FLIP: an Internet protocol for flat labels

In recent years the increasingly frequent use of the Web service, the advent of the cloud computing, the exponential growing of mobile devices with the introduction of pervasive and ubiquitous computing and the emergence of extreme distributed systems have brought to light the problem of the no longer adequate distribution of data packets over Internet and the related IP protocol issues. This paper promotes flat labels as a real alternative to IP addresses for a future Internet architecture and proposes FLIP as first network layer protocol for flat labels. Among several features absolutely not existing in IP protocol, FLIP has a native support for crypto-currencies.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.1749v2 | CC-BY 4.0 Open Access | rec: 23 May 2016, publ: 23 May 2016

FLIP: an Internet protocol for flat labels

Giovanni Bembo

Department of computer engineering, University of Rome "Tor Vergata", Rome, Italy.

ABSTRACT

In recent years the increasingly frequent use of the Web service, the advent of the cloud computing, the exponential growing of mobile devices with the introduction of pervasive and ubiquitous computing and the emergence of extreme distributed systems have brought to light the problem of the no longer adequate distribution of data packets over Internet and the related IP protocol issues. This paper promotes flat labels as a real alternative to IP addresses for a future Internet architecture and proposes FLIP as first network layer protocol for flat labels. Among several features absolutely not existing in IP protocol, FLIP has a native support for crypto-currencies.

Keywords – GVN, Generalized Virtual Network, Flat Labels, DHT, ROFL, Dynamo, VRR, Scalable compact routing, Disco, Routing on identities, Name based, Internet architectures, Network layer, Name system, Crypto-currencies, Micropayments

1. INTRODUCTION

A **full migration** to a new architecture: why not? Of course, IP has demonstrated its reliability during past decade and it's hard to think to a new architecture that can totally substitute it. However, research and projects about that eventuality have followed the idea of a *general purpose* oriented architecture that can be mobile friendly and can allow many other useful features. **Flat labels** seem to be perfect candidates to gain the position of actual Internet addresses because can achieve all *positive features* of IPv4 addresses like *non mutable semantics* or IPv6 addresses like to be more than billion, moreover can allow a **separation of their identity from their location** and address not only hosts or devices (the latter depends on the architecture on which they are utilized). A migration from IP to a flat label based architecture is obviously not immediate; we think that the only way to go to a complete migration is to allow a long coexistence of IP and flat label addresses during which the Internet community can use and evaluate the new proposed architecture without losing anything. We call that "**the gradual migration**". This paper introduces FLIP, first Internet protocol for flat labels, that allows a coexistence of IP addresses and FLIP addresses (a.k.a. flat labels) for the same host so adding the new powerful benefits of flat labels to the *usual Internet*; as practical consequence the IP global infrastructure has no changes while community can evaluate new experiences with flat labels (and eventually think to a migration). A prototype of FLIP has been implemented using the **GVN** (Generalized Virtual Network) [2] API in order to be easily layered on IP before, and on a flat label based architecture after (if there will be a gradual global migration).

Let's take a look at some interesting features of flat labels used in combination with FLIP: first of them is a possible concrete realization of **a vision of a global infrastructure**, as described in [17], where "*Anyone will be able to plug in from any location with any device at any time*" [cit.]. FLIP addresses are flat labels, and can map any entity (that we call: **resource**) is thinkable to be able to own an address, so not only hosts or devices, but also files, disk blocks, memory areas, parts of code, virtual machines, users, groups, and everything is identifiable by an ID. That's possible because flat labels are "virtually infinite" and aren't subjected to a specified physical location or to a particular semantic.

This paper begins with an overview of related works and some scenarios where FLIP could be useful; then in section four it gives some basic concepts of various technologies adopted. Section five talks about FLIP details and specifications both when layered on IP as pre-migration / integration / evaluation step and when adopted as possible network protocol for flat names based architectures in an hypothetical post migration final step. Sixth section regards technical implementations of FLIP on IP while in section seven there is a technical description of how resources and *groups* of the latter are built. Finally, eighth one describes the technical implementation of FLIP on a flat label based architecture. Section nine is about a helpful evaluation of FLIP over IP.

