TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

DIPLOM THESIS PROJECT

CoreSim: A Simulator for Evaluating LISP Mapping Systems

Author:
Florin-Tudorel CORAŞ

Supervisors:
Phd. Virgil DOBROTĂ
Phd. Albert CABELLOS
Loránd JAKAB

Contents

List of Figures

Chapter 1

State-of-the-art

1.1 Problem Statement

In October 2006 the Internet Advisory Board (IAB) held a Routing and Addressing Workshop in Amsterdam, Netherlands with the goal of developing a shared understanding of the problems that the large backbone operators are facing regarding the scalability of the Internet routing system. The outcome of the meeting, their findings and suggestions, have been summed up in a Request For Comments (RFC 4984)[MZF07] and forms the input to the Internet Engineering Task Force (IETF) community which aims to identify the next steps towards effective solutions.

While many aspects of a routing and addressing system were discussed, the participants deemed two as most important and subsequently formulated two problem statements:

- Problem 1: The scalability of the Routing System
- Problem 2: The Overloading of the IP Address Semantics

An in depth analysis of the problems (including the two above) affecting the routing system was also provided.

1.1.1 The scalability of the Routing System

The main parameter when discussing about the scalability of a routing system is the size of the Routing Information Base (RIB), because it limits the number of destinations in the system, a router knows about. It only follows that a routing system will, at a given moment in time, be hindered in its growth by the RIB size and this impairment will be surpassed only by economical means, that is upgrading the system. This is known as product life cycle and plans to replace old devices might follow some fixed time patterns but an increase in the speed of RIB fill up would lead to obsolete products before their time.

The Default Free Zone (DFZ) is no different and the above statements hold. Throughout the years, since the Internet's birth, the DFZ RIB shape[bgp09] of the growth curve has been the topic of much research and debate. There have been various hypothesis regarding the sources of the growth but the workshop participants settled on what they viewed the main driving forces behind the rapid growth of the DFZ RIB:

- Multihoming
- Traffic engineering
- Suboptimal RIR address allocations
- Business events such as mergers and acquisitions

All the above factors lead to prefix de-aggregation and subsequently to the rapid growth of the DFZ RIB through the injection of unaggregatable prefixes. This also exposes the core of the network to the dynamic nature of the edges and thus to an increased number of BGP UPDATE messages injected into the DFZ ("UPDATE Churn"). To make matters worse, it is predicted that the update churn situation will be exacerbated by the Regional Internet Registries(RIR) policies through which end sites are allocated Provider Independent (PI) addresses. These addresses are not topologically aggregatable. This situation is explained by the need of multihoming without the "provider lock" that Provider Aggregatable (PA) space creates and also due to renumbering costs which some enterprises find unacceptable in the case when PA addresses are provide.

It is worth mentioning that the deployment of IPv6 is regarded to lead to the RIB and Forward Information Base (FIB) size increase by a factor of four, in other words the situation will be everything but improved.

1.1.2 The Overloading of the IP Address Semantics

One of the assumptions providing support for routing scalability was stated by Yakov Rekhter:

"Addressing can follow topology or topology can follow addressing. Choose one"

Following this idea some authors [O'D97, Chi99] have tried to provide the architecture for a scalable routing system by making use of aggressive topological aggregation. Unfortunately there is some difficulty in creating and maintaining the envisioned congruence. This difficulty arises from the overloading of addressing with semantics of both end users identifiers and router locators: there is the need to identify both clients and routers and only one number space is available. Either way, the overloading has been felt and moreover deemed to have had profound implications for the scalability of the global routing system. A solution for this problem would be to assign the locator part of the IP address in such a way that it will be congruent with the Internet topology. However, these assignments are based on organizational criteria and gave birth and now feed the continual growth of an incongruence between the topology and addressing. It was the workshop participants conclusion that such a desideratum is not fulfillable by means of only one number space in the context of topological routing systems and a split of the "locator/identifier overload" of the IP address semantics would be necessary to scale the routing system. However, details about the architecture of such a system were not explored.

1.2 Current State and Solution Space

Since the days of the IAB workshop nothing has changed for the better in the DFZ, though is has changed for the worse proving right some of their predictions. In that sense the BGP routing table has grown from 200000 entries in 2006 to 300000 entries in 2009 [bgp09].

1.2.1 Initial Research

Over the years considerable effort was put into finding or investigating solutions for the scalable inter-domain routing, some proved to be dead ends and other sparked new ideas, out of them the author wants to acknowledge the following:

MULTI6

This IEFT Working Group (WG) was formed to explore the solution space of IPv6 multihoming and present proposals compliant to what was then the definition of the IPv6 with the goal of avoid-

ing route injection to the DFZ [ABG03]. Their solutions revolved around two ideas: the allocation of PI address space for customers, this is effectively the current implementation, and the second assigning multiple address prefixes to multihomed sites. The second solution implies the use of both ISP address spaces and when one fails the communication is moved to the other address. This was also the largest class of proposed solutions [van]. From a technical point of view both solutions were flawed because the first category does not scale and the second introduces fundamental changes to the Internet routing system. It was the opinion of the workshop participants that the solutions solved problems by introducing new ones for which solutions were not proposed, thus they are incomplete [MZF07].

SHIM6

When MULTI6 was not chartered for solutions a new WG, SHIM6, was formed. They picked up the second idea from the MULTI6 and decided upon a host based solution. The host IP stack includes a "shim" that presents a stable "Upper Layer Identifier" (ULID) to the upper layer protocols but may rewrite the IP packets in accordance with the currently working IP address for transmitted packets. It provides locator agility below the transport protocols, so that multihoming can be provided for IPv6 with failover and load spreading properties, without assuming that a multihomed site will have a PI address prefix which is announced in the global IPv6 routing table[shi]. Both the locator and the identifier were 128-bit IPv6 addresses the former belonging to the packets and the latter to the transport layer.

The problems with this solution were not few and the first, obvious, one is that it required the change of all the host stacks. Morover there was no clear specification onto how end-to-end communications would take place in the context of multiple locators associated to a multihomed host [MZF07].

Important drawbacks were the lack of suport for Traffic Engineering (TE) at the service provider level and the need of performing renumbering when changing providers. It also implies additional state information on the host when considering that the remote communication end can be identified by multiple locators. This poses no problem for individual user hosts but it is a problem for servers which might have lots of ongoing connections[van].

Other open issues are related to ingres filtering done by ISPs through Access Lists(ACLs) and the question of acceptance of another header in the data packets.

GSE

It presented itself as an alternative addressing architecture for IPv6 which would control global routing growth by very aggressive topological aggregation and it did this by introducing another level of indirection. The proposal was to change the IPv6 address structure to bare the semantics of both an identifier and a locator. In this sense the first n bytes of the 16-byte IPv6 address are called the Routing Goop (RG), and are used by the routing system as locators. The last 8 bytes are the End System Designator (ESD), which specify an interface on the end-system. The rest of the bytes (16-n-8), which was belived to be around 2, form the Site Topology Partition (STP) which was described as Site-private "Lan Segment" information[O'D97]. The Site Border Routers need to rewrite the source RG of the site egress traffic to hide the internal site structure from the rest of the internet and also the destination RG of all the site ingress traffic, to hide the site's RG from all the internal routers.

The identifier/locator split requires that a mapping system is used to bind a set of locators to an identifier. For this purpose GSE proposed to use DNS, as a mapping service, but came short to providing solutions for locator failure recovery. Also, host stack modifications were required as the applications were allowed to use just the 8 bytes coresponding to the ESD[MZF07].

ENCAPS

This is a conceptual proposal and it is also built with the identifier/locator scheme in mind. The solution is different from the GSE one in the sense that it uses tunnels between the border routers instead of locator rewriting. In the envisioned system, the border router would read the destination and source addresses of a local site originated packet, do a mapping for these addresses (identifiers in fact) to locators by means of DNS and finally encapsulate the initial IP datagram with a new IP header containing the discovered locators and sends it to its proper destination. The border routers would compute routes between themselves by using existing routing protocols, may they be exterior gateway protocols like BGP or interior gateway protocols like OSPF or IS-IS[Hin96].

This solution avoids the conflicting needs of the ISPs and the customers for PA and PI prefixes by putting them in two different name spaces. Morover, this architecture provides a scalable multihoming solution with the advantage of no changes to the host stacks as the encapsulation is done by the border router. Also, it implies no changes to the practices of both applications and backbone operators[MZF07].

The proposal was by no means complete but, as it will be seen in the next section, it sparked ideeas for a new set of solutions in our time.

1.2.2 New Proposals

LISP

The Locator/Identifier Separation Protocol [FFML09b] came as a response of the Amsterdam IAB Routing and Addressing Workshop problem statement and aims, as previously mentioned solutions, to solve the scalability of the routing system. One of the conclusions of the workshop was that any solution to the routing scalability is necessarly a cost/benefit tradeoff thus, given the high potential for gains of the indirection approach (a locator/identifier split), the map-and-encap ideea was considered one of the most promising solution spaces.

The LISP draft focuses on a router based solution and proposes an incremental deployment of the technology. As with map-and-encap there will be no changes to the hosts stacks and to the majority of the routers within an AS but it will be necessary to deploy some new equipment namely Ingress Tunnel Routers (ITRs) and Egress Tunnel Routers(ETRs) and possibly to develop and deploy a new Mapping System. The role of a site ITR is to find the destination locator for a local site outgoing packet by means of the Mapping System, construct and prepend the LISP header to the original IP datagram and sent the resulting packet to the ETR. The LISP header is build from the destination locator and the ITR locator. The ETR's goal is to strip down the LISP header, if the received packet has as destination address the locator of the ETR, and forward the resulting packet to the destination EID, whithin its local site. This approach also gives the possibility to implement traffic engineering and multihoming in a scalable fashion.

Through its architecture LISP seems to solve the main issues with the current routing system and offers itself as a medium term solution, until new, innovative architectures will be invented and finaly deployed. But before we can talk about a LISP DFZ zone more research is needed for the Mapping System part of the LISP architecture because as in the case of past proposals the solution itself, though elegant, introduces new scalability problems. Considerable effort is being deployed into finding what would be the best suited Mapping System both in terms of lookup latency and scalability. Among the currently proposed system are: ALT, CONS, NERD, DHT and TREE.

It is worth mentioning that at the end of March this year, 2009, the LISP WG has been formed and was chartered to submit the specifications of LISP and the ALT mapping system by March 2010.

The current paper will focus on analyzing the performances of several LISP Mapping Systems in the following chapters.

HIP

Host Identity Protocol [MNJH08, MN06] is another solution which aims to split the locator and the endpoint identifier roles of the IP addresses, and in fact contains as a subset the "solution space" of the SHIM6 protocol. It also incorporate a new layer into the Internet TCP/IP model between the network and the transport layer. . HIP basically introduces a new 3.5-layer to avoid that sockets are bind to IP addresses forcing them to act both as the WHO and the WHERE identifiers. In HIP, upper layer sockets are bound dynamically to Host Identities (HI) instead of IP addresses.

The operation of HIP is as follows: hosts are identified with a HI that, in fact, is a public key of an asymmetric key-pair. Each host has at least one HI that can either be public or anonymous. Since public keys may have different sizes depending on the public key method HIs are represented via its 128 (SHA-1) hash, called Host Identity Tag (HIT) or via 32 bit Local Scope Identity (LSI). The HIT and LSI identify the public key that can be used for authentication purposes and are unique. HIT are mainly used for IPv6 while LSIs for IPv4. This way HIP is compatible with both versions of IP and does not require updating them.

During connection establishment HIP must be used. In this case the transport layer protocol (e.g. TCP) must be enclosed with a HIP header, which contains either the HIT or the LSI. The transport-layer end hosts are bind to this identifier, instead of the IP address. Since HITs are public keys it uses the Diffie-Hellman [Res99] key exchange to provide authentication. Additionally it can also provide confidentiality and message integrity. During the authenticated connection, mobility in HIP is quite straightforward. As HIs are used to identify the mobile node instead of IP addresses, the location of the node is not bound to the identifier. Therefore only a simple signalling protocol is needed to take care of the dynamic binding between the node's IP address and its HI. When the mobile node changes its point of attachment it simply sends a special signalling message (HIP REAddress) through the already authenticated channel. Again, since sockets are bound to HITs and not IP addresses, the connection can continue uninterrupted.

Finally a simple rendez-vous server is required to ensure reachability. This rendez-vous server is aware, through the use of some simple signalling, of the current location (IP address) of its serving HIP nodes. When another HIP-enabled node wants to establish a communication it retrieves its HIT identifier in some public address directory, such as the DNS. This directory stores both the HIT and the rendez-vous IP address. Then the node sends the initial HIP connection establishment method to the rendez-vous server, which in turn, forwards it to the actual location of the node. The remainder datagrams can be sent directly (route optimization).

The main advantages in HIP are that it does not change the architectural principles of the socket interface and that is transparent to applications. In addition since it is based in public key identifiers it relies on well-known and proven security mechanisms that provide authentication, confidentiality and message integrity. However this has also a downside, cryptographic algorithms, especially those based on asymmetric key pairs, are very costly in terms of CPU. Mobile devices have limited CPU power and HIP may impact its performance.

Ivip

This is an independent proposal from Robin Whittle and even though it is based on the Locator/ID separation protocol its author lists in his draft [Whi07] what Ivip takes from LISP, what it leaves out and what it adds. Ivip is not treated in this paper, for further datails reading the [Whi07, Whi09] references is recommended.

APT

For details related to this mapping system reading [J+07] is recommended.

1.3 Current Implementations

LISP+ALT Testbed

LISP+ALT is currently one of the most worked on Mapping Systems and Cisco have already implemented the protocol[FM09] in their IOS and a testbed of about 20 nodes is deployed[lis09]. The LIG (LISP Internet Groper) command has been implemented to allow users to test the network functionality.

OpenLISP

UniversitĂŠ catholique de Louvain is currently developing an implementation of the LISP[FFML09b] protocol in the FreeBSD 7.0 kernel and have reached alpha state[ISB08].

It is the author's opinion that these implementations are not providing means of evaluating the performances of a deployed LISP routing system but only practical proof that the implemented protocols work in size constrained systems. To better analyze the performances of LISP and the LISP Mapping Systems, with the above mentioned implementations, a large scale deployment is needed, but that comes to contradiction with the obvious purpose of the analysis, to test before deployment. Thus, another approach is needed, and among the possible ones, one can intuitively guess that of building a simulator.

To the best of the author's knowledge there was no implementation of such a simulator when the work for this paper started, nor is one when the work is almost complete. In the sections which follow, an in depth, step-by-step, presentation of the implementation difficulties and results implied by the pursuit of this approach is made.

Chapter 2

Theoretical Fundamentals

2.1 Locator/Identifier Separation principle

2.1.1 Endpoints and Endpoint Names

In his paper [Chi99], Noel Chiappa introduces the concept of Endpoint to solve what he belives to be an overloading of the name functionality in the context of networking. There are few fundamental objects in networking, also few names and among these, good examples are, “host names” and “addresses”. He has reached the conclusion that the reasons for this situation are twofold: first, this goes back as the erliest of papers in networking when the authors were not careful to distinguish between the concepts of an object and their associated names. This has caused widespread confusion between the properties of the object and those of the name. The second reason would be the lack of a rich set of fundamental objects. When dealing with new problems difficulties were encountered in finding/acknowledging the status of separate entities for previously existing, but “masked” objects.

In the days of the ARPANET the “address” had a straightforward meaning and was build by concatenating the number of the host with a port number. But as the scale of the internet has grown, the “tight” association between the functions of the term “address” and this sole “instantiation” of the name has stemed confusion. Chiappa and Saltzer [Sal93] explain this by the small number of well defined concepts at hand.

To set asside the confusion it was proposed that the term “address” should be redefined and used with one of the current implied uses and be limited to this particular one. Also, the fundamental object, the “endpoint” is defined. In fact, it is acknowledged as previously present in the network but unnamed.

For a better understanding of the technical aspects, the author, also definies bindings and namespaces. The binding being the association between a name and an object, but they may also map from one “kind” of name to another. Furthermore, instances of the same object may have more than one name and these names may be from the same class of names , or otherwise “namespace”. Depending on the namespaces we may have many-to-one bindings or alternatively many one-to-one bindings.

The relation between the structure of the names in a namespace and the function is also observed. This may be explained by the need for ease of use and a good example would be the structure implied by the IP addresses in order to ease routing. It is also obvious that names may have multiple ways of representation (eg. decimal printed notations and the bit representation for IPs)

The namespaces togheter with the possibility to represent names in multiple ways imply the existance of contexts which may help make the distinction between attachment of names to objects and mappings from one namespace to another.

All the above theoretical introduction can be used now to analyze the TCP/IP Architecture which did not make a clear distinction between the objects and the names of those objects. In fact it can be observed that the namespaces are as old as the NCP prococol architecture, namely the addresses and the host-names.

IP addresses are in fact the only "names" used throughout the TCP/IP architecture but they have multiple uses:

- Used by the routers when forwarding the user data packets
- Used to name a place in the network, the destination of a packet. Otherwise known as the "network attachment point" or "interface"
- Used in transport layer identifiers for both of the end-to-end communicating hosts.

The overloading of this single name and "field" with all these functionalities is not for the best and a soulution would be to split these three functions up, and have them performed by separete fields.

The other TCP/IP architecture namespace is that of host-names. They are human readable strings and contained in a hierarchical structure and can be looked up in the distributed and replicated database we all know as the DNS (Domain Name System). But their goal is, in fact, to map between the human readable characters and one or several IP addresses which will mediatate the TCP "conversation" between the hosts, once determined.

As expected, the double function of the IP addresses, that of identifying the interfaces and the hosts has downsides and an important one is the limitations of host mobility. This is because a TCP connection, as already mentioned, is always identified by the pair IP address and TCP port. It only follows that a change of the IP address , requested by the change of the position in the internetwork, will brake the TCP connection.

Chiappa tried to solve this problem by propsing a better definition for the term "address", in fact new definitions, or better bounded ones, for all three possible meanings. He suggests using "address" when referring to an interface, selector when talking about the field used by routers when forwarding packets and introduces a new fundamental object, the endpoint, to identify an end-to-end communicating host. He was unable, though, to give the endpoint namespace/namespaces because their uses are multiple, and as stated in the begining of this section, the forms of names within a namespace are highly dependend on how they will be used. But he did give a rich list of characteristics which he saw as useful.

In the current times, his ideas are somehow "woken up from slumber" to help solve the routing scalability problem by spliting the different functionalities of the term "address": that of locator in the core routing system (selector) and endpoint identifier (the host interface).

2.1.2 Initial Proposals

Address rewriting

The ideea was originally proposed by Dave Clark and later by Mike O'Dell in his 8+8/GSE [O'D97] specification. The aim was to take advantage of the 16-byte IPv6 address and use the lower 8 bytes as End System Designator(ESD), the top 6 bytes as a routing locator ("Routing Goop" or RG) and the ones left in between, 2 bytes, as Site Topology Partition (STP).

The model draws a strong distinction between the transit structure of the Internet and a Site that may contain a rich but private topology which may not "leak" into the global routing domain. Also the Site is defined as the fundamental unit of attachment to the global routing system, being in fact a leaf even if it is multihomed. But the above mentioned structure of the address brings also the desired distinction between the identity of end system and its point of attachment to the

"Public Topology". O'Dell also observed the overloading of the IP semantics and the consquences it has on address assignment when topology changes are done.

The most important part of the proposal, and which in fact insulates Sites from the global routing system, is the rewriting of the RG by the Site Border Routers. In this sense, when generating a packet, the source hosts fills in the destination address with the complete 16-byte IPv6 destination address, RG included, that it receives through DNS resolution, and fills the the source address's RG with a "site local prefix". Now, when a packet destined for a remote host arrives at the local site egress router, it has its source RG filled in to form a 16-byte address. On the other hand, when a packet reaches the destination sites ingress router the RG is stripped off and replaced with a "site local prefix" to keep the local hosts and routers from knowing the domain's RG. The obvious result of this decision is that upper-layer protocols must use only the ESD for end point identification, pseudo-header checksums and the like.

The above mentioned insulation provides a site with flexibility of re-homing and multihoming. And this is because a site's interior hosts and routers are unaware of the RG and thus if a change in the RG, due to administrative decisions, does occur the "interior components" need not know it. Morover, this brings forth the possibility of topological aggregation, with the goal of routing scalability, by partitioning the Internet into what O'Dell named as "set of tree-shaped regions anchored by 'Large Structures'". The Routing Goop, in an address, would have the purpose of specifying the path from the root of the tree, or otherwise the "Large Structure", to any point in the topology. In the terminal case that point would be a Site. Thess "Large structures", thus have the goal of aggregating the topology by reational subdivision of the space under them and delegation. It also follows that in the case when no information about next hop is known, the large structure could be used as forwarding agents. This will significantly limit the minimally-sufficient information required for a router when doing forwarding. It was also envisaged that additional route information kept is the result of path optimizations from cut-throughs. This has been proven as a wrong observation and will be detailed when treating the limitations of this system.

For further details related to the structures within the IPv6 address and also possible solutions to re-homing and multihoming, reading the draft RCF [O'D97] is highly recommended.

Given the age (this solution has been suggested in 1997) and also the lack of solutions, at the time, for some of the component proposals, it is only natural that today we see limitations with this design and in what follows some of them will be detailed. Good overviews of this system and its limitations are made by [Zha06, Mey07].

The main "hole" in the GSE design, seems to be the use of DNS when learning about destination hosts. Even if one assumes that root servers will stay relatively stable it must also accept that the ones "under" will not. And if considering that a site is multihomed which and how many of its RG should be returned as reply to a DNS server lookup for that site? Furthermore , given its role, a DNS server must all the time know the RG of the site it currently resides in such that a proper answer can be given for DNS queries. This comes to contradiction with the above stated "insulation principle". In addition, the support for 2-faced DNS server is brought up, that is, the server must know if the query is remote or local site in nature in order to know if the RG should be included or not in the reply message.

Another issue, is handling border link failures. It is possible for the source site to be aware of the status of its border links and choose the one of those which are up but at this point in the path followed by a packet from source to destination, it is imposible to determine if one of the border routers at the destination has lost connectivity to the site. Thus as a solution GSE propsed that the border routers for a site be manualy configured to form a grup and when one loses connectivity to the client site, it should forward the packet to others still connected. It should be noted that this is not an issue specific with GSE but with all solutions which propose a split between the edges and transit routing domains.

It was anticipated above that the "Large Structure" anchorage of the tree shaped regions, with

little or none interconnection between the lower regions was a wrong "hunch". And indeed it is, as the trend in the last, ten or so, years has shown that the interconnections below the top level are the norm rather than controlled circumventions thus the proposed RG structure my need revisiting.

Also, though it has scalable support for multihoming, GSE lacks support for traffic engineering. It may be possible for it to solve this goal but the existing proposal does not solve this problem. Same holds true for IP tunneling across RG boundaries and given the extensive use of Virtual Private Networks (VPN) a thourough examination of tunneling operations is needed in the GSE context.

Map-and-Encap

The idea, as originally proposed by Robert Hinden in his ENCAP scheme [Hin96], speaks about splitting the current single address space in two separate ones: the EID space , which covers the hosts and the one used for transit between the domains(RLOC space). It is belived that, through the decoupling of the EID non topological aggregatable space from the RLOC (provider owned) space, aggregation can be achieved in the core routing domain and eventually routing scalability.

When a source wants to send a packet to a destination outside its domain, the packet traverses the domain infrastructure to a border router. The packet has source address the EID of the initiatior and as destination address the EID of the destination host, which could be obtained by means of DNS. The border router needs to map the destination EID to a RLOC, otherwise an entry point in the destination network, and the proper means to this would be a mapping system. Obviously this phase is called the "map" phase of map-and-encap. After the mapping is done, the border router encapsulates the destination packet, by prepending a new "outer" header, with the destination address the RLOC of the destination network and the source address its own RLOC, and injects the packet in the routing domain. This phase is the "encap" phase of the map-and-encap model.

Thus, the system in its simplest of explanations consists in the insertion of a new header obtained by means of a mapping process. When the encapsulated packet arrives at the destination border router, the router decapsulates the packet and sends what was the original, source host built, packet to the destination EID in the domain. It is observed that both in source and destination domains the EIDs need to be routable (but their scope is local).

Besides providing an architecture with the goal of obtaining routing scalability other advantages of map-and-encap are the lack of host stack and core routing infrastructure changes. Also this scheme works with both IPv4 and IPv6 addresses and retain the original source address, a feature useful in various filtering scenarious [Mey08]

As downsides, this model, as the address rewrinting one, has problems in handling border link failures and the overhead implied by the encapsulation is stemming controversy.

