not be possible to prove or verify without RINASim:

- We simulated and shown basic RINA functionality (enrollment, flow allocation and data transfer) in this thesis (and in [103]). RINA can achieve IPC employing recursively the same (DIF and DAF) components, which simplifies implementation of the network stack. Furthermore, DIF scope isolation allows reusing IPCP's APNs without any duplicity address problems. Hence, there is no need for global address space due to the DIF isolation;
- RINA allows an easy employment of **Aggregated Congestion Control (ACC)**, see PRISTINE Deliverable 3.2 [108] for more. ACC improves QoS experience for communicating parties whenever congestion occurs in the network. RINA offers built-in mechanisms with programmable policies to handle resource allocation in compliance with QoS demands;
- Custom routing algorithm taking into account division of (*location dependent*) *address space* reduces significantly routing table sizes for distributed cloud installations. Solution above – called **Scalable Forwarding with RINA (SFR)**, see paper [109] for details – provides proofs for real-life use-case that *topologically dependent* (hierarchical) *addresses* help in routing comparing to flat *address space*.

RINA theory seems to offer complete naming, addressing and routing concepts. Moreover, RINA's design separating mechanisms and policies is flexible enough to allow scalable changes reflecting demands of future Internet. Nevertheless, RINA needs more validation and verification testing (preferably) on real-life deployment to support previous claims.

6.3 Future Work

We take this thesis just as the beginning of more advanced research involving OMNeT++ simulator as a validation tool for new routing paradigms (such as LISP) and alternative architectures (such as RINA).

We would like to discuss our LISP improvements – map-cache synchronization and merged RLOC probing – within IETF to see whether they can be submitted as draft proposals. Our plans with LISP simulation modules include to add support for proxy xTR functionality and to recognize more LISP control flags (like SMR bits). We would like to use further our LISP simulation modules and test effectiveness of different distributed mapping systems (e.g., LISP-ALT, LISP-DDT). Also, we intend to upgrade VRRP to support IPv6 addresses and all features of VRRP version 3. We would like our low-level LISP simulation modules to be considered as the verification tool for other LISP related use-cases and technologies. Therefore, we want to integrate LISP source codes with official INET framework as the first step (which is something we already accomplished before [173] or [174]).

We plan to carry on work on RINA research topics and further refine RINASim based on new knowledge and up-to-date specifications. An additional goal is to conduct a comparative evaluation of our simulation models with RINA implementation for Linux environment called IRATI. Adoption of the newest version 6.5 of EFCP, SDU protection module integration, NSM and dynamic DA functionality are on our development roadmap for the nearest future.

7 Bibliography

[1] B. Carpenter, "RFC 1958: Architectural Principles of the Internet," June 1996. [Online]. Available: http://tools.ietf.org/html/rfc1958.

[2] Cisco Systems, Inc., "The Zettabyte Era Trends and Analysis - Cisco," May 2015. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html. [Accessed September 2015].

[3] M. Grégr and T. Podermanski, "IPv6 @ CESNET nework | 6lab.cz," Brno University of Technology, September 2015. [Online]. Available: http://6lab.cz/live-statistics/ipv6-cesnet-nework/. [Accessed September 2015].

[4] IRTF, "Routing Research Group (RRG)," [Online]. Available: https://irtf.org/concluded/rrg. [Accessed February 2015].

[5] IRTF, "rrg Discussion Archive - Date Index," [Online]. Available: http://www.ietf.org/mail-archive/web/rrg/current/maillist.html. [Accessed February 2015].

[6] IETF, "Open discussion forum for long/wide-range architectural issues Discussion Archive - Date Index," [Online]. Available: http://www.ietf.org/mail-archive/web/architecture-discuss/current/maillist.html. [Accessed February 2015].

[7] T. Li, "RFC 6227: Design Goals for Scalable Internet Routing," May 2011. [Online]. Available: http://tools.ietf.org/html/rfc6227.