2. RELATED WORKS

IP or not IP? It seems a *Shakespeare-like* question but we think it is not so far from where we are going: there are a lot of projects trying to improve, bypass, extend or substitute the actual distribution capabilities of the IP protocol. Research and projects about that have led to the formation of three main "*schools of thought*". **First school** based his studies on **high performance architectures for content accessing** while **second** focused on research of **oriented or entirely dedicated architectures** to the *general purpose* (that are, architectures not benefitting a feature in spite of another one). First and second school have in common the objective to adopt something that is not in the current Internet, such as new protocols, ad-hoc routers, different name system services or addressing semantics, and so on. **Third school** is different: it includes all projects that use the current IP-based hardware to exploit existing and discretely programmable routing protocols in order to manage packet forwarding and/or update the routes, independently or *user-driven*. For what concerns FLIP protocol, it belongs to second school and incentives using flat labels as solution of many issues; let's see some of its fairly and nearly related projects and discover to what school they belong.

Content Centric Networks and CONET. CCN [3] and CONET [4] belong to first school. They try to improve aspects where IP data distribution has to be "enforced": web contents, shared files, mobile devices. Base of their work is routing on content names, so extending standard IP data distribution capabilities layering a CCN ad-hoc protocols stack over IP.

SDN based projects and OFELIA. Software Defined Networks belong to third school and use almost programmable routing devices on IP based hardware to forward packets and modify routing tables at software level. OFELIA [8] is one of the most important projects in the SDN paradigm; it is based on hardware that supports *Openflow* protocol.

Internames. Internames [6] and MobilityFirst (see below) are two projects almost near to FLIP. Internames is an architectural framework in which names are used to identify all entities involved in communication: content, users, devices, logical points, and services.

MobilityFirst. The goal of MobilityFirst [19] is to better accommodate mobile entities on the Internet in a scalable, trustworthy, and useable manner. The related project takes a radical approach to redesigning the Internet including rethinking end-point naming, such as through IP, and connection-oriented protocols, such as TCP. At a high level, MobilityFirst allows applications to securely interact with abstract, mobile entities in a connectionless fashion, providing connectivity and minimal user-disruption in the presence of mobility.

3. SOME SCENARIOS

According to [17], "*A fundamental way in which nomadic computing differs from conventional desktop operation is the huge variability in connectivity to the rest of the user's computing environment. That level of connectivity often includes extended periods of low bandwidth or no communication capacity at all. Since many users and programs alike make intermittent, but nevertheless essential, use of "off-machine" information and services, they are unable to operate effectively unless extraordinary steps (like reconfiguring their IP address, changing their netmask, removing their proxy, etc.) are taken by sophisticated users or their network administrators. The goal of nomadic computing is precisely to permit users and programs to be as efficient as possible and as unaffected as possible in this environment of uncertain connectivity and unfamiliar locations. That is, nomadicity makes the sometimes-connected computer operate in the same way and as effectively in a foreign location as when it is connected as a friendly to his organization's information network*" [cit.]. With FLIP no one will need to reconfigure his host address (thanks to a flat labels property: separation of location and identity); there will be no more netmasks (if the foreign environment has left IP for flat labels), and so on. Moreover, with FLIP an user can create a group of flat labels and identify it with a FLIP address so that moving from a location to another one will be more simple: all entities identified by flat labels in that group will migrate at the same time, so transforming foreign environment in a familiar one. **FLIP groups** eliminates the distinction between networks and storing systems, threating **the net as a gigantic database**. To gracefully understand FLIP's, or in general, flat labels' potential, let's see some scenarios.

Scenario 1: searching for moving code

As first scenario, this paper introduces a realization of a **rapid service discovery** discussed in [18] using FLIP. Alice is running an app on her *FLIP ready* smartphone (that is, her smartphone implements FLIP protocol other than the IP one). The app allows her to watch online video streaming from some free and FLIP ready storing servers. Videos on servers are in different format because are loaded without transcoding from other uploaders. Alice choose her video from a list and tries to play it in streaming but the app haven't the appropriate codec to do that because the file has a ".cod" extension; **automatically and transparently**, the app begins a key/value search using the peer-to-peer FLIP

subsystem to find nearest FLIP hosts that have published the app FLIP group address, then choosing between them those that have published the ".cod" extension FLIP group address. Finally, the app downloads the proper codec from the latter and Alice can play her video. Let's note that there is no possibility for the codec to come from a fraudulent server because codecs are encrypted symmetrically and the symmetrical key is also randomly generated and asymmetrically encrypted in the codecs with the private key of that group (only the group owner holds it); on the other hand, every FLIP address is a public key so that decrypting is as easy as to know a FLIP group address. This scenario is a classic example of **code mobility** where flat labels play an important role giving an identity to every portion of code (called the "know-how" as described in [16]) in Internet.