Both of the above presented models, which have inspired new solutions in our current context, seem to give, arguably with a general approach, ways to solving the routing scalability problem. But in doing so, given their reliance on the addition of a new level of indirection to the addressing architecture, practically their intrinsec need to translate EIDs into RLOCs, have created a new problem and the solution to it are the mapping systems. The scalability problem has now shifted from the routing system to the mapping system and the success or failure of future solution will heavily depend uppon the careful design of the mapping system architecture.

2.2 Locator/Identifier Separation Protocol (LISP)

Before diving into the protocol details some basic LISP terms must be defined. They are definitions and therefore taken from their source, the Internet Draft (ID) [FFML09b] Morover, for a better understanding an example of a LISP compliant topology is provided in figure 2.1.

When a host (in a LISP capable domain) wants to send a packet, it first finds by DNS means the destination EID. It then sets as source address its EID and as destination address the EID it found in the previous step. If the destination EID is in another domain the packet traverses the source domain network infrastructure and reaches an Ingress Tunnel Router (ITR) [Mey08].

If the ITR has cached the EID-to-RLOC mapping for the destination EID, it encapsulates the received packet in a LISP header with its RLOC as source address and the mapping provided RLOC as destination address. It then sends the packet over the internet to the destination ETR which decapsulates the packet and sends it to the destination computer.

On the other hand, if the ITR does not have any knowledge about the destination address, it creates a Map-Request message, injects it in the mapping system and awaits reply. The mapping system "routes", by implementation specific means, the packet to an authoritative ETR for the searched EID, capable of providing the required EID-to-RLOC mapping. When the ETR receives the request it decapsulates the packet and sends through the Internet (not the mapping system) a Map-Reply to the source ITR with the required information. Now the ITR can proceed to sending the packet to one of the mapping specified ETR as in the above mentioned case.

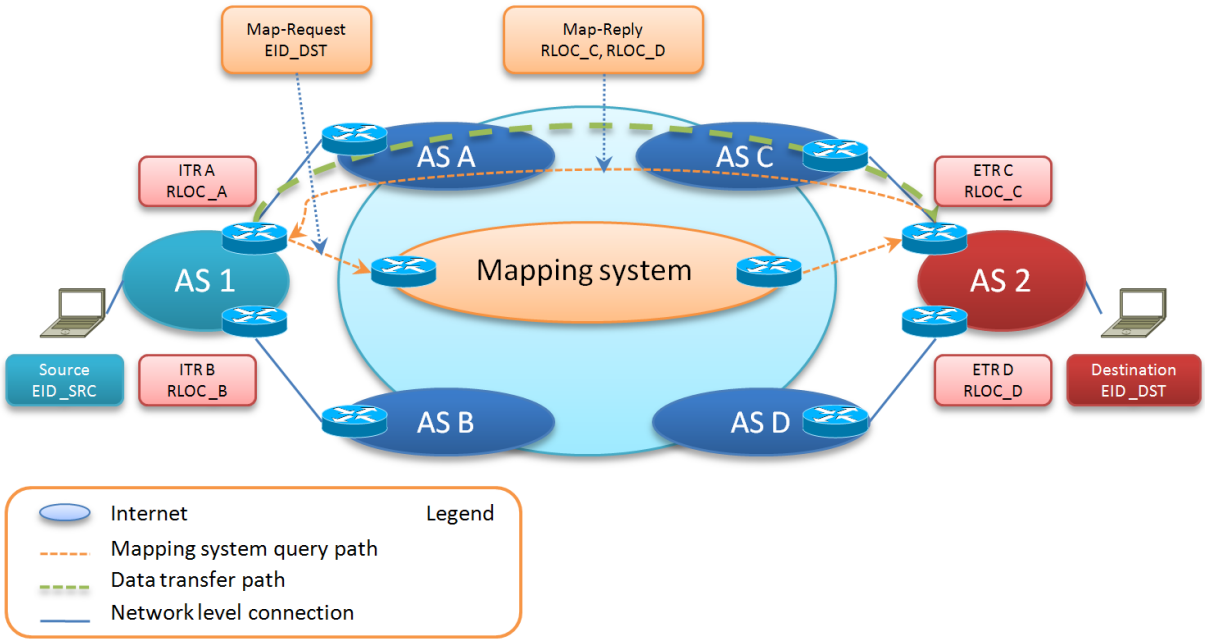

Figure 2.1: LISP Topology Example

The figure presents a generalised model for the steps needed to be taken when intra domain communication is done. Suppose the source to be the computer connected to AS1, the destination the computer connected to AS2 and the source knows about the EID of the destination computer by means of DNS. If the communication is desired, then the source proceeds to creating a packet with its EID (EID_SRC), as source adress and EID_DST as the destination address. Because the destination is in another domain, the packet needs to traverse the AS1 network infrastrucure and reach one of the ITRs, the figure depicts ITR A as the receiver of the packet. ITR A does not have a mapping for EID_DST and thus LISP-encapsulates the packet, setting as outer header destination address the searched EID, as source address its RLOC and injects the packet in the mapping system. When ETR C receives the packet it decapsulates the request and sends a reply with the two to be used RLOCs (RLOC_C, RLOC_D), through the Internet, to ITR A. Now ITR A can send LISP-encapsulated packets, destined for EID_DST, to any of the two RLOCs (priorities may be implied also, not discussed in this example) and they will decapsulate and send them to the destination host.

Ingress Tunnel Router (ITR) : a router which accepts an IP packet with a single IP header (more precisely, an IP packet that does not contain a LISP header). The router treats this ”inner” IP destination address as an EID and performs an EID-to-RLOC mapping lookup. The router then prepends an ”outer” IP header with one of its globally-routable RLOCs in the source address field and the result of the mapping lookup in the destination address field. Note that this destination RLOC may be an intermediate, proxy device that has better knowledge of the EID-to-RLOC mapping closer to the destination EID. In general, an ITR receives IP packets from site end-systems on one side and sends LISP-encapsulated IP packets toward the Internet on the other side. Specifically, when a service provider prepends a LISP header for Traffic Engineering purposes, the router that does this is also regarded as an ITR. The outer RLOC the ISP ITR uses can be based on the outer destination address (the originating ITR's supplied RLOC) or the inner destination address (the originating hosts supplied EID).

TE-ITR : is an ITR that is deployed in a service provider network that prepends an additional LISP header for Traffic Engineering purposes.

Egress Tunnel Router (ETR) : a router that accepts an IP packet where the destination address in the ”outer” IP header is one of its own RLOCs. The router strips the ”outer” header and forwards the packet based on the next IP header found. In general, an ETR receives LISP-encapsulated IP packets from the Internet on one side and sends decapsulated IP packets to site end-systems on the other side. ETR functionality does not have to be limited to a router device. A server host can be the endpoint of a LISP tunnel as well.

TE-ETR : is an ETR that is deployed in a service provider network that strips an outer LISP header for Traffic Engineering purposes.

xTR : is a reference to an ITR or ETR when direction of data flow is not part of the context description. xTR refers to the router that is the tunnel endpoint. Used synonymously with the term ”Tunnel Router”. For example, ”An xTR can be located at the Customer Edge (CE) router”, meaning both ITR and ETR functionality is at the CE router.

EID-to-RLOC Cache : a short-lived, on-demand table in an ITR that stores, tracks, and is responsible for timing-out and otherwise validating EID-to-RLOC mappings. This cache is distinct from the full ”database” of EID-to-RLOC mappings, it is dynamic, local to the ITR(s), and relatively small while the database is distributed, relatively static, and much more global in scope.

EID-to-RLOC Database : a global distributed database that contains all known EID-prefix to RLOC mappings. Each potential ETR typically contains a small piece of the database: the EID-to-RLOC mappings for the EID prefixes ”behind” the router. These map to one of the router's own, globally-visible, IP addresses.

Data Probe : a LISP-encapsulated data packet where the inner header destination address equals the outer header destination address used to trigger a Map-Reply by a decapsulating ETR. In addition, the original packet is decapsulated and delivered to the destination host. A Data Probe is used in some of the mapping database designs to ”probe” or request a Map-Reply from an ETR; in other cases, Map-Requests are used. See each mapping database design for details.

For a detailed description of LISP, the variants, its control and data plane operations checking the LISP draft [FFML09b] is recommended

2.2.1 LISP Map Server

This information is based on the LISP Map-Server draft [FF09].

The main concepts used in this subsection are:

Map-Server : a network infrastructure component which learns EID-to-RLOC mapping entries from an authoratative source (typically, an ETR, though static configuration or another out-of-band mechanism may be used). A Map-Server publishes these mappings in the distributed mapping database.

Map-Resolver : a network infrastructure component which accepts LISP Encapsulated Map-Requests, typically from an ITR, quickly determines whether or not the destination IP address is part of the EID namespace; if it is not, a Negative Map-Reply is immediately returned. Otherwise, the Map-Resolver finds the appropriate EID-to-RLOC mapping by consulting the distributed

Encapsulated Map-Request : a LISP Map-Request with an additional LISP header prepended. Sent to UDP destination port 4341. The ”outer” addresses are globally-routeable IP addresses, also known as RLOCs. Used by an ITR when sending to a Map-Resolver and by a Map-Server when sending to an ETR.

Negative Map-Reply : a LISP Map-Reply that contains an empty locator-set. Returned in response to a Map-Request of the destination EID does not exist in the mapping database. Typically, this means that the ”EID” being requested is an IP address connected to a non-LISP site.

Map-Register message : a LISP message sent by an ETR to a Map-Server to register its associated EID prefixes. In addition to the set of EID prefixes to register, the message includes one or more RLOCs to be be used by the Map-Server when forwarding Map-Requests (reformatted as Encapsulated Map-Requests) received through the database mapping system.

Overview

The Map-Server is a device which publishes EID prefix information on behalf of the ETRs and connects to the LISP distributed mapping database system to help answer LISP Map-Requests seeking the RLOCs for those EID prefixes. To publish its EID-prefixes, an ETR sends Map-Register messages to the Map-Server. A Map-Register message contains a list of EID-prefixes plus a set of RLOCs that can be used to reach the ETR when a Map-Server needs to forward a Map-Request to it.

A Map-Server connects to LISP+ALT network and acts as a ”last-hop” ALT router. Intermediate ALT routers forward Map-Requests to the Map-Server that advertises a particular EID-prefix and the Map-Server forwards them to the owning ETR, which responds with Map-Reply messages.

The LISP Map-Server design also includes the operation of a Map-Resolver, which receives Encapsulated Map-Requests from its client ITRs and uses the distributed mapping database system to find the appropriate ETR to answer those requests. In an ALT network, a Map-Resolver acts as a ”first-hop” ALT router. It has GRE tunnels configured to other ALT routers and uses BGP to learn paths to ETRs for different prefixes in the LISP+ALT database. The Map-Resolver uses this path information to forward Map-Requests over the ALT to the correct ETRs. A Map-Resolver may operate in either a non-caching mode, where it simply de-capsulates and forwards the Encapsulated Map-Requests that it receives from ITRs, or in caching mode, where it saves information about those Map-Reqeusts, originates new Map-Requests to the correct ETR, accepts and caches the Map-Replies, and finally forwards the Map-Replies to the original ITRs.

Note that a single device can implement the functions of both a Map-Server and a Map-Resolver. As is the case with the DNS, however, operational simplicity argues for keeping those functions separate.

ITR Mapping Resolution

The ITR knows by default configuration the address of Map-Resolver. Obviously this needs to be the locator (RLOC) which is routable on the underlying core network, and not be an EID which should be resolved through another mapping because it would impose a circular dependancy. When using the Map-Resolver the ITR needs no other knowledge about the mapping system and for the LISP+ALT mapping system it doesn't even need to implement BGP or GRE tunnels.

When it needs a mapping for an EID-to-RLOC mapping it does not have stored in its local cache, it sends an Encapsulated Map-Request to its configured Map-Resolver.

As reply to a request, the ITR can expect two possible messages:

- A negative LISP Map-Reply. When the Map-Resolver determines that the requested EID does not exist in the mapping system. Also, the returned EID prefix should be saved by the ITR to avoid future queries about the non LISP capable site.
- A LISP Map-Reply. This comes from the authoritative ETR which owns the EID-to-RLOC mapping or from a Map-Server answering in the name of the ETR.

Prefix Registration

The ETRs register their prefixes to a Map-Server by means of a LISP Map-Register message. Security is also provided, but details are not treated in this paper. As with the ITR, in the context of LISP+ALT, if the ETR decides to use a Map-Server it does need to participate further in the mapping system, thus no need to implement BGP or GRE tunnels. To be noted though that the Map-Server does not preclude the ETRs from participating in the mapping system, but the goal of the Map-Server is to ease the complexity at the customer site.

Processing

When a Map-Server receives a Map-Request for an EID inside one of its registered EID prefixes it re-encapsulates the Map-Request and forwards it to one of the matching ETRs. No further altering of the Map-Request is done and the ETR Map-Reply is returned to the RLOC from the Map-Request.

When a Map-Resolver receives a mapping request it proceeds to de-capsulating the request and checking it local database of mapping entries. If a matching entry is found it will return a non authoritative LISP Map-Reply. Also if it can determine that the required EID belongs to a non LISP capable site it will return a negative LISP Map-Reply. The third case, when the Map-Resolver does not have enough information about the EID prefix, is solved by forwarding the request to another device which has more information about the requested EID.

The draft specifies two possible scenarios:

non-caching : In this case the Map-Resolver just forwards the unencapsulated Map-Request with the LISP header unaltered. The ETR/Map-Resolver which receives the request respond directly to the ITR.

caching : The Map-Resolver now stores Map-Request related information in its local queue, including the nonce. It then alters the LISP Header and adds its RLOC as source address, generates a local nonce and forwards the map request in the mapping system. Now, the

reply will come to the Map-Resolver, which by means of the above mentioned local nonce dequeues the Map-Request stored data, determines the requesting ITR, build a correct Map-Reply and sends it to the ITR. Also, the Map-Reply information is stored in its local cache.

2.3 LISP Proposed Mapping Systems

2.3.1 LISP+ALT

All the details related to this topology come from the draft [FFML09a]. This is an alternative logical topology for managing Endpoint identifier to Routing locator mappings using the the Locator/Identifier Separation Protocol. This logical topology uses existing technology and tools, specifically the Border Gateway Protocol [RLH06] and its multi-protocol extension, along with the Generic Routing Encapsulation[FLH$^+$00] protocol to construct an overlay network of devices that advertise EID-prefixes only. These devices, the Endpoint Identifier Prefix Aggregators, hold hierarchically-assigned pieces of the Endpoint Identifier space (prefixes) and also know the next hops toward the network element which is authoritative for Endpoint Identifier-to-Routing Locator mapping for that prefix. Tunnel routers can use this overlay to make queries against and respond to mapping requests made against the distributed Endpoint Identifier-to-Routing Locator mapping database. It is worth noting that the database is distributed and is stored in the ETRs [FFML09a].

One of the goals was to minimize the number of changes to hadware and/or software that would be required in order to deploy the mapping system. Its authors even belive that in most cases existing technology may be used to deploy LISP+ALT and this because of the introduction of new devices besides the existing ones which form a robust physical topology.

The LISP 1.5 Model

This model makes use of the same query/response protocol intricates as LIPS 1.0. In particular, the LISP+ALT implemetation provides two mechanisms for an ITR to obtain EID-to-RLOC mappings:

- Data Probe: An ITR may send the first few data packets into the ALT to minimize packet loss and to probe for the mapping; the authoritative ETR will respond to the ITR with a Map-Reply message when it receives the data packet over the ALT. It should be noted that in this case, the inner Destination Address (DA), which is an EID, is copied to the outer DA and is routed over the ALT. The use of Data Probes is highly experimental and should be disabled by default.

- Map-Request: An ITR may also send a Map-Request message into the ALT to request the mapping. As in the Data Probe case, the authoritative ETR will respond to the ITR with a Map-Reply message. In this case, the DA of the Map-Request must be an EID.

Both for LISP 1.0 and 1.5 the EIDs are routable but if in the former they are routable in the routing domain for the latter are routable, except for the local site, only over a separate, virtual topology referred to as The Alternative Virtual Network. This network is built as an overlay on the public Internet and uses tunnels to interconnect LISP+ALT routers. BGP is run over these tunnels to propagate the information needed to route Data Probes and Map-Request/Replies.

In this context, the ETRs are sources of unaggregated EID prefix data which the ALT topology, by BGP means, tries to aggressively aggregate. It is not mandatory for ETRs to participate in ALT nor is it prohibited. As static solution, they may choose to communicate their mappings to the associated LISP+ALT routers at subscription time via configuration. ALT is also not conditioned by the participation of ITRs.

It is worth mentioning that the LISP draft [FFML09b] and the LISP+ALT draft [FFML09a] don't agree on the type of LISP version this mapping system implements. The former presents it as being an implementation of version 3 but the latter, which also defines all the details related to LISP+ALT presents it as an implementation of version 1.5.

LISP+ALT Overview

LISP+ALT is a hybrid push/pull architecture in which aggregated EID prefixes are "pushed" among the LISP+ALT routers (there exists the option for ITRs to receive aggregated prefix information if they participate in ALT) and specific EID-to-RLOC mappings are "pulled" by ITRs by one of the two possible, above mentioned, means.

The fundamental idea embodied in LISP+ALT is the use of BGP, running on top of an tunneled overlay network, to establish the reachability required to route Data Probes and Map-Requests in the alternate logical topology. In this system, LISP+ALT routers receive unaggregated prefixes by means of eBGP links to authoritative ETRs or by static configuration. These prefixes are aggregated by the ALT topology. The knowledge of the prefix "ownership" is spread throughout the topology through the eBGP links which interconnect the ALT routers. Also ITRs may eBGP peer with ALT routers to learn the best ALT router to forward a Map-Request for a particular prefix. It is expected that in most of the cases the ITR will have a default EID mapping poiting to one or more LISP+ALT routers.

It is to be noted that even though the draft speaks about using GRE tunnels to build the virtual topology there is no constraint or binding to their usage and thus their mentioning should not be taken as prohibiting or precluding the use of other, available tunneling mechanisms.

ITR Traffic Handling

When an ITR receives a packet from one of the hosts for which it acts as a border router, it tries to match the destination EID to a possible entry in his cache. If this fails, the ITR encapsulates the packet in a LISP header, copying the inner destination address (EID) to the outer destination address (RLOC), and trasmits it, through a GRE tunnel, to a LISP+ALT router. It may be possible for the ITR to choose among more ALT routers, if it does have connection to more than one, by means of the routing information received through BGP from them. The "first hop" ALT router uses EID-prefix routing information learned from other LISP+ALT routers via BGP to guide the packet to the ETR which "owns" the searched EID. When the ETR receives the packet, it replies to the ITR with a Map-Response , which contains the RLOCs to be used to contact the destination EID, and also strips down the LISP header and forwards the packet to the destination.

Upon receipt of the Map-Reply, the ITR installs the EID-to-RLOC mapping in its local cache. This process will avoid further lookups of the same EID prefix in the mapping system and thus, enables the ITR to forward possible new packets to the responsible RLOC directly. It is possible, though, for a cache entry to indicate RLOCs which due to various reasons might be "down". In this case the packets destined for the EID will be dropped (not routed through ALT).

EID Assignment and Aggregation

The assignment is belived to be done by LISP Internet Registries and in the case of multiple allocations that may or may not be in power of 2 blocks. In the favorable case, when they are, they will be aggregated into a single, advertised EID-prefix. The ALT architecture should resemble a tree structure hierarchy with the goal of allowing aggregation at merge points in the tree. It is belived that such a structure will aid in minimizing the number of advertised EID prefixes near the top of the tree.

The ALT topology should be relatively static as there will be no changes due to subscription or policy reasons, thus aggregation would be sustainable. Another point is that ALT routes based on BGP and for this reason it needs to aggregate also based on BGP rules, therefore an ALT router should be configured with BGP sessions to all the components (or more specific prefixes) of an aggregate if it is to announce the aggregate. As an exception, there exists the possibility for a router to be statically configured to learn the state of all the components, specifically those which may be holes or down in the covering prefix, to enable the aggregate creation.

The creation of the aggregate is, though, the result of local policy and in some cases, an ALT router, will be configured to do so at all times with a statically configured discard route. In other cases, it may be configured to do so only if one of the components of the aggregate is learnt through BGP.

This requires that two ALT routers are to exchange their sets of prefixes if their aggregate is to be the same. Furthermore it implies that an ETR needs to be a BGP peer to all the ALT routers that are configured to aggregate on of its prefixes.

It is to be noted though, that the internet draft speaks of no “best” way to build the ALT network and considers that a solution will arrise as testing and prototype deployment proceeds.

LISP+ALT Router Requirements

The draft specifies the following functionalities/requirements for an ALT router:

- It runs, at a minimum, the eBGP part of the BGP protocol
- It supports a separate RIB which uses next-hop GRE tunnel interfaces for forwarding Data Probes and Map-Requests.
- It can act as a ”proxy-ITR” to support non-LISP sites.
- It can act as an ETR, or as a recursive or re-encapsulating ITR to reduce mapping tables in site-based LISP routers.

Traffic Engineering with LISP+ALT

LISP+ALT does not hold the EID-to-RLOC mappings but it offers a mechanism through which the ITRs can query the ETRs responsible for the EID prefixes of interest. The benefits of this approach are the following: first, the “flapping” of state information through BGP is not to be considered likely because the RLOCs are learned through ITR-ETR exchange, also mapping information won’t become stale due to slow propagation through the alternative topology. The second benefit is that this deffering of EID-to-RLOC mapping to an exchange between the ITR and ETR provides the possibility of supporting inter site traffic engineering by means of weights and preferences, set in the LISP header, and possibly by returning more specific EID-to-RLOC mappings.

This mechanism avoids current practices of routing prefix deaggregation, through which, more specific prefixes are injected into the DFZ for traffic engineering purposes. These deaggregations are done with the goal of improving local or regional traffic flows. This information can now be changed through LISP, in the ITR-ETR exchange of packets and it will be up to the ITR to decide if such, more specific, information will be stored or not. It can be observed that this technique has the desirable properties of reducing the number of deaggreagated prefixes seen in the DFZ and allowing richer and more fine-grained TE policies.

2.3.2 LISP-DHT

Chord

Chord[SMLN+03] is a Distributed Hash Table which simplifies the design of peer-to-peer systems and applications based on them by addressing problems like load balancing, decentralization, scalability, availability and flexible naming.

In this system each node has an unique m-bit identifier (denoted ChordID), and all the nodes are organized in a ring. Each node maintains a routing table with at most m entries and this is called the finger table. The aim of this finger table is to improve the search time for a key in the system. The position, of an entry, in the table determines the distance on the Chord ring the entry points to. Thus, the i^{th} entry in the table will contain the identity of the first node at least 2^{i-1} positions further away in the identifier space.

Being a distributed hash table, Chord needs a way to map between the keys and nodes, and for this it uses a consistent hashing function, namely SHA-1. Morover, due to the fact that the nodes and the keys use the same namespace, the following algorithm is used to associated keys to a node: all the keys which have their identifier smaller than that of a given node A, but higher than that of the previous node B will belong to node A.

As previously mentioned, the Chord nodes maintain finger tables for performance reasons and their absence does not imply incorrect operation of the protocol but just a larger latency when searching for a key. The reliability and consistancy can be increased, if a given node, maintains pointers to several successors and predecessors on the ring. Also. load balancing is assured by the random assignment of keys to nodes, assignment produced by the hash function.

The ring join operation requires only the knowledge of an already participant node which will provide initialization values for the finger table, successor node and predecessor node. The process olso requires the joing node to update the successor pointer of its successor predecessor, the predecessor pointer of its succesor and also the finger tables of the nodes which should use the newly joined node. After this the key transfer process, through which the joining node becomes responsible for the keys between his predecessor (excluding it) and himself, can occur.

Additional information related to Chord and the protocol's performances can be found in [SMLN+03]

Modified Chord

By default Chord nodes are supposed to choose their ChordIDs in a random fashion. This holds true for the keys also, which should be "spread" by the SHA-1 hash function evenly across the Chord ring, assuring in this way load balancing. But, in the LISP context, if we consider the nodes the ETRs and the keys the announced EID prefixes, such a mapping system would preclude the authoritative entities from managing their own prefixes. Obviously such a situation is unacceptable.

To circumvent this limitation, the LISP-DHT draft[SMLN+03] proposes that nodes should use as ChordIDs an EID, and more specifically, the highest EID in the announced EID prefix. Also, domains announcing more than one prefix are required to enter the ring with more than one instance, each having the ChordID associated to their respective prefix. This assures that a node will be the one managing the prefixes it announces (it may be the case that a Chord node "manages" unallocated prefixes).