[8] D. Meyer, L. Zhang and K. Fall, "RFC 4984: Report from the IAB Workshop on Routing and Addressing," September 2007. [Online]. Available: http://tools.ietf.org/html/rfc4984.

[9] G. Huston, "BGP Growth Revisited," November 2011. [Online]. Available: http://www.potaroo.net/ispcol/2011-11/bgp2011.html.

[10] G. Huston, "BGP in 2014," January 2015. [Online]. Available: http://www.potaroo.net/ispcol/2015-01/bgp2014.html.

[11] G. Huston, "Addressing 2014 - And then there were 2!," January 2015. [Online]. Available: http://www.potaroo.net/ispcol/2015-01/addressing2014.html.

[12] Y. Rekhter, T. Li and S. Hares, "RFC 4271: A Border Gateway Protocol 4 (BGP-4)," January 2006. [Online]. Available: http://tools.ietf.org/html/rfc4271.

[13] G. Huston, "BGP Reports - BGP Table Data," 7 August 2013. [Online]. Available: http://bgp.potaroo.net/index-bgp.html.

[14] D. Mowery and T. Simcoe, "Is the Internet a US invention?—an economic and technological history of computer networking," *Research Policy,* vol. 31, no. 8-9, pp. 1369-1387, December 2002.

[15] M. Boucadair and D. Binet, Solutions for Sustaining Scalability in Internet Growth, France: IGI Global, 2014.

[16] B. Carpenter, J. Crowcroft and Y. Rekhter, "RFC 2101: IPv4 Address Behaviour Today," February 1997. [Online]. Available: http://tools.ietf.org/html/rfc2101.

[17] S. Brim, "LISP Analysis," March 2008. [Online]. Available: https://tools.ietf.org/html/draft-brim-lisp-analysis-00.

[18] J. Abley, K. Lindqvis, E. Davies, B. Black and V. Gill, "IPv4 Multihoming Practices and Limitations," [Online]. Available: http://tools.ietf.org/html/rfc4116.

[19] T. Bates and Y. Rekhter, "RFC 2260: Scalable Support for Multi-homed Multi-provider Connectivity," January 1998. [Online]. Available: http://tools.ietf.org/html/rfc2260.

[20] J. Day, Patterns in Network Architecture: A Return to Fundamentals, Boston: Prentice Hall, 2008.

[21] Postscapes, "Internet of Things Market Forecast," [Online]. Available: http://postscapes.com/internet-of-things-market-size. [Accessed February 2015].

[22] C. Perkins, "RFC 5944: IP Mobility Support for IPv4, Revised," November 2010. [Online]. Available: http://tools.ietf.org/html/rfc5944.

[23] C. Perkins, D. Johnson and J. Arkko, "RFC 6275: Mobility Support in IPv6," July 2011. [Online]. Available: https://tools.ietf.org/html/rfc6275.

[24] H. Soliman, C. Castelluccia, K. ElMalki and L. Bellier, "RFC 5380: Hierarchical Mobile IPv6 (HMIPv6) Mobility Management," October 2008. [Online]. Available: https://tools.ietf.org/html/rfc5380.

[25] IETF, "Multipath TCP (mptcp)," [Online]. Available: https://datatracker.ietf.org/wg/mptcp/documents/. [Accessed February 2015].

[26] J. Saltzer, D. Reed and D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems (TOCS)* , vol. 2, no. 4, pp. 277-288, 1984.

[27] Cisco Systems, Inc., "Document ID 13753: BGP Best Path Selection Algorithm," 21 May 2012. [Online]. Available: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094431.shtml.

[28] B. Carpenter, R. Atkinson and H. Flinck, "Renumbering Still Needs Work," May 2010. [Online]. Available: http://tools.ietf.org/html/rfc5887.

[29] G. Huston, "The BGP World is flat," November 2011. [Online]. Available: http://www.potaroo.net/ispcol/2011-12/flat.html. [Accessed July 2015].