Scenario 2: the mobile file system

Bob has a FLIP based filesystem (that is, a filesystem able to address and crypt its blocks with FLIP addresses). Bob's filesystem is distributed in three different Italian hosts and Bob's laptop knows only the three first blocks FLIP addresses; such addresses are published by the three Italian hosts. When Bob's laptop tries an access to that filesystem, it connects to the first blocks FLIP addresses and reads from or writes to them. One day Bob has to go to the USA; in the airport he turns off its laptop and embarks on the plane. Once arrived at his apartment in New York, Bob turns on his laptop that holds its FLIP address (identity/location separation); trying to access to the FLIP filesystem, Bob's laptop suffers of slowness due the distance from Italian hosts so that automatically searches for one or more available American FLIP hosts and then orders a migration to them (an host is available to receive migrated blocks if it belongs to an ad-hoc created group for FLIP filesystem, as in scenario 1 where a server is available to storing a codec if it belongs to the FLIP group of that codec). During the migration Bob can still transparently access not yet migrated blocks from Italian hosts, "feeling a gradual speedup" of read and write operations until the migration is completed.

Scenario 3: security issues

A system administrator of a big USA company has traveled to Europe for work. Having to access to the user interface of one of the company's servers in USA, he need to bypass the company's firewall. Fortunately, both firewall and server are FLIP ready so that there is only a simple rule to add to the firewall: let pass only FLIP packet signed with sysadmin's private key. There is no more port filtering neither deep content inspections or DDos attack risks: only sysadmin can generate traffic with his FLIP address that is different from the FLIP address of the host with which is connected to the Internet from Europe; indeed, the sysadmin can decide to send FLIP packet from his own FLIP address and not from FLIP address of his temporary host.

Scenario 4: a micropayment system

As last scenario this paper leave the yet unique feature of FLIP that supports crypto-currencies natively. Packets in FLIP can "carry" money so that a communication based service (almost all in Internet) can be delivered if data it receives contains a payment. That type of small, and optionally continuous payments, is called micropayment and can incentive the crowd sourcing as well explained in [18]. A typical scenario of micropayment consists in the pay-for-browse service of a site where every FLIP packet that come from that site can request a payment within the ack response packet; but a real form of crowd sourcing is well explained in the following scenario 4. Bob is on the train for Rome and needs to connect to Internet with his smartphone that is only wi-fi capable and there isn't hotspots on the train. Fortunately the smartphone is FLIP ready so he localizes Alice's smartphone (FLIP ready too) that is publicizing the FLIP group "crowd bitcoin connections" (a group of mobile devices that can allow connections to Internet for bitcoins). After a few handshaking, Bob connects to Internet through Alice smartphone (that is 4G and wi-fi capable), and pays her 1 Satoshi (=0.00000001 bitcoins) every received FLIP packet.

4. BASIC CONCEPTS

FLIP is a network communication protocol for flat labels. It can be layered on IP and at the same time be applied to a flat label architecture allowing coexistence of the latter and the IP one. FLIP addresses are flat labels and layering on IP is done using GVN, while the most reliable candidates as new flat label architecture on which to layer FLIP, we think, are ROFL [1] and Disco [13]. Before entering in details, let's briefly explain the concept of "flat label" and speak about GVN and Disco.

4.1 Flat labels. Flat labels are bit strings of variable and finite length. Unlike in IP addresses, their variable size allows an additional flexibility that belongs to the use of hash functions that "normalize" the size of the flat label to one length while retaining its uniqueness. Every FLIP packet has its payload encrypted asymmetrically and symmetrically: first bytes (asymmetrically encrypted) of the payload can contain a temporary symmetrical key to decrypt the rest of the payload itself; that's because it may be a bad idea to encrypt it asymmetrically once you have seen its length. With regard to safety and in order to maintain a level