In this context each key will have as associated value the RLOCs of the border routers associated to the EID prefix originating domain. The knowledge of the predecessor node, and more importantly, of its ChordID is required for key management and consistent routing. Meaning that a given node can check to see if the predecessor unlawfully claimed part of his EID-prefix.

ITRs

Even though, the design specified until now is able to accomodate Mapping Servers and route queries from one point in the ring to the appropriate destinations, it is not able to incorporate the functionality of the ITRs. These machines won't route queries and almost certanly won't be resposible for prefixes (that would be the job of ETRs), therefore it is impossible to admit ITRs to the Chord ring (with the current definition of the Chord node) because they would "steal" part of the prefixes and wouln't be able to route queries. Thus, as anticipated above, a new type of node needs to be defined, and the name proposed in the draft is "stealth node". The stealth DHT node brings the count of node types to two:

- the service nodes, which behave like fully-fledge DHT nodes
- the stealth nodes, which can inject messages onto the DHT, but are never used for routing or key management.

This separation is possible because two distinct phases need to be accomplished during the joining procedure: the gathering phase, which ends with the node having enough information to take part in Chord, and the announcement phase during which the node advertises its presence and aquires part of the keyspace.

The stealth nodes would complete only the first phase, while the service nodes both. These stealth nodes, the ITRs, have all the necessary information for injecting packets in DHT, they have successors, predecessors, finger tables, but are not successor or predecessor to any node. Morover stealth nodes cannot store keys and are not queried by Map-Requests.

The draft also speaks about methods of implying security by means of certificates and ways of increasing the robustness of the system by means of redundancy but this is not the subject of the current paper. For further details checking [MIB08] is recommended.

Chapter 3

Design

3.1 Simulator architecture

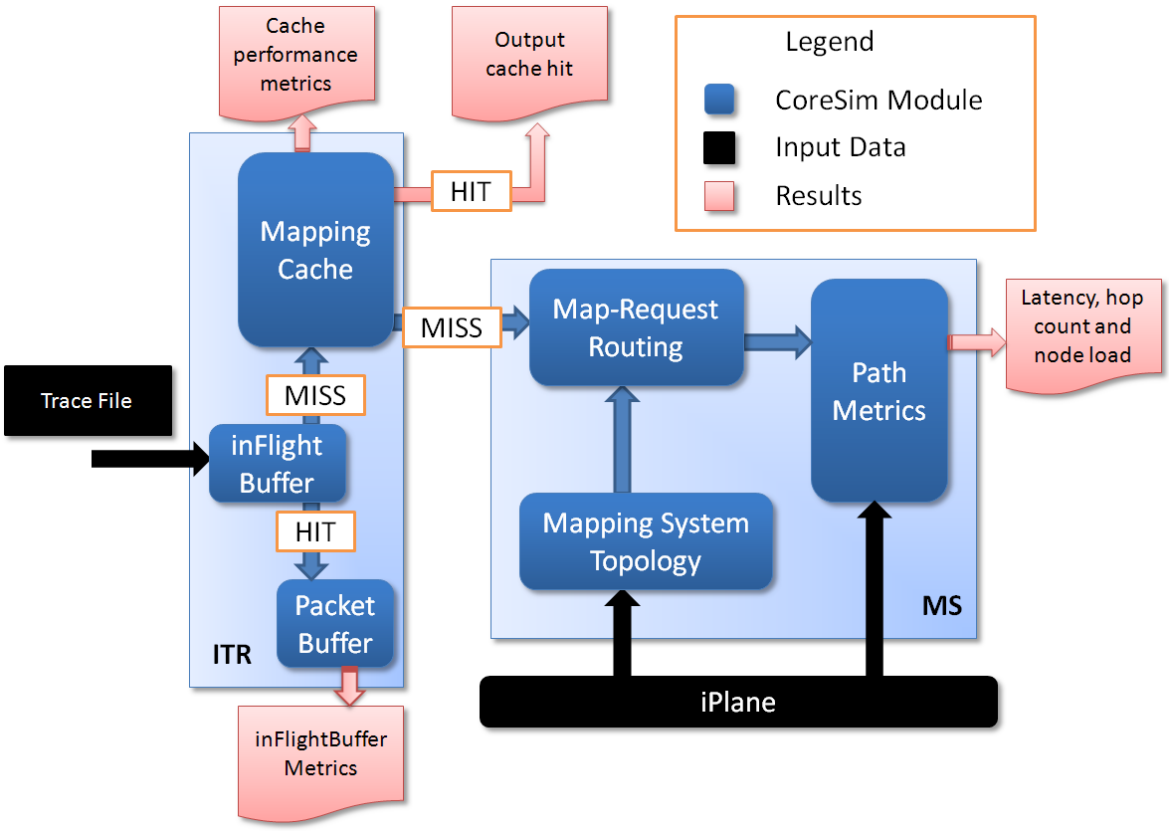

Figure 3.1: CoreSim Architecture

The above figure depicts the high level architecture of the simulator and the way its main components interact in order to provide the desired functionality. Much more insight is provided in the following sections where all of the above blocks are explained through the examination of their implementing classes.

3.1.1 Simulator Components

This section offers a basic overview of what CoreSim is able to perform and gives some "surface" details related to how the architecture was conceived and subsequently implemented. The key

word in the description, which defines and also implies further functionality, is modularity. This was a request and also a necessity.

The modular nature of the project makes it cluster functionality in several classes built on top of other "smaller", more specialised ones. As presented in the above figure 3.1, at the highest of levels, the simulator can be seen as having the following components:

- Topology
- Chord
- ALT
- ITR
- DBI
- Utilities

The above order crudely follows the timeline implementation path, and thus has some exceptions to this rule . But it does speak, in a rough way, about the components importance. Do not imagine them as having a ladder hierarchy but a layered one, with the functionality deciding the occupied layer.

From this perspective, the whole simulator is built on top of the **Topology** module which contains what is the view of the iPlane project (iPlane will be presented in section 3.3.1) related to the Internet architecture. And, in a few words, this means it is able to make associations between a given prefix and an originating Autonomous System Number (ASN), between the ASN and the organization border router, between the border router identifier, a POP number, and its interfaces IP addresses and to provide information about the "meshines" of the network, i.e. to provide information about which border routers communicate, may they be from the same AS or different ones, with latency data included. This is, in its basic form, raw information obtained from iPlane files, further relations between the previously defined objects (prefix, ASN, POP) are implemented with the help and in the software. This module should be thought of as the provider of building blocks for the intelligent but otherwise physical topology unaware modules, which paradoxically, by means of this information will build further functionality aimed at easing "the life" of routers at topology level. Further details about the architecture of this module will be offered in section 3.3.

The first implemented mapping system was LISP-DHT, and it was done in the simulator module **Chord**. As presented in section 2.3.2, this involves, among other, building the Chord overlay with the restrictions required by a LISP mapping system. Furthermore, this component, as well as the other simulator implemented mapping system, acts as an Internet distributed database and has the main design goal of providing latency estimates for searches ITRs initiate when in need of a end-host-address to locator mapping. Before providing this service further functionality like the routing of the queries and some basic rules defining the query nature need to be implemented. More details can be found in section 3.4.

Next on the list is the **ALT** module. A theoretical presentation of the LISP+ALT Mapping System was given in 2.3.1 and the ALT module aims to implement, considering the limitations brought by the experimental nature of the system, all the ideas presented there. Moreover, after building the ALT topology, the module offers the possibility of computing the latency implied by an Ingress Tunnel Router initiated query. However, a clear view about the structure of a possible global ALT Topology does not exist and assumptions had to be made. For this matter, the author wishes to acknowledge the influential ideas of Olivier Bonaventure and Robin Whittle, which aided in the design of the ALT topology. Also, the LISP mailing list proved a valuable source of information and insight. Much more details related to how this mapping system was implemented can be found in section 3.5

The **ITR** module implements the functionality associated with an Ingress Tunnel Router. Specifically, it possesses the ability to transparently query any of the Mapping Systems for a identifier to locator mapping and subsequently install the result in its cache for system speed improvements. Moreover, as an experimental feature a packet buffer was implemented, to better comprehend the performance impact and memory requirement it would bring if such a buffer would be implemented in hardware. Both these requirements made necessary the implementation of a discrete "timeline" to better handle the time dependent nature of the cache and buffer insertions and extractions. It is worth noting that such functionality is transforming the simulator into an event driven one, more arguments will be given in section 3.1.2. The intricates of the ITR module are presented in section 3.6.

In an early stage of the simulator, after completing some elementary tests, it was concluded that a way of locally storing iPlane latency data was needed. The reasons behind this were two: decrease of the simulation times and reproducibility of results. As a result the **DBI** module was created to act as an interface to sqlite databases. To better understand the reasoning and the way this was implemented checking section 3.7 is recommended.

The module **Utilities**, as the name goes, provides various "commodities" when writing code for the other components. It was built out of the need to reuse some parts of the code without rewriting it in all the modules but to have a single point of "failure" for some of the offered methods. Two of the most important classes are the IPUtilites and the Timeline but details related to them can be found in section 3.8.

3.1.2 Simulator nature

The simulator presents a hybrid (or dual) nature because it contains both an event based block and a trace driven one. Specifically, the mapping system block takes as input mapping requests but it computes all the metrics and subsequently returns the results instantly, without generating an event when the resulting lookup latency time elapses. But, on the other hand, the ITR module reacts to the event generated by the arrival of a packet and at its turn controls its auxiliary modules (caches and buffers) by means of events. It event contains a discrete timeline updated by the timestamps from the received packets.

To be noted also that the topology generating modules themselves are trace based and not event based because in both cases the topology is considered as being static.

3.2 From Design to Implementation

In order to better understand what the simulator is capable of, why and how it does so, a review of the incremental growth it sustained during a three month period would be probably insightful. This section tries just that, it replays the chain of events which brought CoreSim to its current state.

3.2.1 The Problem

Obviously, the main driving force behind the implementation of such a simulator has to do with the Internet Advisory Board Workshop on Routing and Addressing from 20061.1. Not only did they state the problems the Internet would be facing in the next years but they ignited the IETF community's interest in solving these issues by means of a restricted, by then theoretical, solution space.

But this was not enough, it is indeed a necessary but not sufficient condition for a simulator's implementation. Mapping systems had to be "invented" and among the first to come were LISP+ALT and LISP-DHT. Moreover, their viability had to be proven, and both the LISP protocol and LISP+ALT mapping system have an experimental implementation in the Cisco IOS. A 20 node

experimental network is now deployed. And only now the arguments for writing the simulator can be given: *while the LISP partisans were busy defining and implementing the protocol they had no means to check if a large scale (global) implementation would be possible*. CoreSim has its own set of limitation and assumptions but the view it provides seems to be, to the author's knowledge, more complex than anything else written for this same purpose.

3.2.2 Program Requirements

Initially, the idea was to simulate one of the LISP mapping systems (LISP-DHT) and evaluate its performances for Global Native Mobility support. This seemed a natural step, considering the core/edge separation principle and the additional level of indirection, also required by native mobility support, brought by LISP. Supplementary, the feasibility of using the iPlane files for building a realistic Internet topology had to be considered.

3.2.3 Additional Requirements

Besides its main goal (as blurry as it was in the beginning) some additional requirements were imposed on the simulator. They had to do with the software architecture because future plans to extend the "end result" may exist. This translated in two requirements:

- *modular design*
- *code reusability*

A supplementary constraint was time, everything had to be finished in under three months.

3.2.4 Deciding upon a programming language

Multiple factors need to be put in balance when choosing the programming language for a given project. In the case of the simulator, some of these factors seemed to have more weight than others and obviously put some pressure on the decision. A very important limitation was the three months time frame, which corroborated with a level of uncertainty concerning the need of implementing new features resulted in a first request that the programing language should be a high level one. This also brings the advantage of code maintainability, if good programming practices are followed.

The other requirements, modular design and code reusability, asked for a language capable of supporting Object Oriented Programming (OOP).

The initial list included C++, Java and Perl. But after discarding C++ on the basis that it is a medium level programming language, Perl was chosen out of the two left. The reasoning behind it was that even though Java is a high level programming language Perl was expected to provide the same functionality with less code. To be noted though that the author had no prior interaction with this programming language and the decision did have, to some extent, a subjective character.

Indeed Perl is a high level programming language and does offer many features that ease the programmer's tasks, unfortunately at the expense of CPU and memory requirements. Some of these features include automatic memory management, strings, lists, hashes and dynamic typing but what the author did not know when work started at the project was that Perl OOP, though implemented in an interesting way, does not perform encapsulation the "classical" way and lacks interfaces. These drawbacks are very important when working in a team, but because only one person wrote the code for the simulator the inconveniences were almost indistinguishable.

3.2.5 Back to the drawing board

The work went according to plan and the *Topology* and *Chord* were already implemented when our focus on mobility abruptly shifted. The reason for this was a new collaboration opportunity with a research team lead by Olivier Bonaventure, at the UCLouvain, Belgium. They suggested that a comparison of mapping systems would be much more interesting, moreover a comparison between LISP-DHT and LISP+ALT could provide much insight for the IETF community, who as we speak, though presents a strong "core" supporting LISP+ALT, are somehow divided between more mapping systems. Hence, after this some considerable time was spent documenting the LISP+ALT topology, or better said, imagining a LISP+ALT topology, work began at its implementation. Several implementation phases existed, all distinguishable through their set of assumptions, but after a better understanding was built the final result came. The "calm period" was short lived, or dead at birth, because just as the work finished, professor Bonaventure proposed to implement a new mapping system, he invented, but based on the current DNS architecture. Work at a particular implementation of this last mapping system is finished, as these lines are written, but no simulations were run. The reasons couldn't be more simple, the time that a simulation for a one day trace file takes to complete is approximately one week. The three months (a little more in fact) have ended and work, at this thesis, is considered complete, at least from the author's point of view.

3.3 Topology Module

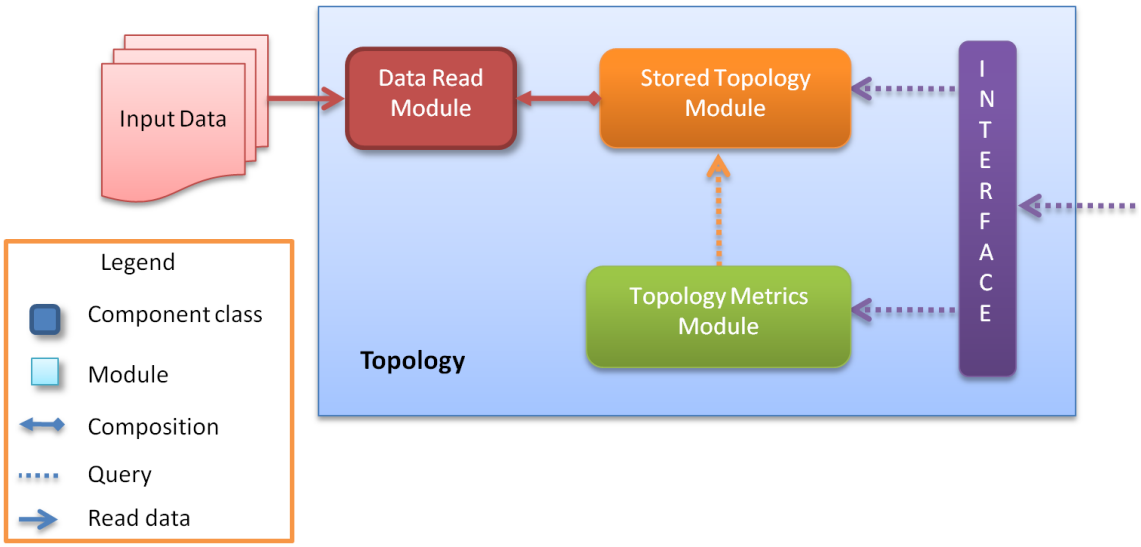

Figure 3.2: Topology Module Block Diagram

As previously mentioned in section 3.1.1 this module, shown in figure 3.2, reflects iPlane's view of the Internet and aids the other modules in building their specific topology. One of the requirements imposed on the simulator was that a "real" or as close as possible to the Internet "reality" topology be used instead of a synthetic one. The main benefit resulted out of this restriction is that CoreSim can be used to simulate the behavior of not any topology but the behavior of what would be tomorrow's Internet if LISP is to be deployed. In this context, one can easily judge the importance of the iPlane data but can as well question its veridicity. Hence, the next section provides a conceptual proof of trust.

3.3.1 iPlane

iPlane Project

From a design point of view, iPlane is a deployed application-level overlay network that generates and maintains an "atlas" of the Internet. It does so by adopting a structural approach, as opposed to common black box latency prediction models, through which it predicts end-to-end path performance with the aid of measured Internet segments part of Internet paths [MIP+06].

In order to build such a system iPlane adopts an incremental approach by splitting a complex function in its constituents. From this perspective the system is presented as doing three main functions: mapping of the Internet topology, measuring the Internet core and performance prediction. Moreover, these functions being complex in nature are also split into "simpler" more specialised ones.

The mapping of the Internet topology consists of two main steps: determining the points which will be probed by the distributed vantage points and the clustering of the router interfaces. The former is accomplished with the help of BGP snapshots collected from RouteViews [Mey], RIPE NCC [rip09] and others, which provide a rich set of globally routable prefixes within which iPlane supposes that ".1" addresses are router interfaces. The latter step makes good use of the observation that interfaces belonging to the same router or maybe same PoP behave in a similar way, thus their clustering is performed. For this job iPlane employs the Mercador [GT00] technique and a self developed one.

On the compact routing topology built at the previous step, where each node is a cluster of interfaces and a link connects two of clusters, iPlane starts measuring several link attributes. To insure unbiased results and load balancing among the nodes, an algorithm was devised to perform the task assignments to vantage points. A central entity distributes this workload and in turn accepts reports from all the vantage points with the scouted results. From the gathered data iPlane is able to extract the latencies of path segments but also infers other link attributes like: loss rate, capacity and available bandwidth.

All the above mentioned steps result in a static view of a small "piece" of the Internet. The number of vantage points (over 300) supported by PlanetLab [Pet] nodes is indeed impressive but they alone cannot build a full view of what the Internet has become, i.e. a general query between two random IP addresses can not be dealt with just "raw" data. Thus, the next step taken towards supplying an information plane is the performance prediction phase. This is done in two steps, the path prediction step and second the aggregation of link measured results. The path prediction composes observed path segments, the ones previously measured, to predict unknown paths that appear due to the general nature of the requirements from the service users. In the general case, an intersection point between the path from the source to a given BGP prefix and the one from a vantage point to the destination is done. If more such intersection points exist the one which minimizes both total AS hop count and the hop count before exiting the source domain (early exit intradomain routing policy) is chosen. The resulted path properties, as previously mentioned, are inferred through aggregation of constituent path segments.

For a better understanding of iPlane and detailed measurement results reading [MIP+06] is recommended.

iPlane Data

Even though iPlane provides a rich set of link attributes, for now, CoreSim makes use just of the latency measurements but this in conjunction with the Internet topology "map" offered on the project's site[M+]. Specifically, the simulator's Topology module makes use of the following files:

origin_as_mapping.txt : This file provides mapping information between a BGP prefix and the originator autonomous system. It is build by merging BGP feeds from RouteViews [Mey],

RIPE [rip09], Abilene [abi] and Dante [dan] in one large set of AS paths. The last AS on such a path is considered the owner of the prefix. The file format:

```
<prefix>        <origin_as_number>
```

ip_to_pop_mapping.txt : This file contains the mappings between IP addresses and the clusters to which they belong. As mentioned previously, iPlane considers all the interfaces which are found in the same AS and in the same location to form a cluster. The format of the file is the following:

```
<ip_address>   <cluster_number>
```

inter_pop_links.txt : This file contains iPlane's view of the Internet topology. The information here acknowledges the links between PoPs and by doing so also between ASes. Latency estimates are also given. The file format is:

```
<cluster1>     <as1>   <cluster2>      <as2>   <latency>
```

iPlane Query Interface

An important part of the simulator is the ability to compute the time an ITR Map-Request query takes to be solved. For such a feature, CoreSim requires to know all the latencies between the mapping system nodes involved in routing the query and such information is not stored in the above mentioned iPlane files. Yes, latency estimates between a very small subset of the routers mesh is provided but in the context of an overlay system latencies between random points in the Internet are required. This information is offered by iPlane by means of shared service for which they designed a query interface [MAKV07].

Out of the two possible query interfaces, SUN RPC and XMLRPC interface, the simulator uses the latter as it is well supported within perl, the used programming language throughout CoreSim. The XMLRPC interface, as described in [MAKV07], exports a single method, namely *iplane.query* which accepts as its arguments an array of paths. Each path is represented by a structure with two named members *src* and *dst*, which map to the source and destination addresses in string format. The *iplane.query* returns an array of structures as its response, where each structure in the array contains iPlane's predictions for one of the paths given as input arguments. Each structure in the returned array has the following named members:

- *src* and *dst* are strings that map to the source and destination IP addresses
- *aspath* and *clusterpath* are arrays of integers containing the AS level and cluster level hops on the path.
- *path* is an array of strings containing IP addresses of the router interface level hops on the path.
- *measorpred* is an integer set to 0 if this path exists in the measured atlas, or to 1 if the path had to be predicted.
- *latency* is an integer value set to the end-to-end latency.
- *loss* is a double value set to the end-to-end loss rate.

Out of these returned members the simulator makes use only of *src*, *dst* and *latency*.

To be noted that specific details concerning the iPlane server and perl examples of how to contact this service were kindly offered by Harsha V. Madhyastha.

3.3.2 iPlane is not enough

Unfortunately iPlane is not able to provide the latency between all the possible pairs of POPs it has in its files, therefore a latency estimator is used to provide values for the cases where iPlane does not provide a real measurement. In order to design the latency estimator a dataset that contains roughly 200k latencies between arbitrary pairs of PoPs has been used. This dataset is randomly split in two sets, one for training and designing the estimator and the another one for validation purposes.

In order to design an estimator for the latencies all the information that can be associated to a PoP is taken into account. In particular the aim is to correlate the the geographical distance between two PoPs and the latency between them. The geographical location is obtained using the MaxMind database. This database is open source and has an 99.8% accuracy at country level, 75% accuracy at city level (within a range of 25 miles), 22% accuracy at more than 25 miles, and 3% that the IP is not covered by such database.

First, in figure 3.3 it is show a 3D histogram of the relation between the distance and the latency of the training dataset. As the figure plots mostly, the samples are of PoPs which are at less than 5000km, and between 10.000 and 15.000km.

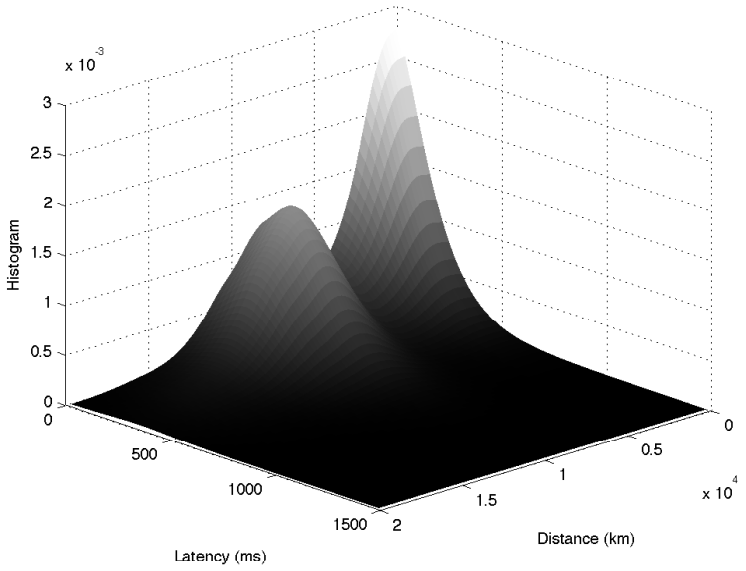

Figure 3.3: 3D histogram (latency vs. distance)

Now, figure 3.4 shows a scatter plot of the distance vs. the latency, along with a linear regression

$$Latency[ms] = 0.017203 \cdot Distance[km] + 65.968$$

As the figure shows there is a linear relation between both metrics, although the fit is not perfect. This is because the distance takes into account the propagation delay, however in a real path other terms affect the end-to-end latency, such as the queuing or processing delay. Unfortunately there are no public data traces that include this information (i.e. number of hops) into the regression. Therefore the linear regression is considered as the first estimator for the latency.