[30] J. Shoch, "IEN #19: A note on Inter-Network Naming, Addressing, and Routing," XEROX PARC, January 1978. [Online]. Available: http://www.postel.org/ien/pdf/ien019.pdf.

[31] C. Sunshine, "IEN #178: Addressing Problems in Multi-Network Systems," University of Southern California, April 1981. [Online]. Available: http://www.postel.org/ien/pdf/ien178.pdf.

[32] J. Saltzer, "Name Binding of Objects," Massachusetts Institute of Technology, 1978. [Online]. Available: web.mit.edu/Saltzer/www/publications/nbo/nbo.pdf.

[33] J. N. Chiappa, "Endpoints and Endpoint Names: A Proposed Enhancement to the Internet Architecture," 1999. [Online]. Available: http://www.chiappa.net/~jnc/tech/endpoints.txt.

[34] J. Munkres, Topology: A First Course, Prentice Hall College Div, 1974.

[35] ISO, "Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing". Patent ISO/IEC 7498-3:1997, 1997.

[36] J. Saltzer, "On the Naming and Binding of Network Destinations," *Local Computer Networks,* pp. 311-317, August 1982.

[37] F. Baker, E. Lear and R. Droms, "RFC 4192: Procedures for Renumbering an IPv6 Network without a Flag Day," [Online]. Available: http://tools.ietf.org/html/rfc4192.

[38] T. Li, "RFC 6115: Recommendation for a Routing Architecture," February 2011. [Online]. Available: http://tools.ietf.org/html/rfc6115.

[39] R. Hinden, "RFC 1955: New Scheme for Internet Routing and Addressing (ENCAPS) for IPng," June 1996. [Online]. Available: http://tools.ietf.org/html/rfc1955.

[40] R. Smart and D. Clark, "[RRG] GSE History," January 1995. [Online]. Available: http://www.ietf.org/mail-archive/web/rrg/current/msg02455.html.

[41] M. O'Dell, "GSE: The Alternative Addressing Architecture for IPv6," February 1997. [Online]. Available: http://tools.ietf.org/html/draft-ietf-ipngwg-gseaddr-00.

[42] D. Jen, M. Meisel, H. Yan, D. Massey, L. Wang, B. Zhang and L. Zhang, "Towards A New Internet Routing Architecture: Arguments for Separating Edges from Transit Core," 2008.

[43] D. Farrinaci, V. Fuller, D. Meyer and D. Lewis, "RFC 6830: The Locator/ID Split Protocol (LISP)," January 2013. [Online]. Available: http://tools.ietf.org/html/rfc6830.

[44] R. Moskowitz and P. Nikander, "RFC 4423: Host Identity Protocol (HIP) Architecture," May 2006. [Online]. Available: http://tools.ietf.org/html/rfc4423.

[45] E. Nordmark and M. Bagnulo, "RFC 5533: Shim6: Level 3 Multihoming Shim Protocol for IPv6," June 2009. [Online]. Available: http://tools.ietf.org/html/rfc5533.

[46] X. Xu, "Routing Architecture for the Next Generation Internet (RANGI)," August 2010. [Online]. Available: http://tools.ietf.org/html/draft-xu-rangi-04.

[47] R. Whittle, "Ivip (Internet Vastly Improved Plumbing) Architecture," March 2010. [Online]. Available: http://tools.ietf.org/html/draft-whittle-ivip-arch-04.

[48] P. Frejborg, "RFC 6306: Hierarchical IPv4 Framework," July 2011. [Online]. Available: http://tools.ietf.org/html/rfc6306.

[49] Y. Wang, W. Zhang and J. Bi, "Name overlay (NOL) Service for Improving Internet Routing Scalability," Venice, Italy, July, 2010.

[50] M. Menth, M. Hartmann and D. Klein, "Global Locator, Local Locator, and Identifier Split (GLI-Split)," *Future Internet 2013,* vol. V, no. 1, pp. 67-94, January, 2013.