of the latter almost equal to that provided by the symmetrical one, asymmetrical encryption requires much longer keys sizes (that is, FLIP addresses are very long): in 2003 the RSA stated that a 1024-bit asymmetric key is equivalent to an 80-bit symmetric key; a 2048-bit asymmetric key is equivalent to a 112-bit symmetric key and 3072-bit asymmetric key is equivalent to a 128-bit key. RSA recommends using at least a 1024-bit asymmetric key if you plan to keep your documents safe until 2010 and use a 2048-bit asymmetric key if you want to secure documents until 2030; the 3072-bit asymmetric key is suitable for documents that should remain confident beyond 2030. A NIST document defines an asymmetric key of 15360-bit equivalent to a 256-bit symmetric key. As a result, each FLIP packet could have larger IP packets sizes in order of several kilobytes, since the protocol has three fields that contain flat label: popular 1500-byte limit is exceeded only by the overhead (so without payload) and this could be a problem for networks with relatively low MTU (e.g. Ethernet) to divide the IP packet containing FLIP in multiple frames. The use of **GVN** solves the problem highlighted thanks to its transparent fragmentation feature: everyone can send and receive big FLIP packets without warring about fragmentation. Using FLIP, the network would manage variable sized addresses according to the size of asymmetric key used: the safer must be the FLIP payload, the longer will the address, the more computing capacity and bandwidth will be needed for those who want to take advantage of that. For simplicity in this paper I'll use, where not otherwise specified, to 1024-bit FLIP addresses and the asymmetric encryption algorithm will be RSA. The only limit of flat labels into the FLIP protocol (that utilizes DHTs, Distributed Hashing Tables, see later) is the choice of the length of the resulting hash string and the type of DHT: once you have made these choices, they can no longer be changed.

4.2 Why to layer over GVN instead directly over IP? FLIP uses GVN for its smart fragmentation abilities. We have directly worked in GVN team to implement fast and reliable fragmentation functions in its API so that every layered protocol can easily utilize it, as FLIP does.

4.3 Disco. Disco is a scalable routing protocol. Its authors have achieved what until now was considered an opened issue: a scalable, *low-stretch*, routing on flat names protocol; they define "stretch" as the ratio of the protocol's route length to the shortest path length. Since shortest-path routing is theoretically achievable but practically impossible due to immense memory requirements and even more communication and computation work, Disco limit itself to guarantee a low stretch of 7 (according on its authors evaluations) in worst case on flow's first packet, and 3 on subsequent packets. There are two "ways" of routing in Disco: first is a name-dependent compact routing, second instead is name-independent; that's because Disco is comprised of *NDDisco*, a name-dependent compact routing protocol and it's possible make an initial choice on what of them to use. Even if using a consistent hashing distributed name database that associate node names to flat labels (addresses), name-dependent compact routing may be dangerous for a security issue: a node can arbitrary change data related to hashes of the portion of database that it stores. Name-independent compact routing is what we prefer: Disco can maintain state of routing tables of every node with high probably without using name resolution; obviously, there is a vanishingly small but nonzero probability that destination's route is not found, but in this case authors advise to recur to name resolution as a fallback.

5. DETAILS AND SPECIFICATIONS

To introduce FLIP's details and its specifications, now let's see FLIP protocol applied to two different architectures by using GVN before over IP and then, only theoretically, over ROFL that maybe is the simplest flat label routing architecture on which implement our protocol; using this approach we will explain FLIP step by step and easily. In the end, I'll discuss about FLIP layered over Disco that is a very performant flat labels compact routing and maybe is also the final target of a complete migration from the IP addresses to the flat labels ones, however not excluding a indefinitely coexistence of both.

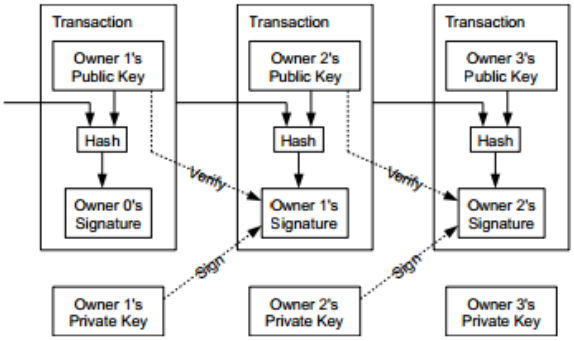


Figure 2: Chain template blocks of a crypto-coin. Every client of crypto-coins has a copy of the latter. A new rectangle is added from every FLIP packet that contains a transaction.