Another approach is also considered for the latency estimator. In this case, the pairs of PoPs depending on their distance are binned and the EDF (Empirical Distribution Function) of the latencies is computed. Then, considering this training data, an estimate for the delay of a pair of PoPs by first computing the distance between them is done, and then generating a random number

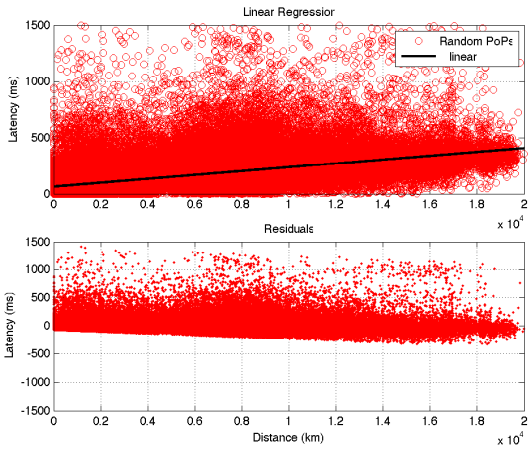

Figure 3.4: Scatter Plot and Linear Regression (latency vs. distance)

that follows the EDF of the appropriate bin. In particular two bin sizes are considered: (i) (0-10km, 10-100km, 100-1000km, 1000-10.000km, 10.000-20.000km) and (ii) (0-10km, 10-100km, 100-500km, 500-1000km, 1000-2500km, 2500-5000km, 5000-7500km, 7500-10.000km, 10.000-15.000km, 15.000-20.000km). With this approach it is assumed that there is a correlation between the bin size and the latency, for instance routers that are at a range of 10km may have the same amount of hops on their paths. Further, this approach is used by taking into account the particularities of their location at a continent-level. It is clear that the topology of the Internet is different if we consider North America or Europe, this is because Europe is more densely populated and routers at the same distance may have a larger amount of hops in between.

Figures 3.5 and 3.6 show the error and MSE (Mean Squared Error) of the above mentioned estimators. As both figure shows the performance of the estimators is similar, except for the $distance/c$ estimator, which always produces under-estimations. This is because it only considers the propagation delay, and assume that end-to-end paths are just a fiber link. Regarding the rest of the estimators the linear regression is slightly more accurate than the rest of them. Further, this estimator is very fast, and will not slow down the simulator. It is important to note that generating random numbers that follow a certain EDF is computationally intensive. Also, as figure 3.5 shows the linear regression estimator is not biased, and since the plan was to carry out large-scale experiments, with a high amount of repetitions, it was concluded that the results would not be influenced.

3.3.3 Design constraints

Several assumptions had to be made:

Prefix aggregation : The simulator filters out the specific prefixes, which are included into "wider" ones, even if they do not belong to the same AS for several reasons. First, for LISP-DHT in the case when two prefixes overlap at the high end IP address space of the less specific one the corresponding Chord instances ChordIDs will be identical and the overlay would fail to work properly. Moreover if a small prefix (say P1) segments the address space belonging to a wider one (P2), it will also become, by error, responsible for the lower key space belonging to P2. Second, some of the more specific prefixes are the product of traffic engineering and thus can be discarded because in LISP TE is attained through other means than specific BGP prefix advertisements. Thirdly, a common prefix space was desired for all three simulators.

Border PoPs election : In the simulator, the smallest prefix announcing unit is the autonomous

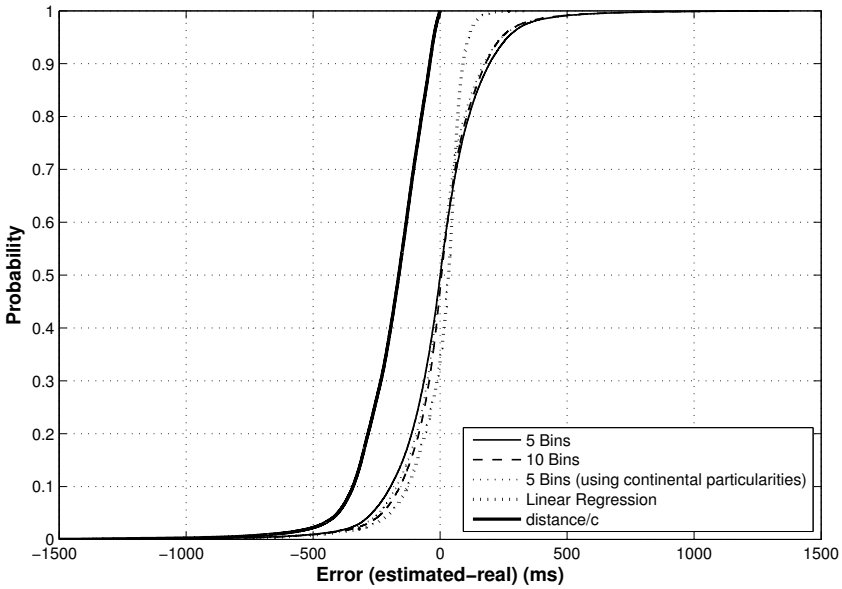

Figure 3.5: Error of the Estimators

system and thus its representant in the core routing domain had to be elected. Ergo, the number of most inter-AS connections was chosen as the election rule.

Symmetric Internet : As explained in section 3.3.2 iPlane does not provide answers for all the latency queries thus, considering that in the Internet core forward and reverse paths, even though sometimes topological different, are characterized by approximately the same latencies, the assumption of Internet symmetry was introduced. Consequently for CoreSim the latency between $IP-address1$ and $IP-address2$ is equal with the latency between $IP-address2$ and $IP-address1$.

Inter PoP latency : Through some conducted experiments it was observed that if, when computing the latency between two IP addresses belonging to two cluster interfaces, all the possible combinations of interfaces IP addresses are used, the success ration of the iPlane query interface can be improved. Therefore this feature was implemented in the simulator. This, though, makes use of the assumptions that iPlane correctly clusters interfaces and that the latency between any two clusters is the same, no matter what pairs of interfaces, with corresponding IP addresses, are used.

iPlane data : The simulator makes use of iPlane's data and algorithms/assumptions, among them the Internet topology, interface clustering results and latency estimates are of vital importance for CoreSim. It is thus accepted that any error from iPlane will propagate into CoreSim and probably bias its simulation results.

3.3.4 Internal components

As shown in figure 3.2 the main components of the Topology module are the *DataRead*, *StoredTopology* and *TopologyMetrics* classes which export their functions by means of a unified interface. The goal was to offer an object which would be able to read all the requested input files and then export functions which would permit exterior queries related to the topology structure and latencies between its nodes.

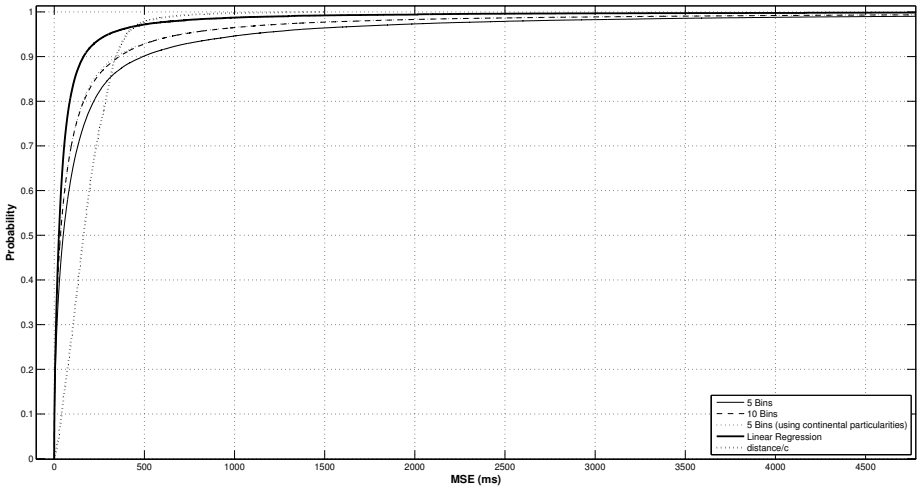

Figure 3.6: Mean Square Error of the Estimators

The first instantiated class is the *StoredTopology* one, which at its turn instantiates the *DataRead* class to read the requested input data. The most important read files are the three iPlane files but other preprocessed ones like the $filteredPrefixList$, used by the mapping systems to build their topology, and the Chord finger tables are read at this point in the program. The *DataRead* class was built to isolate the storage part, *StoredTopology*, from the "read" part and provide flexibility when/if the input source files change. Due to the format of the input data, hash structures are best suited to store it and most important search through it, thus after it is read, the *StoredTopoloy* object creates the necessary hashes and keeps them as "private" fields. Example of such fields are: $interPOPDelayMap$, $popIPMapping$, $prefixToAS$ which, as the name say, are used to map between the data from the iPlane files. Other such structures are used with the aim of speeding up searches between the stored mappings at the expense of system memory.

After the *StoredTopology* is instantiated the *TopologyMetrics* constructor is called with the *StoredTopology* as one of its parameters. Consequently the *TopologyMetrics* class will make use of the *StoredToplogy* when performing latency computations. This module can answer latency queries by performing several checks. The first step it performs, is to verify if the searched IPs belong to known routers and if they do, and the latency is known from the iPlane files the query is answered. If, on the other hand, the previous step does not provide an answer, the program proceeds to query the locally generated databases. These databases, *iPlane.db* and *estimator.db* store iPlane obtained or estimated latencies for speed improvements. If these queries also fail then two lists of IP addresses belonging to the routers interfaces are built. All the combinations between these IP addresses are computed and queries for all the pairs are sent to the iPlane XMLRPC interface. To increase the speed but also not to overwhelm the server, more pairs are inserted in one query.

As mentioned in the beginning of this section and in the figure 3.2, a "public" interface is offered for the rest of the components in a way that keeps them from knowing the inner components and variables of this module.

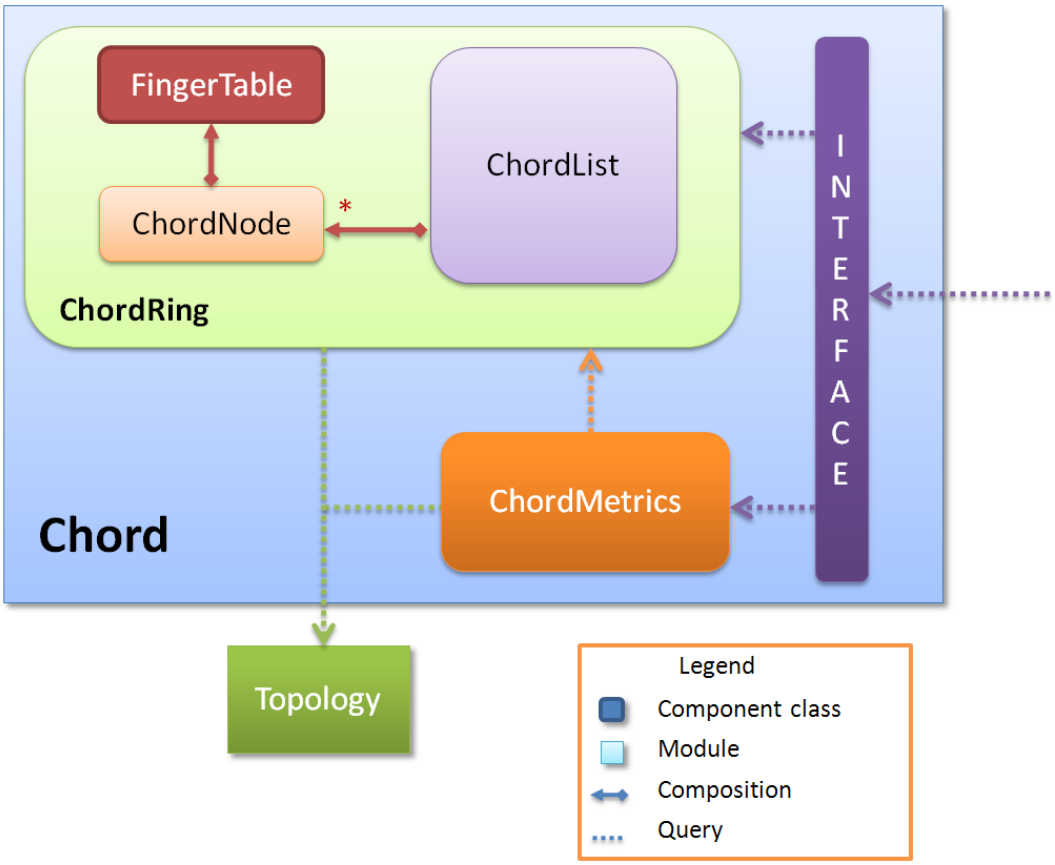

Figure 3.7: Chord Block Diagram

3.4 Chord Module

3.4.1 Topology Design Issues

The most important constraint to building the Chord ring is the necessity that all the Chord instances (called nodes from now on) must possess different identifiers(ChordIDs) on the ring. This brings forth the problem of overlapping prefixes, when, in the worst of cases the two prefixes might overlap at the high end of the wider one's address space. This is problematic because the modified Chord 2.3.2, used for LISP-DHT, chooses as Chord identifier for a given prefix its highest contained IP address.

To make things clearer consider the example below, from the figure 3.8, with the prefixes: 10.1.0.0/16 and 10.1.192.0/18. It can be seen that both prefixes have as their highest IP address 10.1.255.255. This overlapping of ChordIDs causes the Chord overlay to malfunction and therefore measures need to be taken. A good argument to delete the specific prefixes would be that they are used for traffic engineering purposes and in a LISP enabled Internet core such practices, of traffic engineering through BGP prefix deaggregation would not exist. As previously mentioned, TE will be attained by means of exchanged messages between the ITRs and the ETRs. On the other hand, a downside of aggregating specific prefixes into less specific ones is that some small clients will be lost. Unfortunately compromises had to be made.

Moreover, when overlapping appears not at the end of a prefix space but in the middle, the smaller prefix Chord instance "steals" Chord keys, or in other words responsibility for some addresses, from the wider prefix Chord node.

After the specific prefix filtering, when building the Chord ring, some assumptions regarding the clients nature, dynamic or static, had to be made. In a Peer-To-Peer (P2P) overlay the clients

Figure 3.8: Example of Prefixes Overlapping

will join and leave very often, conferring it a very dynamic nature, but in the context of a LISP mapping system stability is the desired and,arguably, attained state. Thus, it was decided to disconsider the transient regime that characterizes a P2P overlay, when the clients joining Chord split the keyspace among them and very often change finger table entries, to adopt a static topology where nodes never join or leave. This assumption holds true for the simulator's scope, as no changes are expected within the mapping system during a time period equal to a day.

Because of the above assumption the simulator was designed to build the Chord nodes finger tables in static way, taking advantage of the lack of changes in the topology. These finger tables are then saved to a file to further increase the speed of the simulator. It should be noted that building the Chord finger tables for a 112233 nodes is not an easy computational task and when running on 30 cluster nodes, this means each cluster node has to compute a bit more than 3700 finger tables, it still takes more than one hour.

3.4.2 Internal Components

Figure 3.7 shows the block diagram of this module. It can be seen that the most important components are the *ChordRing* and *ChordMetric* which practically represent the topology and the rules used to route queries on it. First instantiated is the *ChordRing* which makes use of the *Topology* module to build the nodes and afterwards orders them on the ring according to their ChordIDs. Because all the nodes are known, and no changes of the topology are simulated (static topology), the ring creation algorithm starts by sorting the prefixes in the descending order and then proceeds to instantiating the *ChordNodes* and adding them to the *ChordList*. The *ChordList* is a linked list which for the insertion of a node always requires the successor, thus the benefit of inserting nodes in reversed order. The network layer identifier of these nodes is always the IP address of the origin AS border router. To be noted that an autonomous system which is the originator of more than one prefix will have its border router enter with different ChordIDs for all its prefixes but their identity in the underlying network topology is the same, the border routers IP address.

Next there is the option to read the finger tables from a file if they are already computed or otherwise build them, but as previously mentioned this action takes some time to complete. By now the *ChordRing* is complete and the *ChordMetrics* class is instantiated with the *ChordRing* and *Topology* modules as parameters. This class, at request, can compute the path a query must follow in the mapping system and also the latency associated to the path length. More details about the routing and the metrics are given in the next section.

After all the classes are instantiated the ITR module, by means of an associated chord node, can use the unified interface to query the system about identifier-to-locator mappings and obtains as response the path followed by the query, in a segmented format, and also the latencies associated to each segment.

The UML diagram B.2 can be checked for a detailed view of the modules.

3.4.3 Routing and Metrics

The search algorithm used by Chord is defined in [SMLN+03] and is based on the property that finger tables entries have knowledge about nodes spread across the ring and thus can be used to pass the queries to nodes which have better knowledge about the owner of the searched key. The search ends when the predecessor of the key owner is queried.

A bit more detailed search would go like this: The query initiator looks at the searched key's ChordID and begins to search, in descending order, through its fingers to see which one has the first ChordID smaller than the one of the searched key. When that finger is found, the query is forwarded to it because it is supposed to have better knowledge about the searched key, given its proximity. He proceeds with the same algorithm of finding the first finger node with a smaller identifier than the searched ChordID, and subsequently forwards the query. This continues until one queried finger node has as successor the node responsible for the searched key and finally forwards the query to the key owner.

All these steps are done in the *ChordMetrics* module, which at each step pushes the queried node in a list. After the successor of searched key is found, the latency need to be computed. Two types of queries are supported by the Chord mapping system: *recursive queries* , when each node forwards the query to its adequate finger or *iterative queries* when each node replies to the query initiator, and the latter queries one by one all the nodes on the path to the successor.

The differences between these two approaches reside in the query latencies, namely the recursive type of queries is expected to provide slightly better lookup times. For the recursive type of queries the total lookup time is computed by adding all the segment latencies and the return one, from the successor to the query initiator. For the iterative queries the total latency is computed as the sum of all the round trip times (RTT) from the query initiator to the queried nodes. An advantage of the iterative type of queries would be their "support" for caching the mapping system infrastructure but such mechanisms have not been discussed or implemented for LISP-DHT but only for LISP-TREE.

An assumption made here is that, each AS will have it's own Map-Resolver and will be collocated with the ITR, same holds true for the pair Map-Server and ETR. Given the current state of facts these assumptions can be considered true.

In order to clarify the above ideas, consider the example in figure 3.4.3. This is a very restricted Chord overlay in terms of size and the identifiers are considered on 32 bits because they are IPv4 addresses. Some of the first things to be noticed is how the nodes are distributed in this topology, by this it is understood their relative position, and the contents of the finger tables. If the node with ChordID 33554432 and the prefix 1.0.0.0/8 is taken as example the following are observed: its first 25 finger table entries point to the same node, with prefix 2.0.0.0/8. This is because $ChordID + 2^24 = 50331647$ the ChordID of 2.0.0.0/8. The rule as previously mentioned is :

$finger[k] = n + 2^{k-1}$, where n is the ChordID of the node for which the finger table is computed.

Furthermore, the 26 finger is the node with prefix 50.0.0.0/8 because it is the first node, above of equal, with result of the above equation. The next five finger entries point to the node with prefix 128.0.0.0/8 and the last points to the node with 130.0.0.0/8 because it is the first node higher, in the ChordID space, than 129.0.0.0/8.

The figure also presents a simple query routing example for the LISP-DHT topology if recursive type of queries are used. A Map-Request comes to the node responsible for prefix 1.0.0.0/8 for the EID 150.0.0.1. This first node checks its finger table and notices that its last finger has an ID smaller than the ChordID of the searched node, 2516582401. Because of this, it forwards the query to the finger which at its turn checks its finger table. This node observes that his successor is the owner of the key and, again, forwards the query. The owner of the prefix, after it receives the

Map-Request, returns a Map-Reply, not shown here, to the initiator RLOC from the Map-Request.

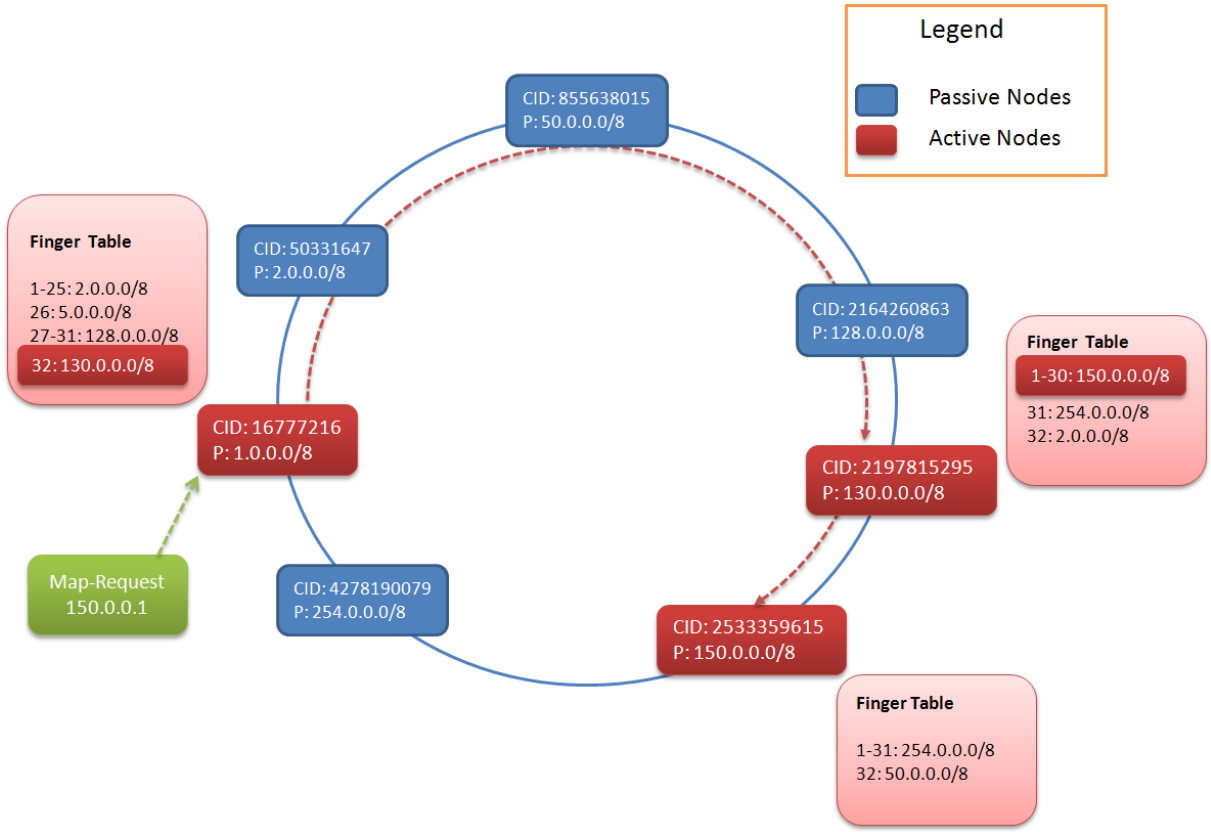

Figure 3.9: LISP-DHT Topology Example

3.5 ALT Module

3.5.1 Topology Design and Issues

At the time the simulator was designed, but also when the current paper was written there existed no clear view about how the Alternative Virtual Topology should look like.The main known and desired feature was and is to build it in such a way as to perform aggressive aggregation of ETR announced EID prefixes. To this cause, a hierarchical, tree like topology, with several layers of "aggregators" is suggested by the LISP+ALT draft[FFML09a] but exact details about how this should be built are not given. Because of this, people from the Routing Research Group (RRG) mailinglist, like Robin Whittle, have criticized[Whi] LISP+ALT's lack of details. He also offered a possible model for the topology (consistent with the mentioned constraints) and afterwards some inherent limitations. One of his ideas proposed that the high level layer (root layer) of the ALT topology should be fully meshed and the component routers should be responsible for very wide prefixes. In fact, the request to have very wide prefixes is correlated with the need to have a full mesh and this because the prefix size determines the number of root routers (a /3 mask implies 8 such routers). Therefore a too specific prefix would prohibit the mesh due to an unacceptable high number of GRE tunnels a root router would need to maintain. He then supposed the existence of a high number (five or six) aggregator levels, going down to prefixes like /20, but the envisaged topology asserted the existence of a very high number of ETRs (2^{24}), number to be expected sometime in the future but not in correspondence with today's reality.