[51] J. Adan, "Tunneled Inter-domain Routing (TIDR)," November 2006. [Online]. Available: http://tools.ietf.org/html/draft-adan-idr-tidr-01.

[52] R. Atkinson and S. Bhatti, "RFC 6740: Identifier-Locator Network Protocol (ILNP) Architectural Description," November 2012. [Online]. Available: http://tools.ietf.org/html/rfc6740.

[53] J. Ubillos, M. Xu, Z. Ming and C. Vogt, "Name-Based Sockets Architecture," September 2010. [Online]. Available: http://tools.ietf.org/html/draft-ubillos-name-based-sockets-03.

[54] D. Jen, M. Meisel, H. Yan, D. Massey, L. Wang, B. Zhang and L. Zhang, "APT: A Practical Transit Mapping Service," November 2007. [Online]. Available: http://tools.ietf.org/html/draft-jen-apt-01.

[55] F. Templin, "RFC 5720: Routing and Addressing in Networks with Global Enterprise Recursion (RANGER)," February 2010. [Online]. Available: http://tools.ietf.org/html/rfc5720.

[56] W. Herrin, "Tunneling Route Reduction Protocol (TRRP)," [Online]. Available: http://bill.herrin.us/network/trrp.html.

[57] C. Vogt, "Six/One Router: A Scalable and Backwards Compatible Solution for Provider-Independent Addressing," Seattle, USA, 2008.

[58] D. Farinacci, V. Fuller, D. Meyer and D. Lewis, "RFC 6830: The Locator/ID Separation Protocol (LISP)," January 2013. [Online]. Available: http://tools.ietf.org/html/rfc6830.

[59] D. Lewis, D. Meyer, D. Farinacci and V. Fuller, "RFC 6832: Interworking between Locator/ID Separation Protocol (LISP) and Non-LISP Sites," January 2013. [Online]. Available: http://tools.ietf.org/html/rfc6832.

[60] V. Fuller, "RFC 6833: Locator/ID Separation Protocol (LISP) Map-Server Interface," January 2013. [Online]. Available: http://tools.ietf.org/html/rfc6833.

[61] L. Iannone, D. Saucez and O. Bonaventure, "RFC 6834: Locator/ID Separation Protocol (LISP) Map-Versioning," January 2013. [Online]. Available: http://tools.ietf.org/html/rfc6834.

[62] F. Coras, A. Cabellos and L. Jakab, "CoreSim: A Simulator for Evaluating LISP Mapping Systems," Cluj-Napoca, 2009.

[63] A. Cabellos, J. Domingo Pascual, D. Saucez and O. Bonaventure, "Validation of a LISP simulator," 2011. [Online]. Available: http://upcommons.upc.edu/e-prints/bitstream/2117/14351/1/Cabellos.pdf.

[64] D. Klein, M. Hoefling, M. Hartmann and M. Menth, "Integration of LISP and LISP-MN into INET," in *Proceedings of the IEEE 5th International ICST Conference on Simulation Tools and Techniques*, Desenzano del Garda, 2012.

[65] D. Klein, M. Hartmann and M. Menth, "NAT Traversal for LISP Mobile Node," July 2010. [Online]. Available: http://tools.ietf.org/html/draft-klein-lisp-mn-nat-traversal.

[66] J. Kim, L. Iannone and A. Feldmann, "A deep dive into the LISP cache and what ISPs should know about it," *NETWORKING 2011,* vol. 6640, no. ISBN: 978-3-642-20756-3, pp. 367-378, 2011.

[67] D. Meyer and D. Lewis, "Architectural Implications of Locator/ID Separation," January 2009. [Online]. Available: http://tools.ietf.org/html/draft-meyer-loc-id-implications-01.

[68] J. Day, "Why Loc/Id Split Isn't the Answer," Pouzin Society, 2008. [Online]. Available: http://pouzinsociety.org/images/LocIDSplit090309.pdf.