5.1 FLIP over IP: details. Is it a gamble to give the pillars of Internet, the IP addresses, some features remotely attributable to elements that are part of the peer-to-peer architecture? We note that each IP address is "*equal*" to

other addresses; no one exists, either IPv4 or IPv6 address, which has an influence of any kind on the other of the same version. Unfortunately, IP addresses do not provide separation between location and identity, so any device identified with one of them is forced to change its ID every time it travel between geographically distant areas (i.e.: the smartphones). In a complete vision of *pervasive computing* and *ubiquitous computing* not only each device should be able to retain their own identity regardless of its location, to communicate with any other and wherever it is without any intervention, and to provide every service that the rest of the devices offer, but the concept of "reachable host" should embrace more than a single category, the devices one, and should expand itself to user contents or, more generally, **user resources**. Thanks to flat labels, FLIP uses addresses suitable to peer-to-peer based architectures, allows separation of identity and location, offers autonomy and dispenses the needed primitives to allow each device to provide those resources that the rest of the devices provide. Its three main objectives are: 1) **aiding the migration from IPs to flat labels addresses** 2) ensuring a **transparent asymmetrical encryption, always active and not constrained to a single subset of algorithms**; 3) providing an on-demand payment system for data transfer with **every crypto-currency**. The *first goal* is achieved thanks to GVN that will easily support FLIP on IP and will provide its libraries for the destination flat label architecture. On IP, each FLIP node (peer) allows you to associate **a fixed-length alphanumeric address** (flat label, that is also a public key) to each IP address in a dynamic way (associated IP can be changed on demand); different speech you will do for the destination flat label architecture where the dynamic association “FLIP address” <-> “flat label destination address” may be not necessary, that’s because both may be the same. Inside and outside IP, each flat label is managed by a peer-to-peer network implemented by FLIP primitives; the primitives will then assign to each entity an unique alphanumeric public address and a secret (private) key. Device, users, files, other data structures, etc. will possess a pair of public and private keys and packets sent from a public address to another will be **always encrypted** (in first bytes of their payloads) with the secret address (private key) of the sender, ensuring that each packet received should have been definitely sent (because signed) from anything else apart from the sender and thus satisfying *the second target* (there will be no more need for SSL, HTTPS, SSH, etc.). As regards the *third goal*, however, our opinion is that the birth of crypto-currencies was a real revolution for the Internet, perhaps having greater importance than Web 2.0 and social networks; potentials hidden behind crypto-currencies are not only of economic nature and FLIP use one of these: *a low (or network) level* manageability of transactions. With FLIP is possible to **send and receive packets with money**; in practice the protocol so **sent (or received) packets carry money**. FLIP primitives implement the **transfer of a string** which is the result of a series of hashes and digital signatures used in crypto-currency technology; this string represents a payment transaction and its destination will insert it in the last block of the chain strings of the crypto-currency used and propagate it to all the other clients who will in turn add it on top of the last block of their copy of the chain (**Figure 2**), thus providing a computationally difficult not refutability of the payment sent by the sender. In addition to these three main objectives, FLIP offers other important features: as mentioned above each entity (resource) has an alphanumeric address; even **the entity types are definable in an abstract manner** by the user and can be identified and published: the user is not limited to a little group of entity-types such as device, users, files or content, but can if necessary create a resource type *X* with alphanumeric address *K* and another one can find it on the network by identifying it with *K*. An example among thousands ones of the relevance of this potential (the only limit is the imagination) could be the resource type X: *”computing power hired for the program P”* where P is in turn identified as a resource; the user who will run the program P can do a search through the P2P FLIP network *for resource type X* (as done in scenarios 1,2 and 4) finding all nodes that offer (paid or not) computing power for P and choose between the found resources most advantageous ones. For each type of resource created the primitives will create a new P2P DHT (see “Implementation” chapters). FLIP also offers the chance to "*bypass*" the firewalls (at network level, not at HTTP protocol level as [5] does) because active encryption makes the sender always identifiable: a user can send packets signed with his private key instead signing them with the private key of the device that he uses (ubiquitous computing), making his data flow recognizable by firewalls that will enforce or not appropriate policies. **Figure 3** shows FLIP protocol over GVN layered on the version 4 IP protocol. For a correct usage of the protocol what follows is needed: 1) a set of clients (peers), each of which can act as a source, destination, publisher of resources and gateway; 2) a set of gateways that are responsible to communicate between two different network technologies; 3) finally, although optional, the set of name servers: suitably programmed (one among many: FNS [9]), they maintain “FLIP address” <--> “domain name” matches and “FLIP address” <-> “network address” matches and accelerate operations of FLIP gateways. Closing the IP address speech, it is important to note that implementing FLIP does not affects existing routing capabilities.