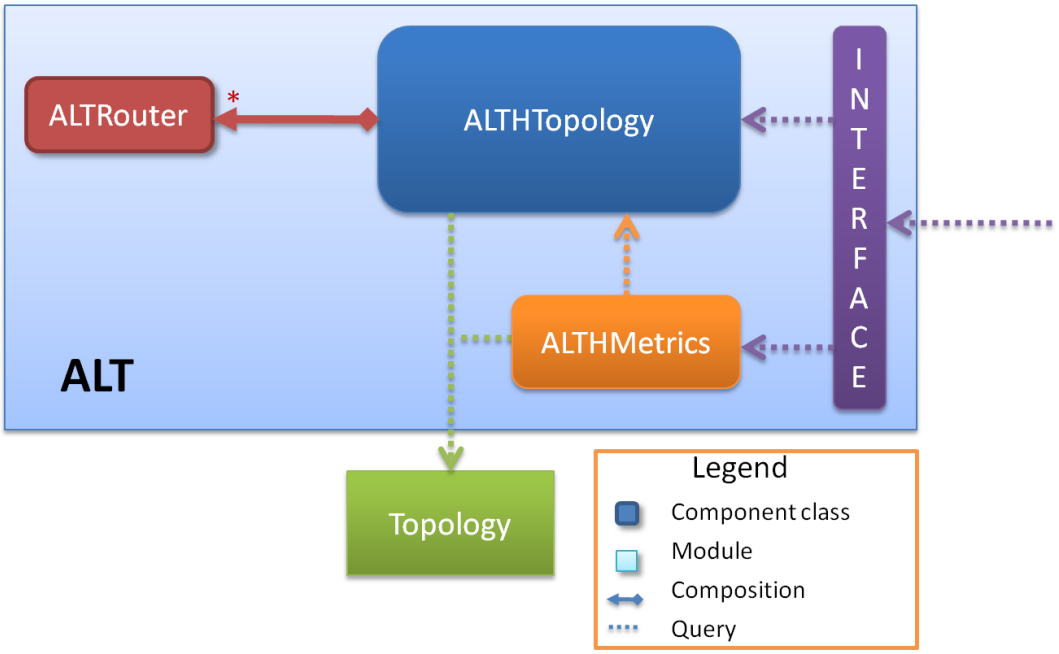

Figure 3.10: ALT Module Block Diagram

The first idea was the building block for CoreSim's ALT topology but the second was considered to be based on some future reality and not the current one. It was thus decided to use only the root layer with one layer of aggregators because today's Internet has just over 30000 autonomous systems, considerably less then the above value. This decision was confirmed after consulting with Olivier Bonaventure and his team.

The first layer of aggregators, deemed responsible for /8 prefixes, "collect", and aggregate, prefixes from over 100000 ALT routers which are "prefix injected" by ETRs. Within the simulator, the ITRs and their associated first router in the ALT topology are considered collocated, the same holds true for the ETRs and their associated first ALT Router. It was obvious that in this tree topology the leaf routers would be represented by the border routers of the prefix originating ASes but the root and aggregators had to be somehow "invented". The most "realistic" solution was to consider each root router as having an anycast IP address and being represented by two different PoPs. Anycast in IPv4 context can be obtained by BGP means. The PoPs were elected from the ones iPlane sees as best connected. The solution for the aggregators , the ones which had no corresponding BGP prefix announcements in iPlane files, was to consider them as topologically "endorsed" by one of their children nodes. This implies that one of their children would enter the topology both as an aggregator and as leaf.

3.5.2 Internal Components

The block diagram of the module is represented in figure 3.10 and depicts the relations between the component classes. As with LISP-DHT, a split is made between the topology and the metrics with the main benefit being represented by the possibility to rewrite one independent of the other.

When initializing the module, the first instantiated class is the *ALTHTopology*. This represents the tree like router topology presented in the previous section, consisting of a root router layer, a layer of aggregators and finally another layer which may be considered as built from Map-Servers. The code is not "hard wired" to this values and other combinations are permitted, but this was the topology deemed as the best reflection of a possible LISP reality. The topology building algorithm is split in two parts. First, the root and aggregators *ALTRouters* are instantiated. These root

ALTRouters are responsible for /3 prefixes and the aggregators are responsible for /8 prefixes. All the aggregators have as parent prefix one of the root routers, and all root routers have 32 child aggregators. Secondly, the leafs are created by instantiating for each prefix in the filtered prefix list an *ALTRouter*. For these routers the parent aggregator is found by checking its associated prefix, and subsequently they are added as children for their parents. This enables the parent routers to construct their Forward Information Base (FIB) by associating every prefix with their responsible router. Also every child adds a route to its parent. Supplementary, during this step the network layer addresses of the routers are set to further enable computation of latencies in this system. As previously mentioned, the leaves are represented by their associated AS border PoP, the root routers are considered "behind" anycast IP addresses, each associated to two of the best connected PoPs , and the aggregators, when the /8 prefix does not exist in the iPlane topology, are endorsed by child PoPs.

After all these steps are done the ALT routing topology is completed and any leaf router can now send messages/queries to any other leaf router. The initiator just has to send the query to its parent and all the routers on the path will know who the next hop is by using their FIB. The data structure used to store the FIB is a hash and the reason for using it is obvious, speed in searching through the entries.

The next step when initializing the ALT Module is the instantiation of the *ALTHMetrics* class. The resulting object is able to use the *ALTHTopology* to route Map-Requests to their destination ETR/Map-Server and the *Topology* module to find the latencies between the routers on the followed path. To find the latency associated to an identifier-to-locator mapping its public function takes as arguments the starting point in the ALT topology, otherwise known as the Map-Resolver, and the searched IP address. Details related to how the routing is done are provided in the next section.

3.5.3 Routing and Metrics

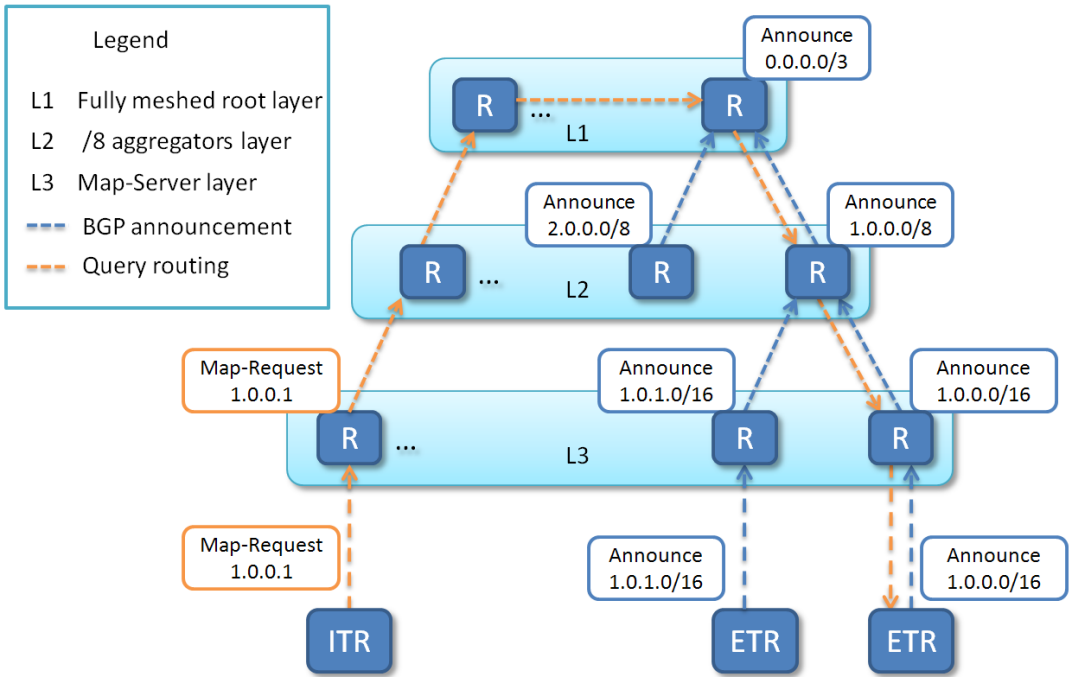

Figure 3.11: LISP+ALT Topology Example

The routing intelligence for the ALT Topology is held, in fact, by the Border Gateway Proto-

col. Therefore, in order to simulate the ALT Topology the Border Gateway Protocol needs to be simulated also, and this is the job of the *ALTRouter* module. When a router responsible for a wide prefix gets added a new child router, responsible for a more specific prefix, included into the first one, a new entry is created in the parent's Forward Information Base, a hash structure for Perl, that maps the child's prefix to the child router. In Perl parlance this means the hash structure gets a new key inserted and this key has as associated value the child router object, or blessed reference. Whenever a Map-Request is routed, the current hop router is asked to provide the next hop router by means of the method *findNextHopForIP*. This method call triggers a search in the FIB for the prefix which includes the searched IP address. If the current router has knowledge of such a prefix, the method returns a reference to the next hop router, otherwise it returns a reference to its parent, which at every point in the topology but the root layer, is considered to know more about the routing domain. A root router will always know the next hop root node when the request comes from one of its children and will always know the next hop child router when the packet comes from another root router.

When a Map-Request is received by the *ALTHMetrics* module, it will make use of the above mentioned mechanism to route the request, and as a result it will be able to compute the path needed to be followed to reach the destination ETR. This path, a list for perl, is used to compute the time the query spent in the system. Given the nature of the ALT Topology, i.e. a routing domain, only "recursive" type of queries are supported, thus only one possible way of computing this time exists: the sum of all the individual segment associated latencies. Anycast must be considered for the latencies between the aggregators and their parent route nodes and also among root nodes. CoreSim "simulates" anycast between two routers by considering that the lowest latency link, from all the possible ones between the two routers, is the one used for communication. More than one link exists because routers that are behind an anycast IP address are represented by more than one PoP.

For a better understanding of the above concepts consider the example in figure 3.5.3. As within the simulator there are three layers of routers. The first is the root routers, the the second is represented by the /8 aggregators and the third are in fact customer routers. In the right part of the figure, depicted with blue lines, one can see how routers from lower layers announce prefixes "upstream" to what in this paper are called "parent routers". Moreover, it can be observed that the second and first layers perform aggregation of the prefixes below them. Specifically, the ETRs are seen announcing the prefixes 1.0.0.0/16 and 1.0.1.0/16 to the router in the above layer, which at its turn announces 1.0.0.0/8 to the root layer 0.0.0.0/3 router.

The figure also depicts how the routing of a query is performed (see orange arrow path). To this cause, the ITR is seen forwarding a Map-Request, for EID 1.0.0.1, to its closest mapping system router, it can even be considered a Map-Resolver for the purpose of this example. Because the Map-Resolver knows nothing about the Map-Request (it is assumed that the Map-Resolver has no cache entry with EID prefix 1.0.0.0/16) it forwards it. The same reasoning can be used for the /8 aggregator and the query is now at the root router. Because the first layer is a full mesh, all the root routers "know" each other (by means of the Forward Information Base table), and most importantly they know each other's announced prefixes. Thus, the query is forwarded to the 0.0.0.0/3 root router. This is because the 0.0.0.0/3 prefix is the only /3 prefix that includes 1.0.0.1. Now the root router checks its FIB and will find the more specific prefix 1.0.0.0/8, which includes the searched EID, and forwards the Map-Request to router announcing it. The same happens in the 1.0.0.0/8 aggregator, which finds the specific prefix 1.0.0.0/16. At this point, the query has reached the Map-Server who knows the "owner" of the EID, or otherwise named authoritative ETR, and forwards the query on its last path segment. What follows is not shown in the figure, but the ETR is expected to reply to the initiator RLOC found in the Map-Request, through the Internet, not through the ALT overlay. The most important information found in the Map-Reply is the locator set which the ETR "advises" to be used when communicating with the 1.0.0.0/16 domain.

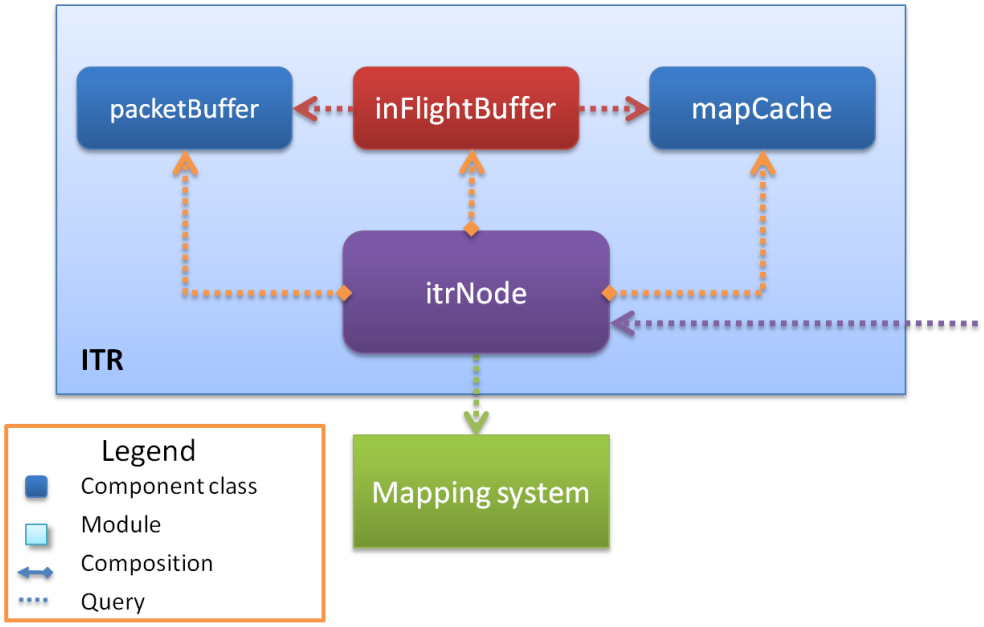

Figure 3.12: ITR Module Block Diagram

3.6 ITR

The Ingress Tunnel Router is thought to be a site local hardware component that deals with the encapsulation of the site's egress traffic with LISP headers and subsequently with the traffic's injection in the LISP core network. The LISP header contains as source address one of the ITR's RLOCs and as destination address the RLOC of an ETR. Therefore, it is also its function to perform EID-to-RLOC lookups in the mapping system for destination EIDs when they are not found in its local cache.

The ITR attaches to the mapping system either by peering with some of the mapping system's nodes or with the help of a Map-Resolver (see section 2.2.1). Either way, it will be able to obtain answers to Map-Requests from authoritative ETRs, when searching for the locators of given EIDs. Furthermore, caches are used to improve the time needed for solving Map-Requests. In the simulator this cache is called the *mapCache*.

CoreSim was supplementary designed to test for possibility of a packet buffer introduction. Thus, given a buffering policy, it will store specific packets belonging to a flow whose initial packet has not yet been sent by the ITR to its destination, due to an unknown ETR locator. The packets are flushed, sent to their destination, when the Map-Response comes. Such a feature requires that a buffer with all the sent and unanswered Map-Requests exists, CoreSim names it *inFlightBuffer*.

3.6.1 Internal Components

The block diagram of the this module is shown in figure 3.6 and it presents the way the *ITRNode* interacts with all its auxiliary components. The *inFlightBuffer*, *mapCache* and *packetBuffer* are, arguably, not required for the *ITRNode* to function properly as the definition for an ITR requires it just to handle mappings. But the above mentioned enhancements are required in order for the ITR to be a building block in a feasible system.

The first instantiated component is the *ITRNode* which takes as one of its parameters the node in the mapping system to which it should associate. CoreSim does not require, except for the TREE mapping system, the presence of a Map-Resolver. In an implemented system, such components are required to "translate" between the "language" of the ITRs and the "language" of the mapping

system nodes. CoreSim modules do not have such problems. After the *ITRNode* associates to the mapping system it instantiates the buffers and cache. Because iPlane does not know "all" the Internet core, a discard buffer had to be implemented to avoid searching more than once in the mapping systems for packets whose destination IP addresses cannot be mapped to locators.

As it will be explained in the next section all these components need to be aware of a virtual timeline in order to be able to react to events in a "time dependent" manner. This event based system also means the components exchange messages (exchange commands), but these will be explained better in section 3.6. From a functional point of view these components have the following roles: the *inFlightBuffer* stores information related to Map-Requests which have not been answered yet, the *mapCache* stores the mappings already solved which are not older than a given timeout, the *packetBuffer* stores packets from flows following a policy described in section 3.6.3 and the *discardCache* stores the IPs which have no mapping associated to the.

The perl data structures used to represent the buffers and caches, except for the *inFlightBuffer* are hashes. For the *inFlightBuffer* a new class has been implemented, *Timeline*, which is a time self sorting linked list. First versions of the simulator had used a hash also, but a search for expired entries in this hash had proven extremely time costly. On the other hand, the sorted linked list always keeps on the first positions the oldest of the entries and therefore the search time is much shorter and the simulator's speed is increased.

3.6.2 Time constraints

The time baseline for the simulator is provided by the packet trace which has timestamps embedded. These are discrete moments of time, recorded by the software or hardware equipment that created the trace, after a packet has been intercepted. Even though not a continuous flow, it can still provide a time reference for the simulator which uses every packet timestamp to update its own view of what time is. For every packet, the *ITRNode* uses the timestamp to update the *inFlightBuffer*, which at its turn is the initiator for a chain of events that result in the installation of new mappings in the *mapCache* and the flushing of the packets ,for which the Map-Responses have come, from the *packetBuffer*. In what follows a detailed description of the time dependent components is given:

inFlightBuffer : it stores information related to already sent Map-Requests but which are not answered yet. In fact, this simulates the time a request would spend in a real mapping system, until a Map-Reply returns, by acting as a time buffer. When one of the mapping system modules is sent a Map-Request it will answer, among others, with information related to the Map-Server and an estimate for the time the query has spent in the mapping system. This time estimate will be used by the *ITRNode* to compute a so called expire timestamp, practically the packet timestamp incremented with the estimate. Then, the *ITRNode* will insert a new entry in the *inFlightBuffer* and it will specify the time after which the entry needs to be removed and subsequently inserted in the *mapCache* by means of the expire timestamp. Also this class provides the possibility for the *ITRNode* to query it for the existence of an ongoing Map-Request.

mapCache : it stores the information gathered from solved Map-Requests, specifically the prefixes for which successfully queried Map-Servers are responsible. Information is added to this cache by the *inFlightBuffer* only. All the entries have associated to them the insertion time stamp which may be updated if further requests for their prefix come or, after a timeout period, be deleted. The *ITRNode* queries this module to see if there exists a stored prefix within the cache, which includes a searched for IP address. An important observation is that for the purpose of this paper the Time To Live (TTL) value for mappings was considered infinite, i.e. mappings are never refreshed.

packetBuffer : it stores, considering a policy, packets from flows which have requests in flight. The flow entries are flushed by the *inFlightBuffer* when it adds the Map-Reply information to the *mapCache*.

discardBuffer : it stores the IP addresses corresponding to unanswered Map-Requests. It also helps is speeding up the simulator by not performing unuseful searches in the mapping system.

3.6.3 ITR Cache policy and packet flow

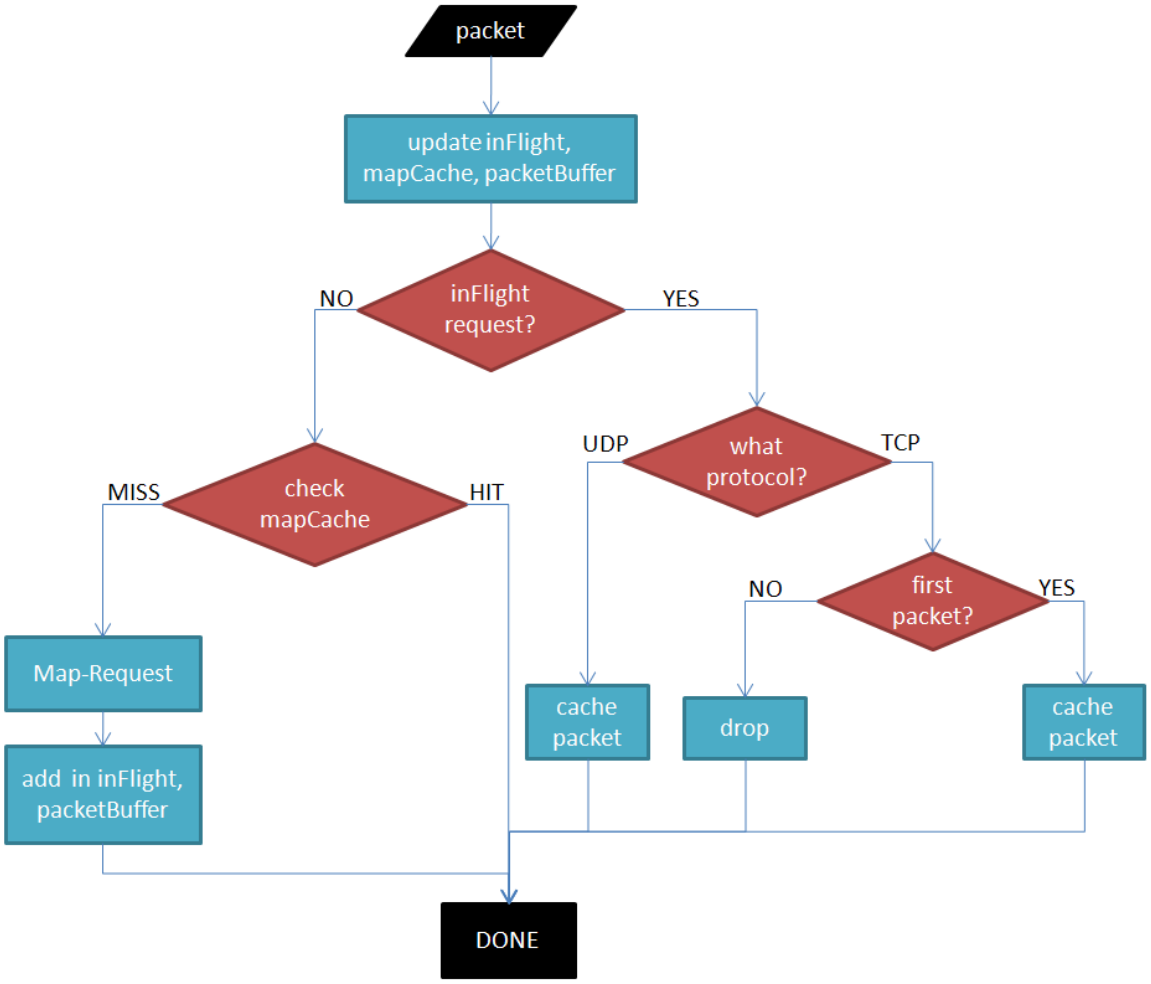

Figure 3.13: ITR Flowchart

When the ITR processes a new packet much depends upon the state of its cache and buffers. Thus, for a better understanding of the operations the ITR performs in figure 3.6.3 a flowchart is presented. Details related to how the updating of the buffers and caches are performed were given in the previous section, hence this section will treat only the caching and buffering policies.

The packet buffer is required only when, for a given flow of packets, a Map-Request has been sent but the Map-Reply has not yet been received. Thus, the first check in the ITR flowchart separates between two very different ways of processing the packet: first, if there is no "in flight" map request, a somehow classical cache check algorithm is performed and second, if the packet is part of an yet to be mapped flow, the buffering algorithm is applied.

The cache check algorithm starts by searching the packet's destination IP address in the *mapCache*. If a mapping is found, cache hit, the packet is considered forwarded, and the simulator

outputs it correspondingly. On the other hand if the mapping is not found, cache miss, a search in the mapping system is initiated, a new entry is added to the *inFlightBuffer* and the packet is buffered.

In case of an "in flight hit", the algorithm proceeds to checking the packet's protocol. In case the packet is UDP the action is to store it and the algorithm is complete. If the packet is TCP though, a supplementary check to see if it is the first in the flow is made. In the affirmative case the packet is buffered otherwise it is dropped. TCP is expected to use retransmissions for the dropped packets.

3.7 DBI

It might be too much to name this class an important module of the simulator. Even code wise it is small, and it does not have a crucial role, as the simulator can run without it, but it does bring significant simulation speed improvements. Moreover, it brings up the possibility to do latency consistent reruns of the simulations as iPlane dependent runs may be subject to latency changes in the Internet. *DBI* has the main purpose of interacting with the sqlite databases that have been deployed for storing the iPlane valid queries results. Furthermore, after some experiments the idea came run with the *iPlane.db* and *estimator.db* databases loaded into memory and thus avoid the harddrives seek times. Simulations speed improvements have been noted again. After *DBI* loads the two databases in memory, whenever new mappings come they are first stored in memory and afterwards written to the files.