[69] D. Saucez, O. Bonaventure, L. Iannone and C. Filsfils, "LISP ITR Graceful Restart," December 2013. [Online]. Available: https://tools.ietf.org/html/draft-saucez-lisp-itr-graceful-03.

[70] D. Saucez, J. Kim, L. Iannone, O. Bonaventure and C. Filsfils, "A Local Approach to Fast Failure Recovery of LISP Ingress Tunnel Routers," *NETWORKING 2012,* vol. 7289, pp. 397-408, 2012.

[71] J. Day, "RINARefModelPart1-0 130925: Part 1 - Basic Concepts of Distributed Systems," Pouzin Society, 2013.

[72] J. Day, "RINARefModelPart2-1 130925: Part 2 - Distributed Applications, Chapter 1 - Basic Concepts of Distributed Applications," Pouzin Society, 2013.

[73] J. Day and E. Trouva, "RINARefModelPart2-2 140102: Part 2 - Distributed Applications, Chapter 2 - Introduction to Distributed Management Systems," Pouzin Society, 2014.

[74] J. Day, "RINARefModelPart3-1 140102: Part 3 - Distributed InterProcess Communication, Chapter 1 - Fundamental Structure," Pouzin Society, 2012.

[75] J. Day, "RINARefModelPart3-2 140102: Part 3 - Distributed InterProcess Communication, Chapter 2 - DIF Operations," Pouzin Society, 2012.

[76] J. Day, "An introduction to the Recursive InterNetwork Architecture," January 2015. [Online]. Available: http://ict-pristine.eu/wp-content/uploads/2014/12/GhentIntroRINAPt1-150119.pdf. [Accessed April 2015].

[77] J. Day, I. Matta and K. Mattar, "Networking is IPC: a guiding principle to a better internet," in *CoNEXT '08 Proceedings of the 2008 ACM CoNEXT Conference* , New York, NY, USA, 2008.

[78] R. Watson, "Delta-t Protocol Specification," Lawrence Livermore Laboratory, December 1981. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/5542785.

[79] R. Watson, "The Delta-t transport protocol: features and experience," in *Proceedings 14th Conference on Local Computer Networks*, Minneapolis, USA, 1989.

[80] E. Trouva, E. Grasa, J. Day and S. Bunch, "Layer discovery in RINA networks," in *IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD),* Barcelona, Spain, 2012.

[81] J. Day, "D-Base-2011-017: IPC Resource Manager (IRM) Specification," Pouzin Society, 2012.

[82] J. Day, "D-Base-2012-014: Basic Enrollment Specification," Pouzin Society, 2012.

[83] J. Day, "D-Base-2010-007: Delimiting Module," Pouzin Society, 2009.

[84] J. Day, "DelimitingGeneral130904: Delimiting Module," Pouzin Society, 2013.

[85] J. Day, M. Marek, L. Bergesio and M. Tarzan, "EFCPSpec140824_MT_LBJD_MM_v6.6: Error and Flow Control Protocol Specification, Data Transfer + Data Transfer Control," Pouzin Society, 2015.

[86] J. Day, "D-Base-2012-010: Relaying and Multiplexing Task Specification," Pouzin Society, 2012.

[87] J. Day, "D-Base-2011-015: Flow Allocator Specification," Pouzin Society, 2011.

[88] J. Day, "RINA-RFC-2010-002: Notes on the Resource Allocator," Pouzin Society, 2010.

[89] E. Grasa, S. Bunch and P. deWolf, "Specification of Managed Objects for the Demo DIF," Pouzin Society, 2012.

[90] J. Day, "Notes on the OIB/RIB Daemon," Pouzin Society, 2010.

[91] ISO, "Information technology – Open Systems Interconnection – Service definition for the Application Service Object Association Control Service Element". Patent ISO/IEC 15953:1999, 1999.