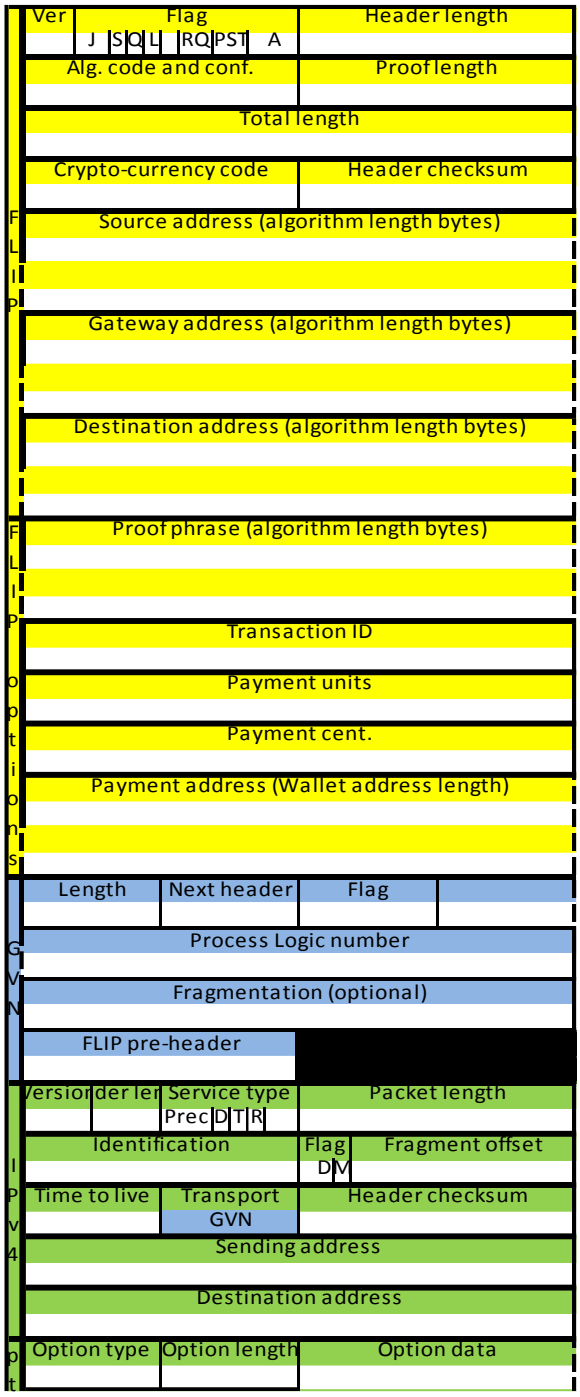


Figure 3: FLIP over GVN (over IP).

5.2 FLIP: some protocol specifications. Now I'll show the description of the most important fields that make up the protocol of course independent from the different packet based communication technologies:

- **Header length:** FLIP header length (in words)
- **Algorithm length:** length of the input string of the chosen asymmetric encryption algorithm (in bytes).
- **Join (J) Flag:** if set to 1 it asks the target node to join the network.
- **Proof Request (Q):** if set to 1 then a proof is required from the destination. For example, such proof can be requested as identity evidence of a node that attempts a Join, or to authenticate a query that will update a record in a name server, or an user to bypass a firewall, etc.
- **Proof Flags Response (S):** if set to 1 it declares that the FLIP packet contains the proof requested.
- **Leave (L) Flag:** if set to1 then the source tells a client or a server name that is going to leave the network.
- **Flag Resource Request (RQ):** this can be set to 4 values:
  - 00: no request
  - 01: a response is required from nodes that are more *close* and from which you can take advantage for the resource that follows the FLIP header (in DATA section)
  - 10: the node is responding as a possible beneficiary of the requested resource in the DATA section
  - 11: reserved for future purposes
- **Payment Session Flag Type (PST):** this can be set to 4 values too:
  - 00: no payment request.
  - 01: time based. Payment will be required at equal time intervals.
  - 10: now. Payment is required now.
  - 11: on demand. Payment will be required several not specified times (by setting this flag to 10 every time it is needed).
- **Algorithm Type Flag (A)** : kind of asymmetric/symmetric cryptographic algorithms used.
- **Crypto-currency code:** crypto-currency code used for transactions (e.g.: 01: Bitcoin, 02: Litecoin, etc.)
- **Gateway address:** 1) the gateway to reach the next gateway to the destination, 2) or the destination itself if no gateway is required.
- **Proof. Phrase:** a random string that is encrypted with the private key of the source and that will be signed by the destination with its private key and then returned to the source in the same field.
- **Payment. Address:** the address of the e-wallet to which send the payment.
- **The payload (DATA Section):** first bytes in this area are encrypted with the private key of the sender (whether it is a device, a user or otherwise). The rest is encrypted symmetrically with the temporary key encrypted in first bytes.