The entry format for the two databases is:

```
<IP address 1>          <IP address 2>          <latency>
```

3.8 Utilities

This module is composed of many standalone classes, one written out of the need for code reusability and some out of the need for new data types. Out of them:

ipUtility : this class encapsulates all the methods used to perform operations with/on IP addresses throughout the simulator. It was also intended as a way of improving redesign time because changes need to be made in just one place.

Estimator : this class implements the latency estimator discussed in section 3.3.2

Timeline : this class implements a time self ordering linked list, used by the *inFlightBuffer*. Details are given in the next subsection.

Timestamp : this class provides means of operating with hexadecimal time stamps found in the packet traces.

IPPacket : this class implements a new data type used to store IP packets information.

3.8.1 Timeline

Two of the most important components are the *Estimator* and the *Timeline*. If the former was presented in great detail in section 3.3.2 the later will be presented in what follows. For a clear view of the public methods belonging to the other classes, checking the UML diagrams in section B is recommended.

The *Timeline* module, as previously mentioned, stores its data in a linked list. This list has been implemented in Perl with the help of hashes, which for this scope, always have as associated value to the key "next" a reference pointing towards the next element in the list. Supplementary, the insertions in this list are always done such that its elements stay time sorted. The variable used for sorting is the expire timestamp therefore , on the first positions of the list there will be the entries which are soon to expire, and at the end there are the ones which are expected to expire later. This enables the *inFlightBuffer* to make few checks when updating the *Timeline*, as opposed to the case when the entries are kept in a hash, and for each update a search among all of them is needed. As expected, this also brought down the simulation time.

3.9 Scenario

The simulator has for now only one simulation scenario which consists in replaying captured packet traces with the purpose of evaluating the behavior and performance for LISP-DHT and LISP+ALT mapping systems correlated with those of LISP Ingress Tunnel Routers (ITRs).

The simulator requires that for each packet in the trace four fields be supplied: *timestamp*, *destination IP address*, *protocol* and *packet size*. The usage of the first three fields has been presented throughout this paper, the forth is required to make estimates about memory requirements for the ITR's state.

Two trace files have been used, one from the Universite catholique de Louvain (UCL) in Netflow format, and the second at Universitat Politecnica de Catalunya (UPC) in the Endace Extensible Record Format, containing only packet headers. Both traces had to be converted to the text format required by the simulator.

3.9.1 UCLouvain trace file

One of the traces used for the evaluation was captured at the 1Gbps link which connects the UCL campus network to the Belgian National Research Network (Belnet). It is a one day unidirectional Netflow trace, saved on March 23, 2009 of the egress traffic. This consists of 752GB of traffic and 1200 million packets for an average bandwidth of 69Mbps. A total of 4.3 million different IP addresses in $123,804$ BGP prefixes have been contacted by $8,769$ different UCL endpoints during the 24 hours of the trace.

NetFlow generates level 4 flow traces but the simulator requires packet traces. Therefore this NetFlow had to be been converted into a packet trace: for each flow, the number of packets specified in the record were generated, distributed evenly across the time duration and the size of the flow. For example, if a flow lasts 10 seconds, has 5 packets and a size of 2000 bytes, the corresponding packet trace has a packet of 400 bytes every 2 seconds during 10 seconds.

3.9.2 UPC trace files

The second unidirectional trace used was captured at the 1Gbps link connecting the UPC campus network to the Catalan Research Network (CESCA) using specialized capture cards with microsecond timestamping precision, synchronized with GPS. It consists of the egress traffic on May 26, 2009 between 08:00-16:00 local time, and contains about 3700 million packets.

This is going on right now

Chapter 4

Experimental Results

4.1 Output Files Examples

This section presents the format of the output files

4.1.1 itrOutput File

Format and example:
[timestamp] [hit] [dest_ip] [pop] [as] [prefix] [latency] [peers] [node_latency_list]

```
49c6c64520000000 0 82.241.183.40 1336153 12322 82.224.0.0/11 389  5
                 193.191.16.25 29 130.242.82.193 107 216.98.120.38 159
                 64.213.23.29 188 213.228.3.179 0 213.228.3.179 13
                 193.191.16.25
49c6c6457c28f5c2 -1
49c6c645d89374bc
49c6c64634fdf3b6
49c6c646883126e9 0 81.247.35.66 284 5432 81.244.0.0/14 486 5
                 193.191.16.25 29 130.242.82.193 107 216.98.120.38 159
                 64.213.23.29 175 62.7.117.250 105 81.247.98.1 18
                 193.191.16.25
49c6c646916872b0
49c6c646edd2f1a9
49c6c6474a3d70a3
```

Details related to the fields:

timestamp : hexadecimal timestamp in epoch format

hit : it is 0 for miss, -1 for in flight hit

dest_ip : destination IP address

pop : destination pop

as : destination as

prefix : the BGP prefix which includes the searched IP address

latency : latency estimate

peers : the number of peers on the path towards the ETR

node_latency_list : list of peers and latencies

For the cache hit, or inFlight hit it is output just the `timestamp` and `hit` fields

4.1.2 mapCache File

Format and example:
[action] [timestamp] [prefix] [prefixes_in_cache]

```
add 49c6c64583958106 82.224.0.0/11 1
add 49c6c645d89374bc 82.224.0.0/11 1
add 49c6c64634fdf3b6 82.224.0.0/11 1
add 49c6c646916872b0 82.224.0.0/11 1
add 49c6c646edd2f1a9 82.224.0.0/11 1
add 49c6c647049ba5e2 81.244.0.0/14 2
add 49c6c6474a3d70a3 82.224.0.0/11 2
add 49c6c647a6a7ef9d 82.224.0.0/11 2
```

4.1.3 inFlightBuffer File

Format and example:
[action] [timestamp] [prefix] [searched_ip] [entries_in_buffer]

```
add 49c6c64520000000 82.224.0.0/11 82.241.183.40 1
del 49c6c64583958106 82.224.0.0/11 82.241.183.40 0
add 49c6c646883126e9 81.244.0.0/14 81.247.35.66 1
del 49c6c647049ba5e2 81.244.0.0/14 81.247.35.66 0
add 49c6c671cbc6a7ef 85.48.0.0/12 85.58.0.187 1
del 49c6c6728ccccccc 85.48.0.0/12 85.58.0.187 0
add 49c6c67571eb851e 201.206.160.0/19 201.206.185.118 1
del 49c6c675ee978d4f 201.206.160.0/19 201.206.185.118 0
add 49c6c67f7d70a3d7 89.152.0.0/14 89.155.108.92 1
```

4.1.4 packetBuffer File

Format and example:
[action][dstIP][timestamp][protocol][size][nbOfPacketsInBuffer]
[bytesInBuffer][nbTCPPackets][nbUDPPackets][tcpBytes][udpBytes]

```
add 82.241.183.40 49c6c64520000000 6 47 1 47 1 0 47 0
del 82.241.183.40 49c6c64583958106 6 47 0 0 0 0 0 0
add 81.247.35.66 49c6c646883126e9 6 681 1 681 1 0 681 0
del 81.247.35.66 49c6c647049ba5e2 6 681 0 0 0 0 0 0
add 85.58.0.187 49c6c671cbc6a7ef 6 54 1 54 1 0 54 0
del 85.58.0.187 49c6c6728ccccccc 6 54 0 0 0 0 0 0
add 201.206.185.118 49c6c67571eb851e 17 1041 1 1041 0 1 0 1041
add 201.206.185.118 49c6c67597ced916 17 1041 2 2082 0 2 0 2082
add 201.206.185.118 49c6c675bdf3b645 17 1041 3 3123 0 3 0 3123
```

4.2 Chord Implemenation Validation

The implementation of the Chord protocol for the LISP-DHT mapping system has been validated using the LISP-DHT implementation developed at Universite Catholique de Louvain, which is based on OpenChord 1.0.5

4.3 Mapping Latency

The *Mapping Latency* is the time required to obtain a Map-Reply for a map request to an EID. This latency is particularly important in the PULL model as it defines how long the traffic has to be ”frozen” towards a given destination while waiting for a Map-Reply.

In the simulator, the first step towards obtaining the mapping latency is computing the necessary exchange of messages triggered by a Map-Request and subsequently computing the path they must follow. Then, the mapping latency is computed as the sum of the latencies associated to path segments with the help of the iPlane dataset.

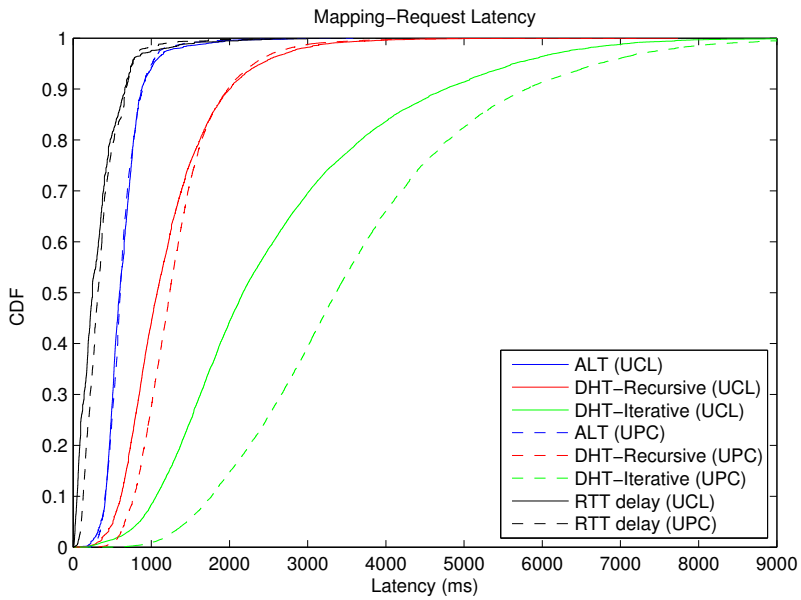

Figure 4.1: Mapping Latency distribution

Fig. 4.3 shows the cumulative distribution of the mapping latencies for the mapping systems of interest.

The curve tagged *Direct Delay* is a reference latency. This curve is the round-trip time, through the legacy Internet, between the source of the Map-Request and the node responsible for the mapping. This assumes that the Map-Request is sent directly by the ITR to the appropriate node and that the Map-Reply is sent directly to the ITR, without passing through any mapping system overlay.

Fig. 4.3 shows that, with a hierarchical deployment, LISP+ALT achieves better performances than LISP-DHT with a mapping latency closest to the direct delay.

It is also worth observing that the mapping latency in LISP-DHT is dramatically influenced, as expected, by the querying mode (i.e., recursive of iterative). While the median mapping latency is about 1 second in recursive mode, it increases to 2.1 seconds in iterative mode. The iterative mode induces more latency than the recursive mode because the ITR sequentially sends a message to each mapping system node involved in the Map-Request routing, until the node owning the

mapping is reached. On the opposite, the request is gradually moved towards the node owning the mapping with the recursive mode.

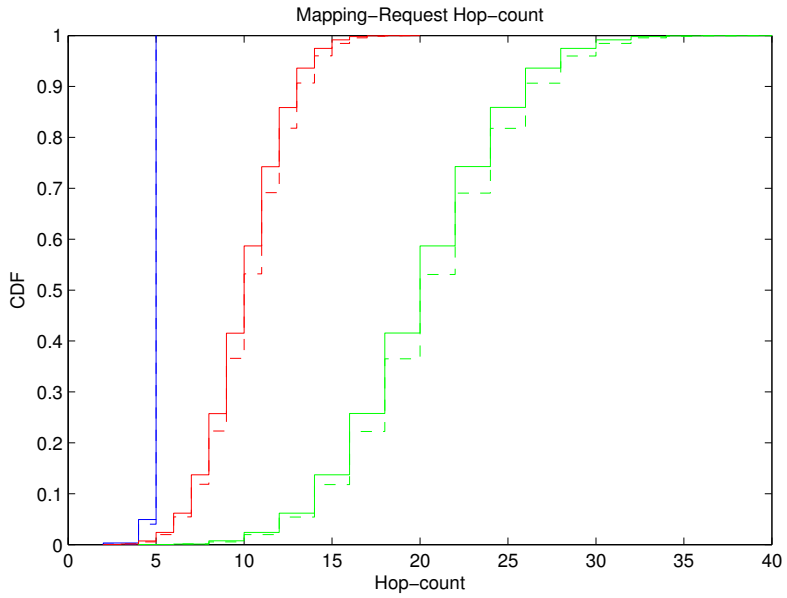

Figure 4.2: Hop count distribution

The maximum observed mapping latency is 3.6 seconds for LISP+ALT, 5.6 seconds for LISP-DHT recursive and more than 10 seconds with the iterative mode. In other words, in some cases, more than 10 seconds have to be waited before getting a Map-Reply in LISP-DHT Iterative while LISP-DHT Recursive requires two times less and three times less for LISP+ALT. In 95% of the cases, LISP+ALT has a sub-second mapping latency which is the same order of magnitude than what is observed for DNS in [JSBM02].

The main difference we can observe between LISP+ALT and LISP-DHT comes from the fact that, in LISP-DHT, nodes that are close to each other on the Chord ring are not necessarily topologically close. In contrast LISP+ALT tends to follow the Internet topology.

Globally, for half of the data-driven Map-Requests, 600ms have to be waited before the first packet can effectively be sent, representing more than **twice** of the direct delay. This latency being raised to about 2 seconds in LISP-DHT Iterative.

However, even if a 2 seconds delay is important, one could say that most of the traffic is TCP. Then, if a miss occurs with the first packet of the three-way handshake, the mapping will likely be installed in the cache before the retransmission of the first TCP `SYN` as the default retransmission timer is 3 seconds on most of the implementations. Moreover, if the other end of the new TCP flow does not have a mapping, another 3 seconds have to be waited because the first `SYN + ACK` will be dropped and that a new `SYN` has to be retransmitted. In that case, the three-way handshake is at least 6 seconds, whatever the mapping latency.

Nevertheless, data-driven Map-Requests are not always triggered by a TCP `SYN`. They can 0 for any packet inside an established flow . When a Map-Request is sent inside a flow, it is because the mapping has been removed before the end of the flow. In the simulator, a mapping is removed after three minutes without hit. In a real deployment of LISP, this would be after the expiration of the Map-Reply time to live. As a result, the flows towards a removed mapping fall into a blackhole. If this blackhole lasts few seconds, this can have a dramatic impact on TCP performances (i.e., window size reduction) or real time applications like VoIP. Therefore, minimizing the mapping latency is crucial.

Fig. 4.3 helps to better understand the mapping latency difference between the mapping systems. Fig. 4.3 shows the cumulative distribution of the number of hops necessary for a Map-Request to reach the mapping server (it does not include the extra hop for the returned Map-Reply) . The number of hops is not directly related to the mapping latency but each hop induces latency and it is likely that if the number of hops increases, the total latency increases too. The hierarchical deployment of LISP+ALT in the simulator limits the maximum number of hops to 5 but and 95% of the cases, this maximum number of hops is observed. In order to have a shorter path, the destination EID should be in one of the `/8` prefixes that doesn't have a more specific part announced separately. This corresponds to the second layer of the simulator's LISP+ALT hierarchy and a Map-Request would reach the responsible server after 4 hops in this case. If the ITR is in one of these `/8` networks, then the maximum limit would be 4 hops, but this is not the case for either of the traces. These networks mostly represent early allocations of IPv4 space to US universities and large US corporations and are not popular destinations in the UCL and UPC networks. This explains the hop number distribution in LISP+ALT. On the otherhand, the maximum number of hops is 17 with LISP-DHT Recursive and 34 with LISP-DHT Iterative. It is worse to notice that LISP-DHT, independently of the querying mode, needs more hops than LISP+ALT to resolve a mapping. This means that more nodes are involved in the resolution of a mapping when using LISP-DHT than when using LISP+ALT.

4.4 Cache Miss

A cache miss occurs when a packet has to be encapsulated by the ITR but it does not know a mapping for it. When a miss happens, the packet cannot be sent because the ITR does not know where to send the packet. However, due to the first miss for an EID, a Map-Request is sent and a mapping will eventually be installed in the ITR's EID-to-RLOC Cache for subsequent packet destined towards the EID.

The mapping latency has then an important role as it will influence the number of miss between the first packet seen by the ITR and the reception of the mapping. Sec. 4.3 shows that the used mapping system can dramatically influence the mapping latency. In this section, the interested is on characterizing the impact of the mapping system on the number of misses.

Fig. 4.3 shows the evolution of the miss rate with the time for the mapping systems of interest. The *x*-axis is the time of the day and the *y*-axis is the rate of miss (in miss/sec) in logscale. The reference curve tagged `UCL Trace (pps)` gives the packet rate (in packet/sec) of the trace.

At a first glance, LISP+ALT performs better than LISP-DHT and LISP-DHT is better in recursive than in iterative mode. This is consistent with Fig. 4.3 that shows higher mapping latencies for LISP-DHT than LISP+ALT.

Whatever the mapping system however, the miss rate is 200-500 time smaller than the packet rate meaning that most of the traffic uses mappings already installed in the EID-to-RLOC cache.

4.5 Node Load

Fig. 4.5 shows the distribution of the load on the different ALT nodes involved in the simulation.

The *x*-axis represents the load of the node. For the bottom layer, the node's load represents the amount of Map-Request addresses to it – the node generates a Map-Reply. For the upper layers, the node's load is the number of Map-Requests that have transited by it. The *y*-axis is the cumulative distribution of the load on the nodes.

Fig. 4.5 shows that an order of magnitude separates each curve with the most loaded nodes at the higher layers. A load up to $2.5 \cdot 10^6$ is observed at layer-1, the most loaded nodes presents a $1 \cdot 10^5$ at layer-2 and the load never exceeds $1,000$ at layer-3.

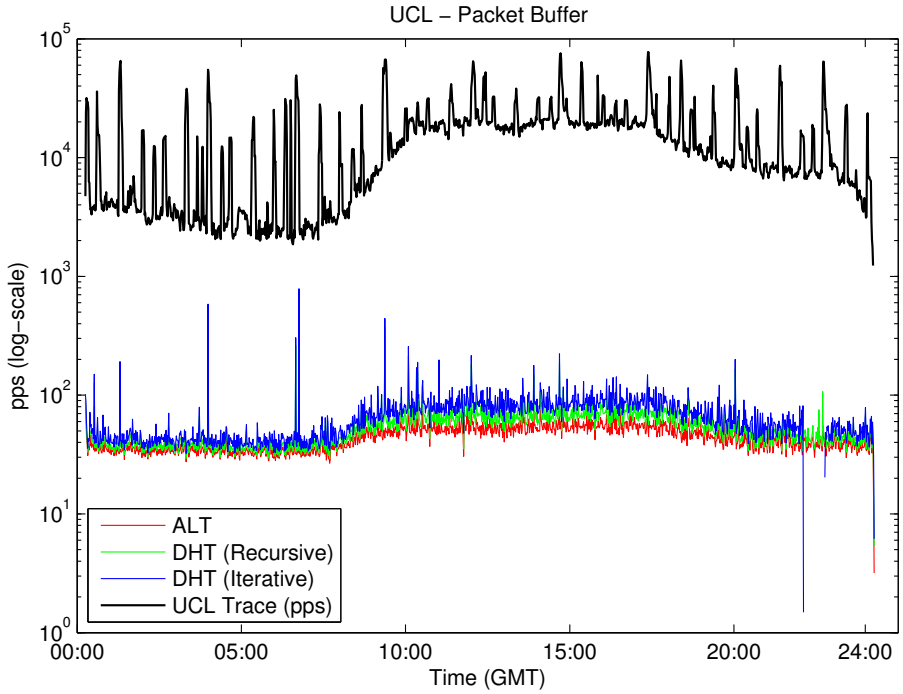

Figure 4.3: Evolution of the miss rate over time

The layer-3 shows that 10% of the nodes have a load of 1 meaning that, during the trace, they have been contacted only once. This situation is observed, in the simulator, when the EID is contacted either once or more than once every 3 minutes. The last case means that the mapping is never removed from the cache and, as the TTL is infinite in the simulator, no Map-Request is required. It can be observed that in virtually all the cases (99%), the load is below 200 meaning that the nodes process less than one packet every 7 minute.

The load between the nodes in layer-1 is even more unbalanced than in layer-2. The layer-1 consists of 8 logical nodes for the seven `/3` covering the routable prefixes. The most loaded node is 10 times more loaded than the second two most loaded nodes. And those second most loaded nodes are 10 times more loaded than the four least loaded nodes. the `224.0.0.0/3` node has no load as it covers multicast and reserved prefixes. This inequality comes from the approach we followed. We considered each IP address being queries with the same likelihood and thus uniformly decomposed the IP space with height virtual nodes, each one responsible of a particular `/3`. In practice, some part of the IP space are more contacted than other. Nevertheless, having a unequal repartition of the load among the different `/3` is not an issue, the logical node can be sustained by more or less nodes depending on the expected load, this is already the case in BGP where `.com.` is more important than `.be.`.

The load that nodes have to support is an important factor for the deployment of a mapping system. While the number of Map-Requests for a particular EID prefix cannot be controlled as it depends on the traffic over the Internet (i.e., the number of ITRs sending traffic towards the EID prefix), it is possible to design an architecture where the forwarding load is controlled.

In LISP-DHT, nodes owning mappings are also transit nodes meaning that a node owning an EID prefix `A/a` for one company is potentially a transit node for a prefix `B/b` owned by another company. This behavior comes from Chord where each node on the ring is at the same time a client, a server and a router for Chord messages. This property is very interesting in a P2P environment but is not recommended for a mapping system. First, from a business perspective, it is not a good property to have a potential competitor able to control part of a company's traffic. Moreover, a

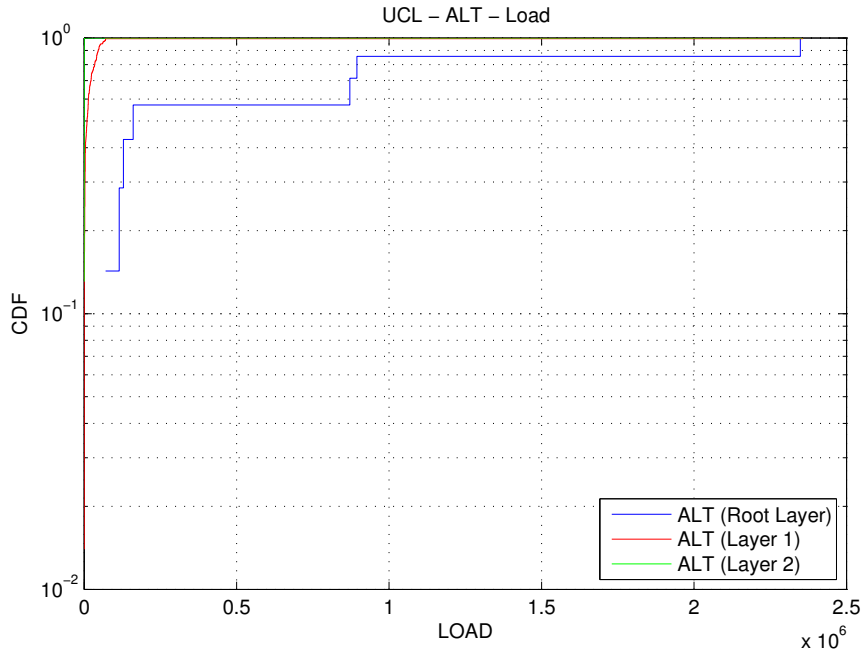

Figure 4.4: Distribution of the load on the 3 LISP+ALT Layers

small node can become the transit node of a big entity generating a lot of request. For example, the node exactly at the other side of the ring of a big ISP will be a transit node for up to half of the requests from this ISP but this node could be a mapping server for a small company. In LISP-DHT, it is hard to provision the nodes involved in the LISP-DHT topology as it does not only depends on the number of expected Map-Request/Map-Reply but also the position on the ring. The position on the ring defines the predecessors and this position cannot be controlled as it is defined by the EID prefix.

The approach followed for LISP+ALT is the opposite. The topology is composed of two types of nodes. The mapping nodes and the transit nodes. Transit nodes are in charge of forwarding the Map-Requests towards the correct mapping node. Mapping nodes own the mapping for their EID prefix. Transit and mapping nodes are separated: a mapping node is never used as a transit node. With such architecture, it is possible to provision the transit network according to the needs. Like in today's Internet, the transit network can be composed of several mapping service providers. From a business perspective, a company can chose a particular mapping provider and sign service level agreements to ensure the privacy and the performance. Entities owning the EIDs are supposed to maintain themselves the mapping nodes, but they could delegate this task to their mapping providers.