[92] ISO, "Information technology – Open Systems Interconnection – Connectionless protocol for the Association Control Service Element: Protocol specification". Patent ISO/IEC 10035-1:1995, 1995.

[93] ISO, "Information technology – Open Systems Interconnection – Common Management Information Protocol: Specification". Patent ISO/IEC 9596-1:1998, 1997.

[94] S. Bunch, "D-Base-2010-009: CDAP – Common Distributed Application Protocol," Pouzin Society, 2010.

[95] S. Bunch, J. Day and E. Trouva, "D-Base-2012-016: Common Application Connection Establishment Phase (CACEP)," Pouzin Society, 2012.

[96] V. Veselý, M. Marek, T. Hykel and K. Rausch, "Deliverable 2.4: RINA Simulator, basic functionality," January 2015. [Online]. Available: http://ict-pristine.eu/wp-content/uploads/2013/12/PRISTINE-D24-RINASim-draft.pdf. [Accessed July 2015].

[97] Brno University of Technology, "kvetak/RINA," GitHub, 2014. [Online]. Available: https://github.com/kvetak/RINA. [Accessed July 2015].

[98] PRISTINE consortium, "RINASimulator / RINA Sim Code," Open Source Projects, 2014. [Online]. Available: https://opensourceprojects.eu/p/pristine/rinasimulator/. [Accessed July 2015].

[99] OpenSim Ltd., "OMNeT++ - Manual version 4.6," 2015. [Online]. Available: https://omnetpp.org/doc/omnetpp/manual/usman.html#sec534. [Accessed July 2015].

[100] V. Veselý, M. Marek, O. Ryšavý and M. Švéda, "Multicast, TRILL and LISP Extensions for INET," *Journal On Advances in Networks and Services,* vol. 7, no. 3&4, pp. 240-251, 2014.

[101] V. Veselý and O. Ryšavý, "Locator/Id Split Protocol Improvement for High-Availability Environment," in *Proceedings of The Tenth International Conference on Networking and Services*, Roma, Italy, 2015.

[102] V. Veselý and O. Ryšavý, "Map-Cache Synchronization and Merged RLOC Probing Study for LISP," *International Journal On Advances in Intelligent Systems,* vol. 8, no. 3&4, 2015.

[103] V. Veselý, M. Marek, T. Hykel and O. Ryšavý, "Skip This Paper - RINASim: Your Recursive InterNetwork Architecture Simulator," in *Proceedings of the 2nd OMNeT++ Community Summit*, Zurich, Switzerland, 2015.

[104] V. Veselý, "Deliverable 2.6: RINA Simulator, advanced functionality," November 2015. [Online]. [Accessed November 2015].

[105] L. Jakab, A. Cabellos-Aparicio, F. Coras, J. Domingo-Pascual and D. Lewis, "RFC 7215: Locator/Identifier Separation Protocol (LISP) Network Element Deployment Considerations," April 2014. [Online]. Available: https://tools.ietf.org/html/rfc7215.

[106] D. Saucez, L. Iannone and O. Bonaventure, "LISP Threats Analysis," August 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-lisp-threats-13.

[107] Pouzin Society, "The Pouzin Society - Building A Better Network," 2012. [Online]. Available: http://www.pouzinsociety.org/. [Accessed November 2015].

[108] PRISTINE consortium, "Deliverable 3.2: Initial specification and proof of concept implementation of techniques to enhance performance and resource utilization in networks," April 2015. [Online]. Available: http://ict-pristine.eu/wp-content/uploads/2013/12/pristine-d32-enhance-performance-and-resource-utilization-in-networks-v1_0.pdf. [Accessed September 2015].

[109] F. Hrizi, A. Laouiti and H. Chaouchi, "SFR: Scalable Forwarding with RINA for Distributed Clouds," in *Proceedings of 6th International Conference On Network of the Future (NoF 2015)* , Montreal, Canada, 2015.