5.3 FLIP over a flat label architecture. FLIP would be the first communication protocol to use flat labels based routing; migration from IP routing to Disco for example would simply carry a next generation Internet in our everyday life: before reaching this final step, let's consider a not efficient but interesting predecessor of Disco in order to

have an exact idea of what is really meaning routing on flat labels: ROFL. Although not being a high-performance architecture according to some studies [10], ROFL is (as Disco is too) able to guarantee a slow migration; however, there are some aspects in ROFL and Disco that should be modified to solve redundancy problems or to improve performance. Access control, for instance, can be managed both by ROFL or Disco routers or at network level by FLIP; a risk/benefit ratio to delete the access control from the architecture (that could not be removed from the protocol because it is needed by the migration from IP) could be therefore evaluated. In terms of performance it is also important to modify ROFL or Disco so that routing can be done on all resource IDs and not only on the IDs of the various devices; to do this, each router should have as many DHTs (or equivalent address management logic items) as the types of existing resources in FLIP giving so rise to what I'd call **multi-dimensional routing** where for each resource type (dimension) there are some defined routes. A practical example is feasible thanks to the difference between user resources and device resources: as already mentioned device IDs are not related to the IDs of the users, and a user can geographically move himself and be accessible via the address of another device (in this paper ID, public address and flat label are synonyms). In the scenario only routes of user resources change because one of the lasts has moved; routes of device resources and their DHTs remain unchanged. Finally, in the context of the problems to be solved about FLIP over both ROFL or Disco, a key issue could be the *temporary mismatch* of elements of a given DHT in FLIP with the corresponding element in ROFL or Disco; in fact, applying the multi-dimensional routing both to FLIP and to the flat label based architecture in case of failure of a device, an element may be temporarily present in one and absent in the other until both will report the failure.

6. IMPLEMENTATION: FLIP ON IP

The *heart* of the FLIP protocol is a DHT of *standard flat labels* which includes hosts and users addresses, named *main DHT*; algorithms for handling it are those of Chord [11] with some editing. The choice fells on Chord because it is the most commonly used and its maximum number of hops, although not optimal, is O(log N), where N is the total number of flat labels; however, it is not difficult to use other *topologies* of DHT. This chapter will explain through intuitive codes written in pseudo-c++ the various ways to take advantage of the potential of FLIP; following five algorithms relate to the management of the *main DHT* on IP.

Algorithm 1. Aggregation of a node *n* connected to the Internet with FLIP address *fl_addr* and IP address *ip_addr* having an already known node n_1. (Pseudo-code)

- n = new Node(hash(*fl_addr*));
- n.ipaddr = *ip_addr;*
- n.flipaddr = *fl_addr;*
- n.join(n_1);
- updateNS(); *//optional*

The **first algorithm** shows the connection of a FLIP entity (a host, an user, etc.) to Chord ring: to be an element of the ring the node must have a key that is the result of a hash function, which in this case is applied at the FLIP node itself. The *join*, *stabilize*, *notify* and *fix_fingers* functions belong to the upgraded version of Chord for the managing of concurrent operations and network errors [11]. The function *updateNS()* updates the records in the name system by connecting to a name server with an already known FLIP address; updating records can publish both only the aggregation of the node, or even the election of it as gateway (see below): all these operations are carried out using asymmetric encryption that guarantees the identity of involved nodes.

Algorithm 2. Pseudo-code of the changes in stabilize() function of Chord.

- x = successor.predecessor;
- if (x ∈ (n, successor))
    - rndstr=n.crypt(*random-string*, n.PRIVATE_KEY)
    - proof = x.crypt(x.decrypt(rndstr, n.fl_addr), x.PRIVATE_KEY);
    - if (*random_string* **is equal to** n.decrypt(proof, x.fl_addr))
        - successor = x;
- successor.notify(n);
- notifyNS(); *//optional*