For the LISP+ALT deployment a hierarchical topology is considered (see Section 3.5). In this model, each node is responsible of a separate prefix and knows which are the node in the direct sub-layer responsible for one of its more-specific prefix. Progressively, Map-Requests are transmitted to the appropriate mapping node by moving from layer to layer, like in DNS. Which such decomposition, each node is only responsible of the load of one of its more-specific prefix. Thus, nodes can be provisioned according to the importance of their own prefix and not the whole network.

The UCL traffic trace resulted in the node load distribution presented in Figure 4.5. Darker lines represent fingers of the ITR at UCL, who was originating the Map-Requests. From the total number of $112,233$ nodes participating in the DHT, only $82,637$ nodes have routed or received Map-Request messages and are depicted in the figure. We can observe a sharp surge (even on a

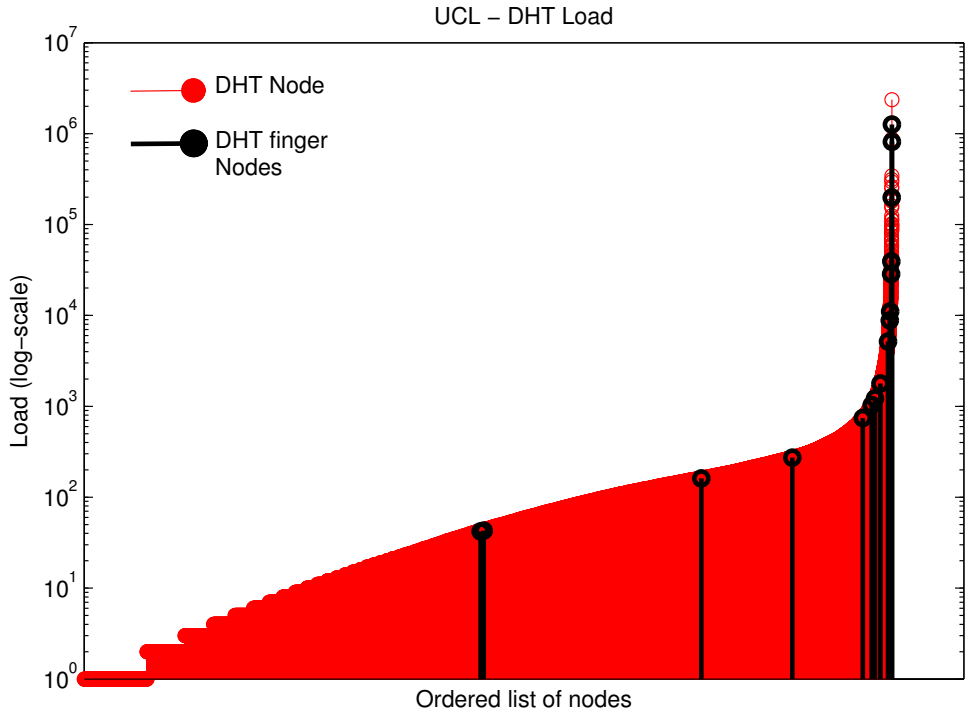

Figure 4.5: UCL LISP-DHT Ordered Load

log scale) after a load value of 1000, that accounts for about 1.8% of the total number of nodes from the DHT. As expected we find many of the ITR's fingers among these hotspots. Of the total number of 2,365,797 Map-Requests initiated, more than half pass through the last finger and one third the second last finger. But from the top ten most loaded nodes 7 are not fingers.

4.6 EID-to-RLOC Cache Size

The size of the routing table in today's Internet has become a problem [bgp09]. The issues mainly comes from BGP that install all the routes in the routing table, even routes that will never be used. With data-driven requests, the map cache to maintain on an ITR is proportional to its traffic. The ITR maintains a cache with the EID prefixes it has to map with RLOCs. Fig. 4.6 shows the evolution of the cache size with the time for the mapping systems of interest.

The day of the trace, 158,943 were announced on BGP according to RouteViews [Mey] meaning that, with the PUSH paradigm, this amount of mapping would have been installed on the ITR. However, the trace only contains 123,804 BGP prefixes which is 22% less. In other word, 123,804 is an upper bound of entries to install in the EID-to-RLOC cache.

However, LISP-DHT and LISP+ALT are using data-driven mapping and Fig. 4.6 shows that the number of entries fluctuates from 8,000 to 15,000 depending on the traffic carried by the ITR.

The work days traditionally start at 8:30am (7:30 GMT) and finish at 6:15pm (5:15pm GMT) in UCL.

During the night, the number of required entries in the EID-to-RLOC cache is about two third of what is required during the day. It is interesting to observe the peak lasting for about 1 hour at the beginning of the work day. It is caused by tasks that are done once at the morning by the people like reading the news and replying in batch to their mails.

A complete study of the mapping cache can be found in [IB07]

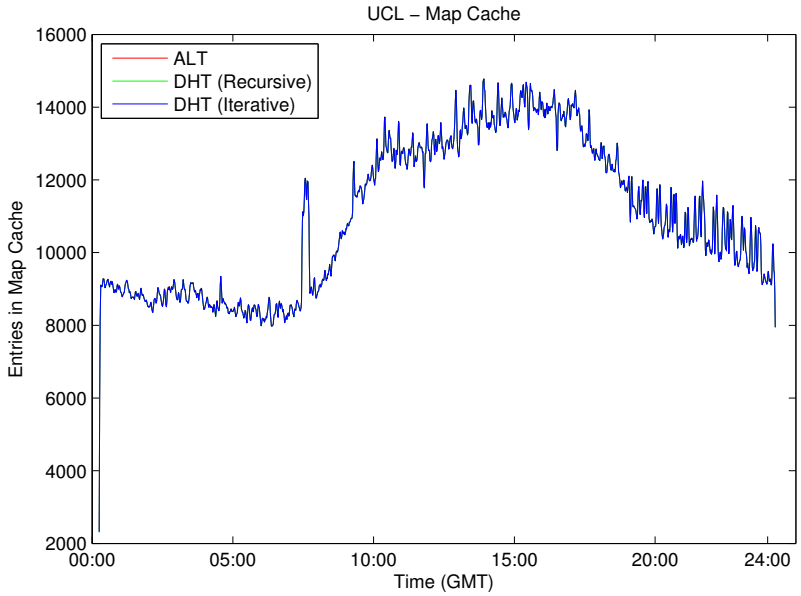

Figure 4.6: Evolution of the map cache with the day.

Chapter 5

Conclusions

Probably one of the most compelling arguments in validating the simulator is the similarity of ITR prefix caching results with Luigi Iannone's results in his CoNEXT'07 paper [IB07].

The performace of the individual mapping systems in terms of latency is not that important and arguments for this are the observed high cache hit ratio and the low number of packets in the packet buffer therefore the decision as to which of them to choose should be based on different criteria, such as the implied architectural advantages,

For normal traffic the buffering of packets seems to be a viable way to make up for the mapping lookup latency. It was also observed that the choice of mapping system does not influence the size of the packet buffer as much as traffic anomalies hence an IPS(Intrustion Prevention System) may be essential for the good functioning of a LISP Ingress Tunnel Router.

For TCP streams the mapping lookup latency influences only the first packet, thus the SYN will be slow but after the mapping is cached the influence of the mapping system won't be felt again. This does not hold true for UDP though where the flow could be controlled through another "channel" and this behaviour wasn't simulated. To better comprehend this future research is needed.

Finally, this paper has proposed a large scale simulator: CoreSim which was used for the evaluation of two mapping systems, LISP+ALT and LISP-DHT. Moreover, if one considers from an operational point of view the load as being very important then the simulations show that LISP-DHT doesn't do a good job at handling it. On the other hand with ALT it is possible to design the overlay in such a way that the load is controlled.

As future work, other mapping systems can be experimented with and the simulator can be extended to support more reqesting nodes at the same time.

Acknowledgements

First and foremost I must thank Loránd Jakab and Alberto Cabellos for their help and contribution in places where I had little or no knowledge and their guidance on the writing of this paper

I would also like to thank Olivier Bonaventure and Damien Saucez for their collaboration.

Special thanks to prof. Jordi Domingo-Pascual for accepting me at UPC and providing me the opportunity to work with such a great team and of course to my coordinator prof. Virgil Dobrotă for his wise guidance.

Last but not least I would like to thank my girlfriend Roxi and my family for bearing with me for the last 2 months.

Florin Coraş

References

[ABG03] J. Abley, B. Black, and V. Gill. Goals for IPv6 Site-Multihoming Architectures. RFC 3582 (), August 2003. 6

[abi] Abilene. http://www.internet2.edu/network/. 29

[bgp09] Growth of the bgp table - 1994 to present. http://bgp.potaroo.net/, 2009. 4, 5, 53

[Chi99] Noel Chiappa. Endpoints and endpoint names: A proposed enhancement to the internet architecture, 1999. 5, 10

[dan] Dante. http://rtmon.gen.ch.geant2.net/. 29

[FF09] Dino Farinacci and Vince Fuller. LISP Map Server. draft-fuller-lisp-ms-00, March 2009. Work in progress. 16

[FFML09a] Dino Farinacci, Vince Fuller, David Meyer, and Darrel Lewis. LISP Alternative Topology (LISP+ALT). draft-fuller-lisp-alt-05, February 2009. Work in progress. 18, 19, 37

[FFML09b] Dino Farinacci, Vince Fuller, David Meyer, and Darrel Lewis. Locator/ID Separation Protocol (LISP). draft-ietf-lisp-01, May 2009. Work in progress. 7, 9, 13, 15, 19

[FLH+00] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784 (), March 2000. Updated by RFC 2890. 18

[FM09] Dino Farinacci and David Meyer. Lisp internet groper (lig). draft-farinacci-lisp-lig-01.txt, May 2009. 9

[GT00] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for internet map discovery. In *Proceedings of IEEE INFOCOM 2000*, pages 1371–1380, Tel Aviv, Israel, March 2000. IEEE. 28

[Hin96] R. Hinden. New Scheme for Internet Routing and Addressing (ENCAPS) for IPNG. RFC 1955 (), June 1996. 7, 13

[IB07] Luigi Iannone and Olivier Bonaventure. On the cost of caching Locator/ID mappings. In *Proceedings of the 3rd International Conference on emerging Networking EXperiments and Technologies (CoNEXT'07)*, pages 1–12. ACM, December 2007. 53, 55

[ISB08] Luigi Iannone, Damien Saucez, and Olivier Bonaventure. Openlisp implementation report. draft-iannone-openlisp-implementation-01, July 2008. Work in progress. 9

[J+07] D. Jen et al. Apt: A practical transit mapping service. draft-jen-apt-01.txt, November 2007. 8

[JSBM02] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. Dns performance and the effectiveness of caching. *IEEE/ACM Trans. Netw.*, 10(5):589–603, 2002. 49

[lis09] lisp4.net. http://www.lisp4.net/, May 2009. 9

[M+] Harsha V. Madhyastha et al. iplane: An information plane for distributed services. 28

[MAKV07] Harsha V. Madhyastha, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane: Measurements and query interface. iplane_interface.pdf, June 2007. 29

[Mey] David Meyer. Routeviews. http://www.routeviews.org/. 28, 53

[Mey07] David Meyer. Update on routing and addressing at ietf 69. *IETF Journal*, 3(2), October 2007. 12

[Mey08] David Meyer. The locator identifier separation protocol (LISP). *The Internet Protocol Journal*, 11(1):23–36, 2008. 13, 14

[MIB08] Laurent Mathy, Luigi Iannone, and Olivier Bonaventure. LISP-DHT: Towards a DHT to map identifiers onto locators. draft-mathy-lisp-dht-00, February 2008. Work in progress. 22

[MIP+06] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane: An information plane for distributed services. In *7th USENIX Symposium on Operating Systems Design and Implementation*, pages pp. 367–380. USENIX, November 2006. 28

[MN06] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (), May 2006. 8

[MNJH08] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host Identity Protocol. RFC 5201 (), April 2008. 8

[MZF07] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984 (), September 2007. 4, 6, 7

[O'D97] Mike O'Dell. Gse - an alternate addressing architecture for ipv6. draft-ietf-ipngwg-gseaddr-00.txt, 1997. 5, 6, 11, 12

[Pet] Larry Peterson. Planetlab. 28

[Res99] E. Rescorla. Diffie-Hellman Key Agreement Method. RFC 2631 (), June 1999. 8

[rip09] Réseaux ip européens network coordination centre (ripe ncc). http://ripe.net/, 2009. 28, 29

[RLH06] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (), January 2006. 18

[Sal93] J. Saltzer. On the Naming and Binding of Network Destinations. RFC 1498 (), August 1993. 10

[shi] shim6 - ipv6 multihoming. http://www.shim6.org/, 2008. 6

[SMLN+03] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003. 21, 36

[van] Iljitsch van Beijnum. Shim6: What, why and when. http://www.iab.org/about/workshops/routingandaddressing/routingws-shim6.pdf. 6

[Whi] Robin Whittle. Lisp critique. http://www.firstpr.com.au/ip/ivip/lisp-links/#critiques. 37

[Whi07] Robin Whittle. Ivip (internet vastly improved plumbing) architecture. draft-whittle-ivip-arch-00.txt, July 2007. 8

[Whi09] Robin Whittle. Ivip - a new scalable routing and addressing architecture for the internet, May 2009. 8

[Zha06] Lixia Zhang. An overview of multihoming and open issues in gse. *IETF Journal*, 2(2), 2006. 12

Abbreviations

AS	Autonomous System
ASN	Autonomous System Number
BGP	Border Gateway Protocol
DFZ	Default Free Zone
DHT	Distributed Hash Table
EID	Endpoint Identifier
ESD	End System Designator
ETR	Egress Tunnel Router
FIB	Forwarding Information Base
HI	Host Identities
HIP	Host Identity Protocol
HIT	Host Identity Tag
IAB	Internet Advisory Board
IETF	Internet Engineering Task Force
IS-IS	Intermediate System to Intermediate System
ISP	Internet Service Provider
ITR	Ingress Tunnel Router
Ivip	Internet Vastly Improved Plumbing
LISP	Locator/Identifier Separation Protocol
LSI	Local Scope Identity
NCC	Network Control Center
OOP	Object Oriented Programming
OSPF	Open Shortest Path First
P2P	Peer-To-Peer
RFC	Request For Comments

RG	Routing Goop
RIB	Routing Information Base
RLOC	Routing Locator
RRG	Routing Research Group
RTT	Round Trip Time
SHA-1	Secure Hash Algorithm-1
STP	Site Topology Partition
TE	Traffic Engineering
WG	Working Group

Appendix A

Results

Results

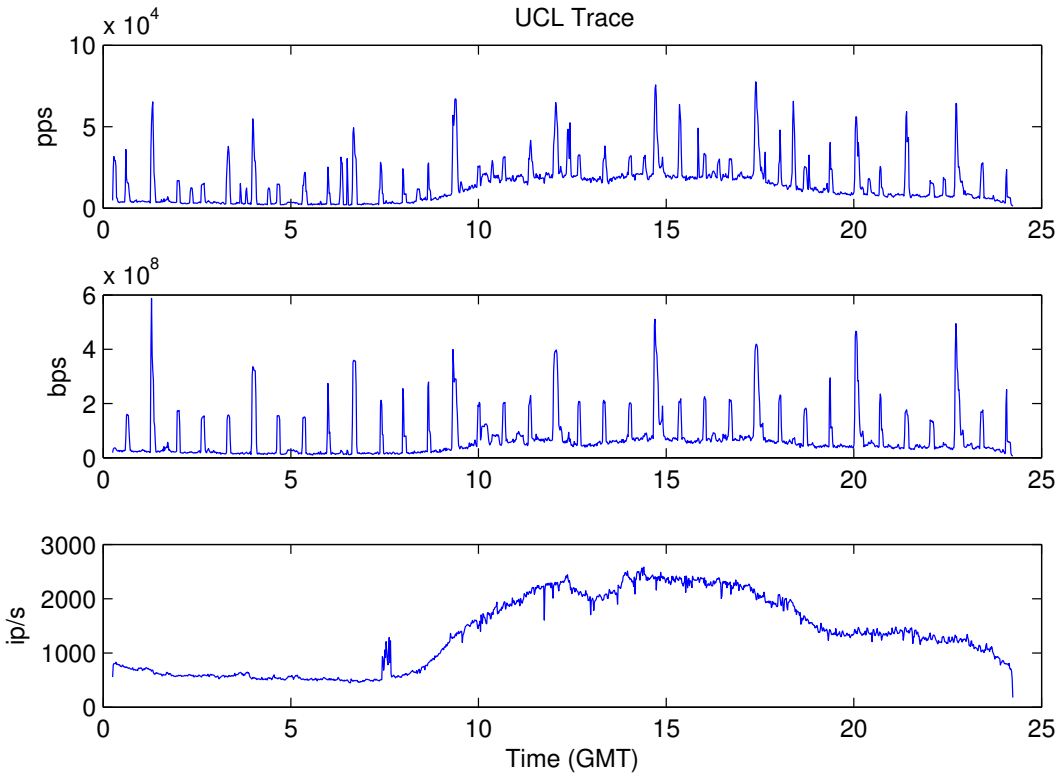

Figure A.1: UCL Trace File

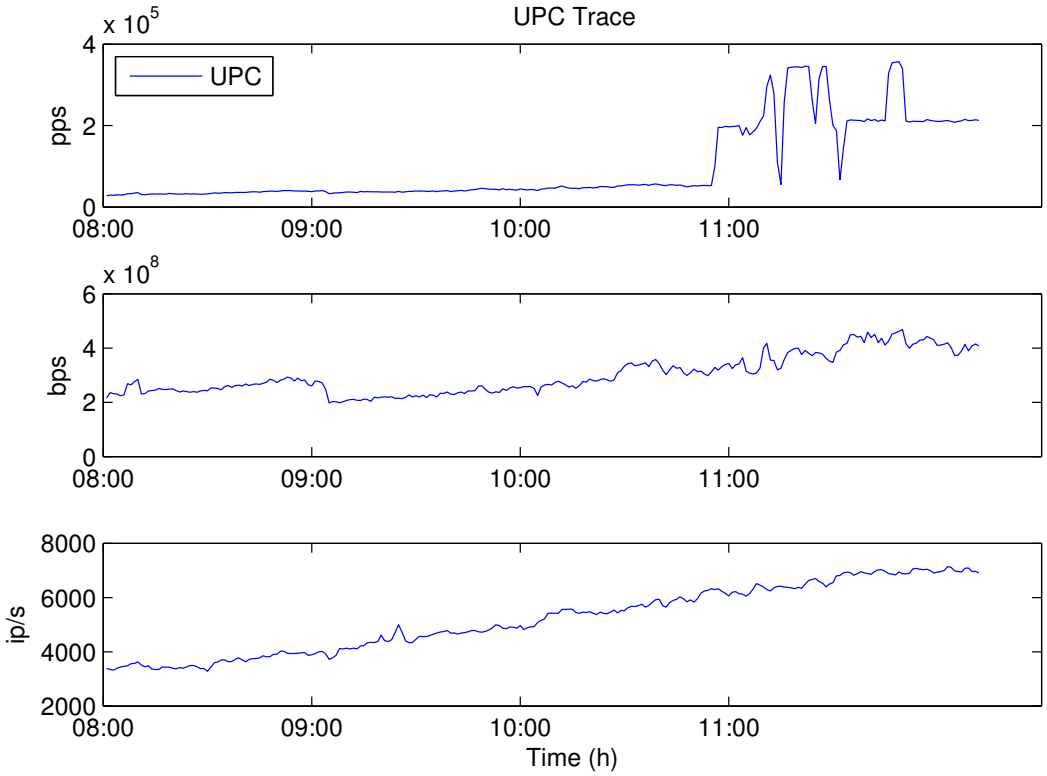

Figure A.2: UPC Trace File

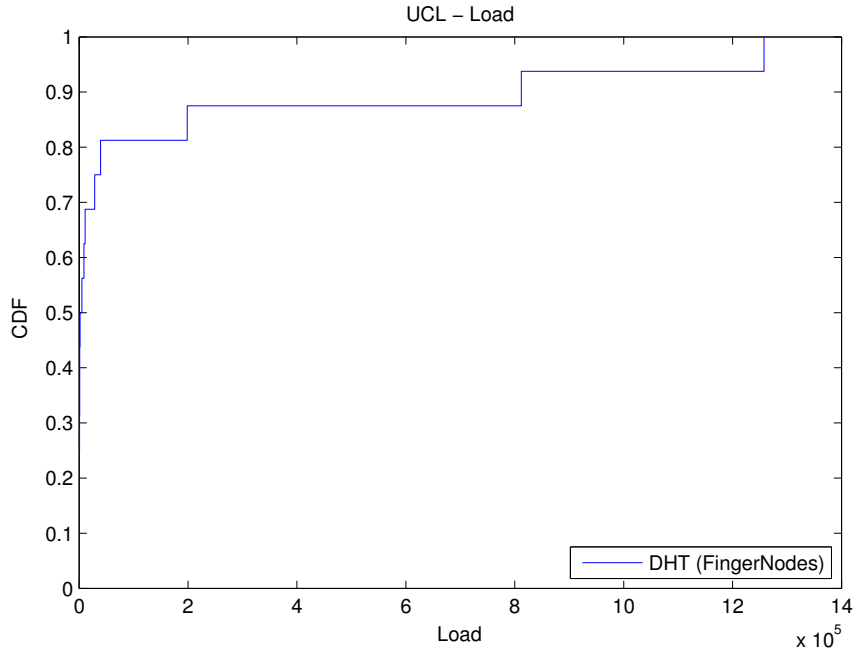

Figure A.3: UCL LISP-DHT Fingers Load

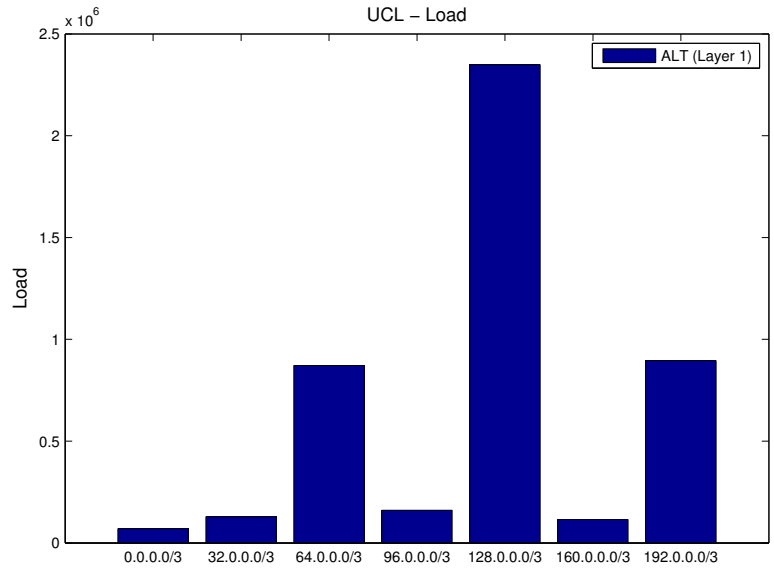

Figure A.4: UCL Distribution of the load on layer 1 in LISP+ALT

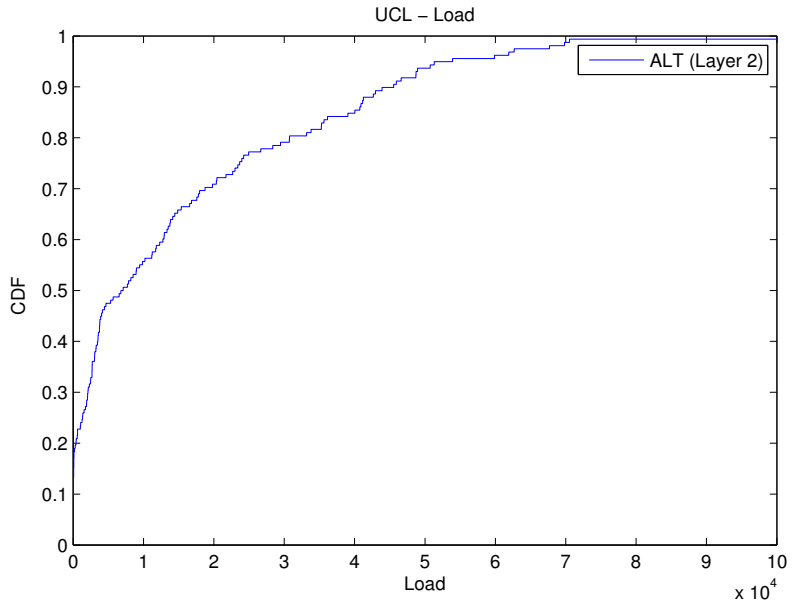

Figure A.5: UCL Distribution of the load on layer 2 in LISP+ALT

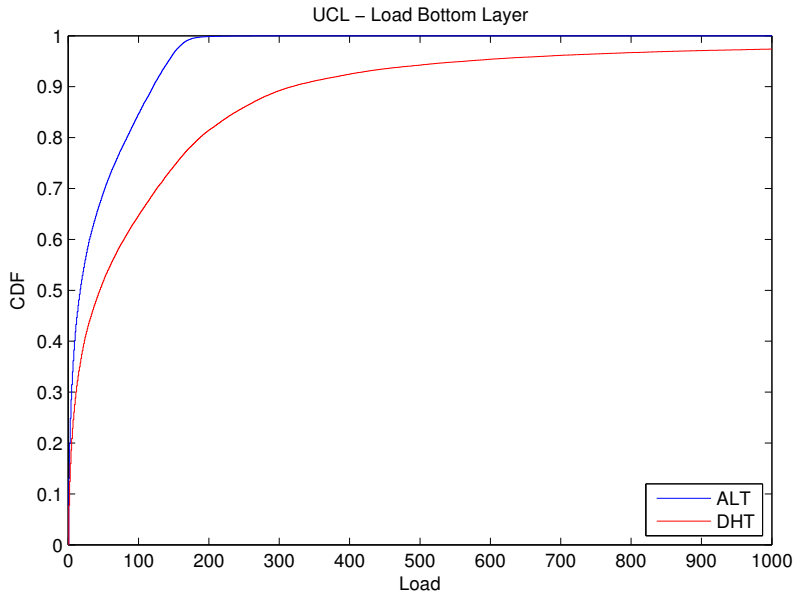

Figure A.6: UCL Distribution of the load on layer 3 in LISP+ALT and LISP-DHT nodes

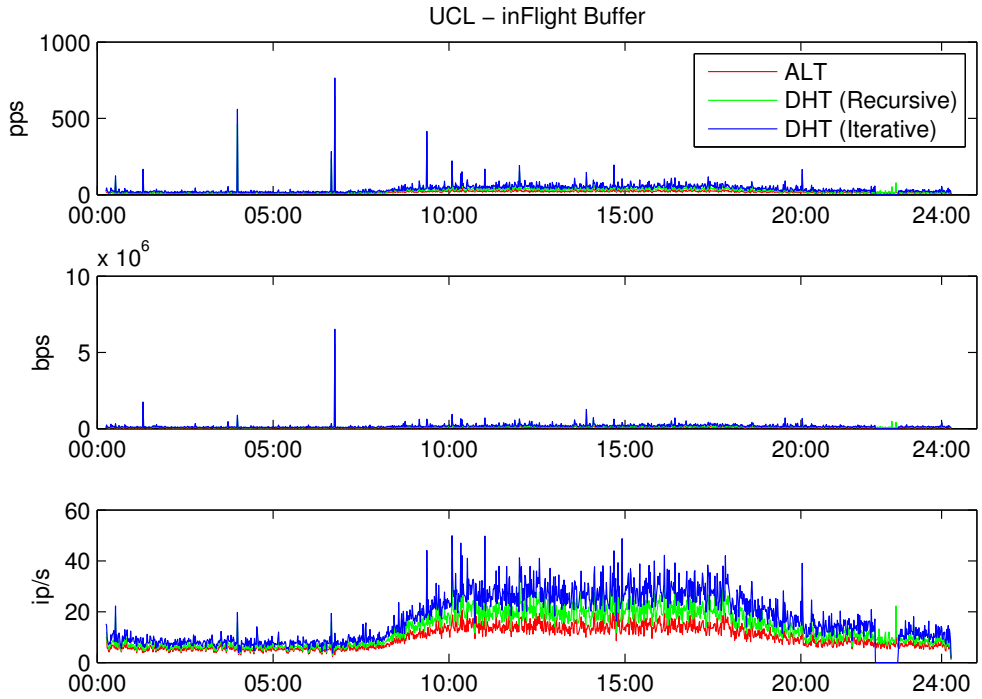

Figure A.7: UCL inFlightBuffer

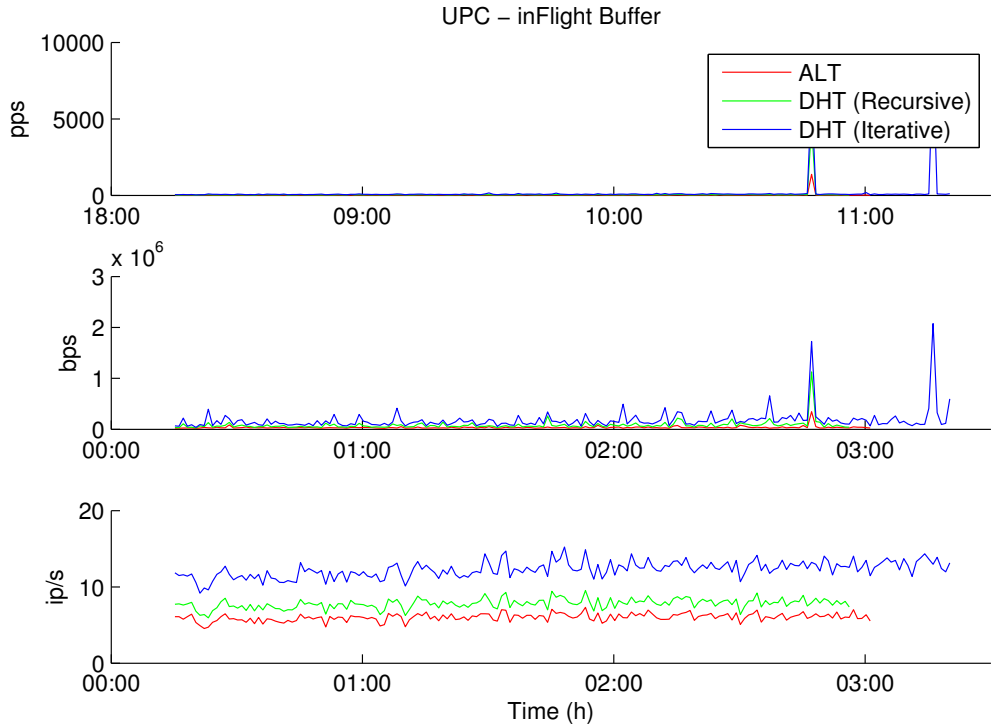

Figure A.8: UPC inFlightBuffer

Appendix B

UML Diagrams

Topology UML Diagram

Chord Module UML Diagram

ALT Module UML Diagram

ITR Module UML Diagram

DBI Module UML Diagram

Utilities Module UML Diagram

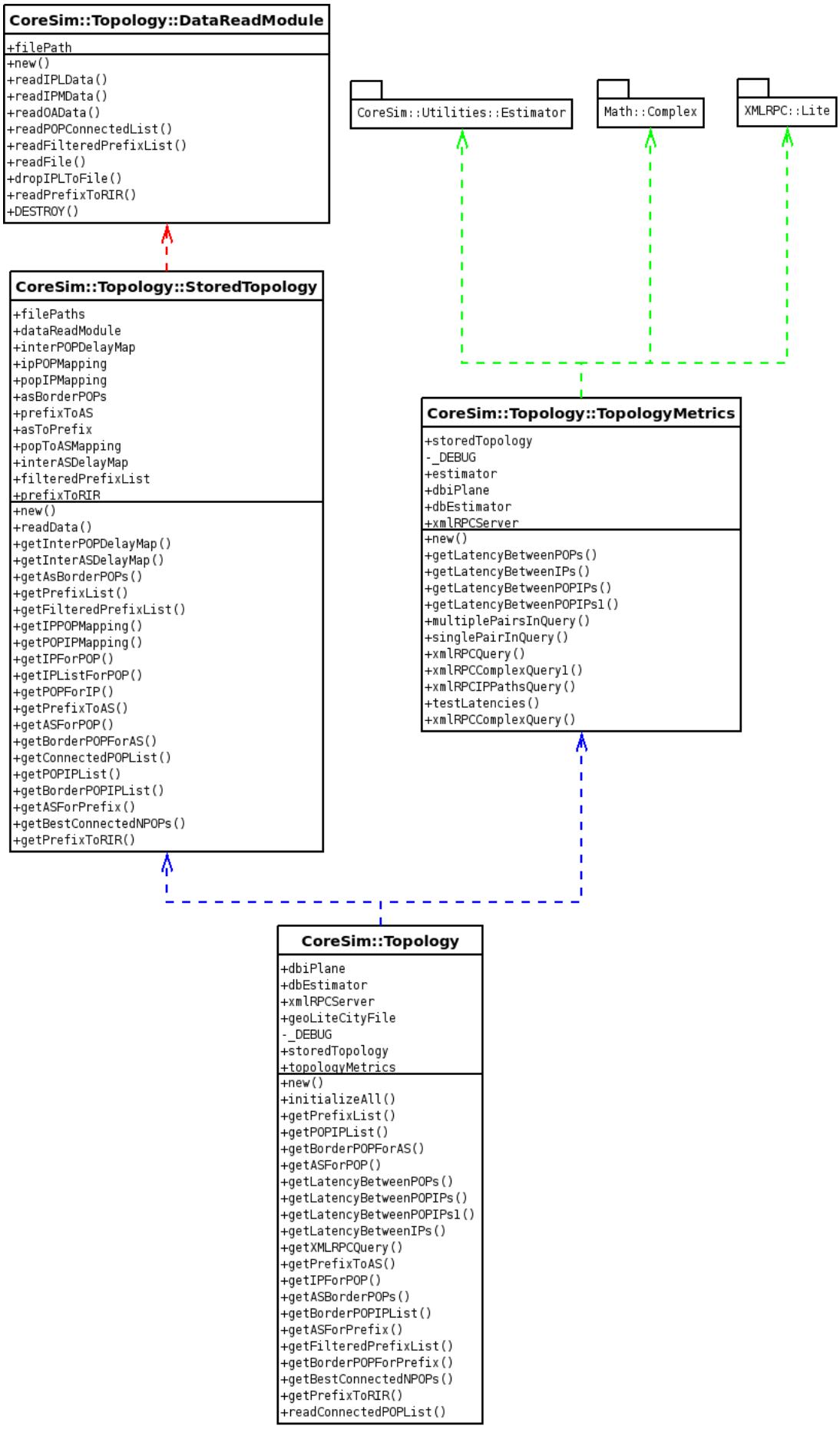

Figure B.1: Topology Module UML Diagram

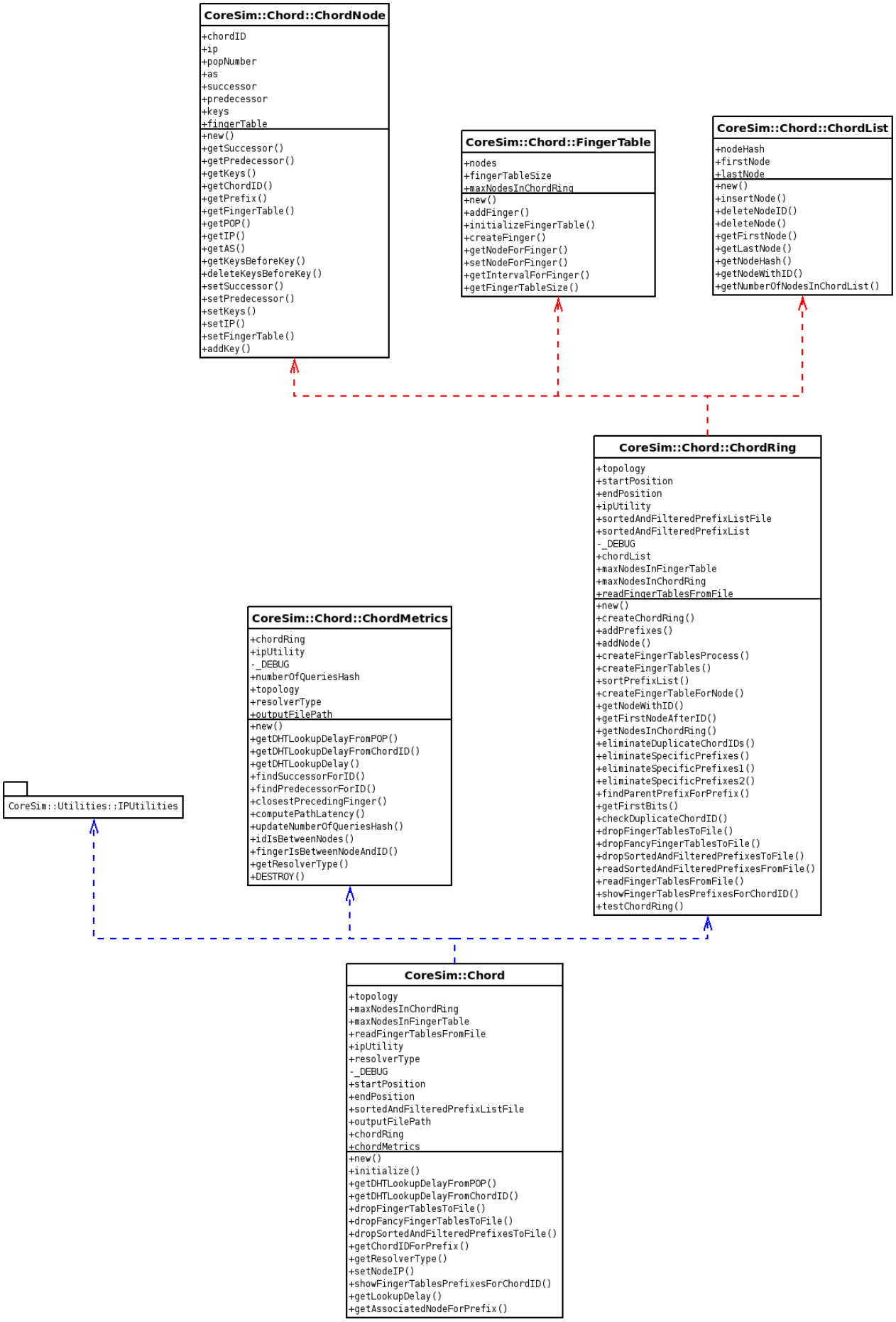

Figure B.2: Chord Module UML Diagram

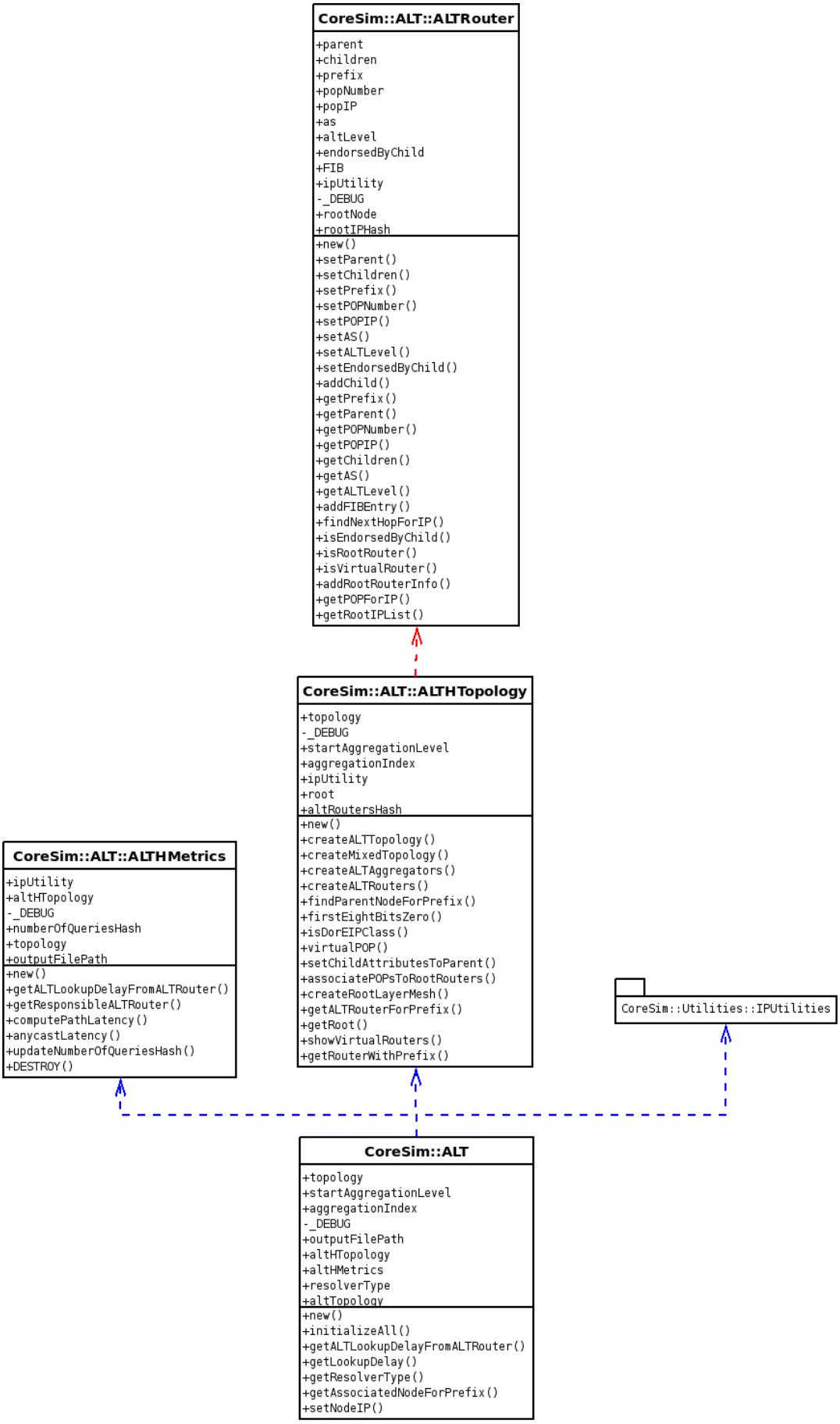

Figure B.3: ALT Module UML Diagram

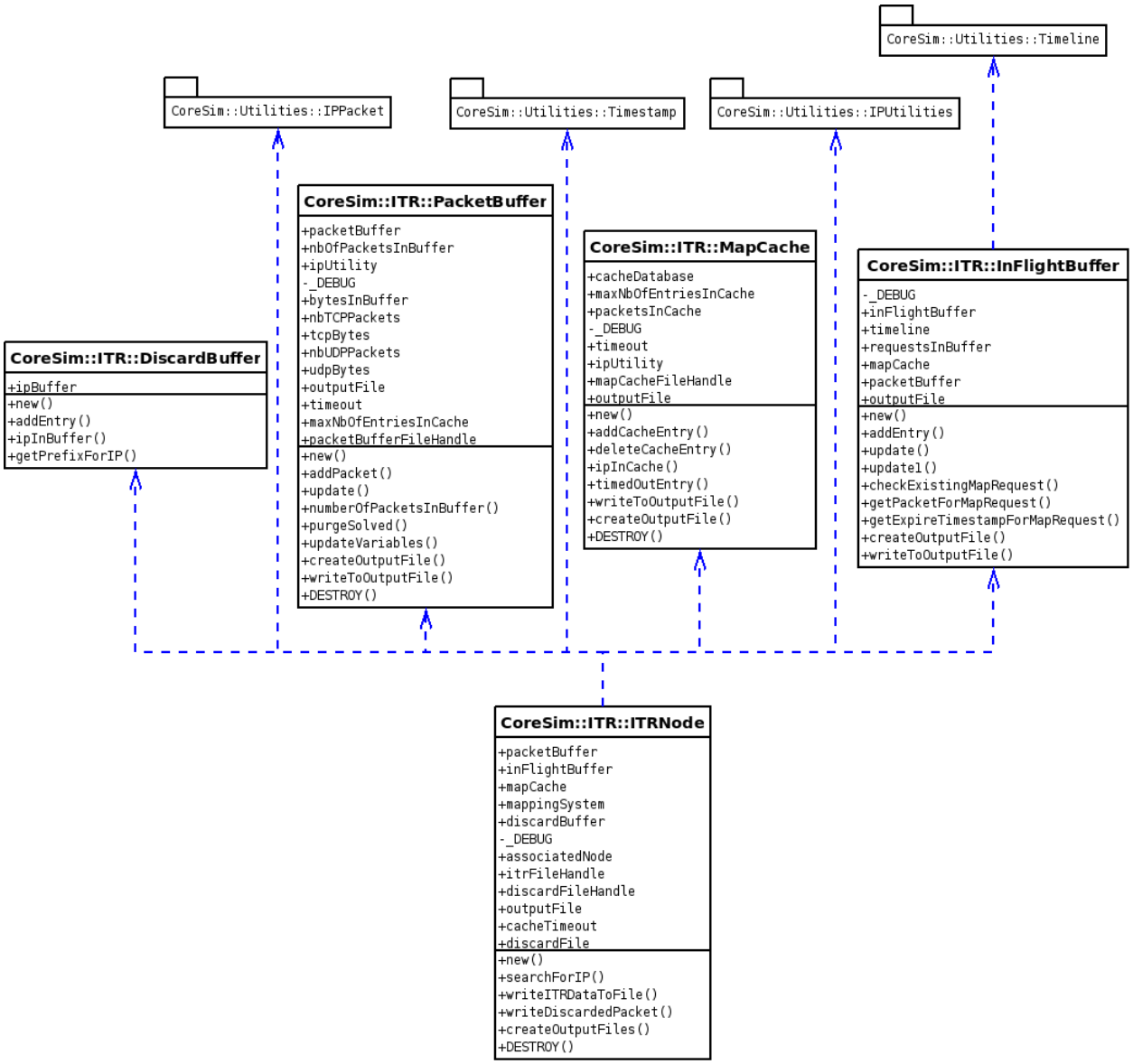

Figure B.4: ITR Module UML Diagram

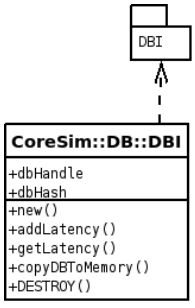

Figure B.5: DBI Module UML Diagram

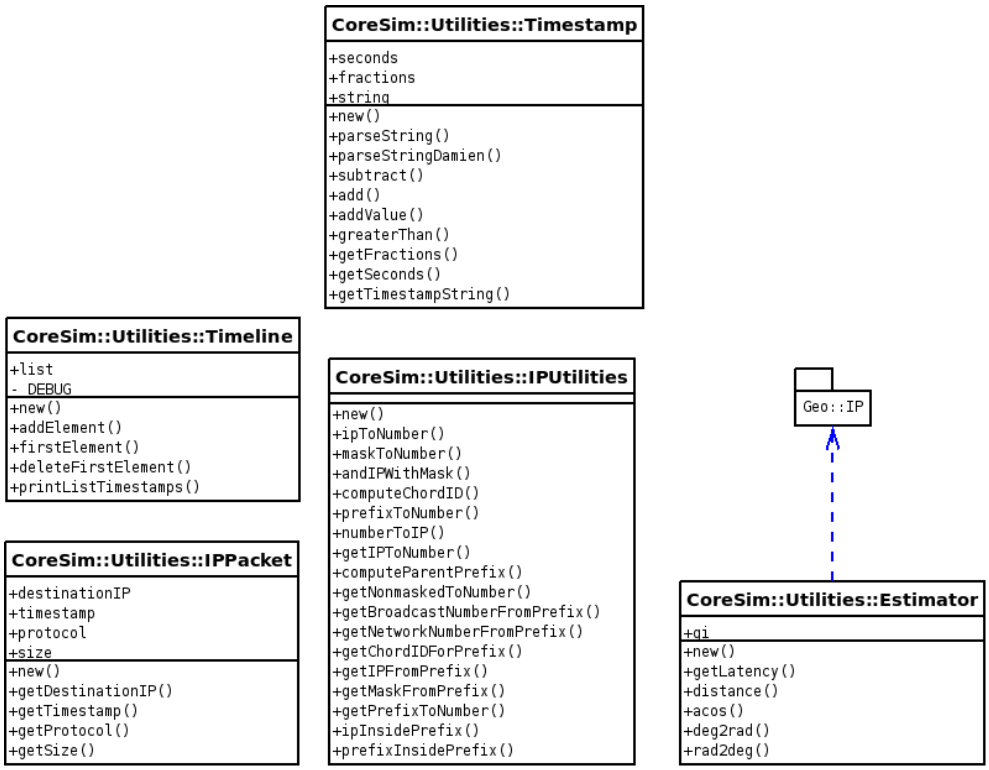

Figure B.6: Utilities Module UML Diagram