The *main DHT* is the first element of FLIP to use asymmetric encryption: **algorithm 2** is a modification of the stabilize() function of Chord where a node *n* authenticates another node *x* before setting it as his successor. Authentication is performed with the classic style of asymmetric cryptography i.e. requiring the signature (*proof*) of a string, that is randomly generated (*random_string*) and then encrypted by the authenticating node *n*, from the authenticated node *x* that uses its private key (x.PRIVATE_KEY) to sign it. In this example the two generic functions *crypt()* and *decrypt()* represent the ideal

mechanism for signing and signature verification in asymmetric cryptography. If the signature decrypted with public key (i.e. with the FLIP address) of the node to be authenticated will correspond to the randomly generated string then authentication will success. The last function, *notifyNS()*, asks to a server name with already known FLIP address to authenticate the node x and possibly update the record (a failure may be noticed); this single function delegated to the server name the responsibility to verifying the existence and authenticity of node x, and optionally to change or to add the record; on multiple requests in a short period, the name server may decide, for example, whether to run all these tests or just the first one, or discard each of them.

Algorithm 3. Pseudo-code of the changes to notify() function.

- if (predecessor is nil or $n_1 \in$ (predecessor, n))
    - proof = n_1.crypt(*random-string*, n_1.PRIVATE_KEY),
    - if (*random_string* **is equal to** n.decrypt(proof, n_1.fl_addr))
        - predecessor = n_1;

Algorithm 3 implements the same modification used for the authentication of a node but in this case it is applied to the notify() function of Chord. This modify is expensive in terms of bandwidth overhead and of computing capacity but necessary and sufficient to certify the identity of the main entities of the ring; for greater security, however, you might consider the possibility to apply it also to the fix_fingers() function of Chord with the cost of a further reduction of the above resources. Now, returning to *main DHT* protocol usage I'll do an example written in a pseudo-C++ style code that relates to the authentication request that a node sends to another one; please considered it purely indicative because programming style is personal and combinations of the instructions necessary for the implementation of a function may be manifold.

Pseudo-code 1. Code written in pseudo-c++ of a function that uses the FLIP. This example function shows a node n while creates and sends a FLIP packet to authenticate another but already known node n_1, and then awaiting for reply. The node n has only an IPv4 address (in addition to FLIP address) as well as the node n_1.

- pkt = new **FlipPacket**();
- pkt.**setProofReq**(1);
- pkt.alg_length = l_{alg1} ;
- pkt.alg_type = $alg_{1;}$
- pkt.source = n.flipaddr;
- pkt.dest = n_1.flipaddr;
- pkt.proof = **crypt**(*random_string*, n.PRIVATE_KEY, k_1, x_1);
- if (n_1.ipaddr != NULL)
    - n.**send**(pkt, n_1. flipaddr);
- else
    - gw = n.select_next_gateway(n_1);
    - if (gw == NULL)
        - if (*gateway* != NULL)
            - gw = *gateway*;
        - else
            - return FALSE;
    - n.**send**(pkt, gw);
- rcvpkt = n.**wait_for_proof_res**();
- if ((rcvpkt != NULL) AND (rcvpkt.getProofRes() == 1))
    - str = **decrypt**(rcvpkt.proof, n_1.flipaddr, k_1, x_1);
    - if (str == *random_string*)
        - return TRUE;
    - return FALSE;

Building the packet the bit for the authentication request is set to 1 and the various fields are filled, in particular those describing the chosen algorithm for encryption are filled with the *well known code number* alg_1 (e.g.: 1 = RSA, etc.) and with its length (e.g. 1024 bit) l_{alg1}. If the chosen algorithm for the *main DHT* is different from that presented in this paper and the authentication request was made in order to aggregate a node, then you may need to set the *Join bits* to 1 too. In this function the condition of existence of an IP address always bypasses the call of the function *select_next_gateway()* (described later) simply sending the packet through the *send()* function; since in this example the target node has certainly an IP address, that condition is unhelpful and here is implemented only to introduce the next paragraph about the *send()* and its use of the name system. Then if the *rcvpkt* object contains the response packet (for completeness you might also verify the correspondence between the source and the node n_1) and the response bit for authentication is set to 1 then you can proceed with verification: the string in the *proof* field is properly decoded by the instructions within the ad hoc function *decrypt()* that you can implement as needed. The return value will be TRUE if the node n_1 will be authenticated, FALSE otherwise.

6.1 Dynamo. This is the Amazon's high availability key-value store with incremental scalability, symmetry, decentralization and heterogeneity. Actually, it is used only by the Amazon's internal services, but for my goal in this paper we need two assumption: first is that it is available worldwide. Data is partitioned and replicated using consistent hashing and consistency is facilitated by object

like technique and a decentralized replica synchronization protocol. Second assumption that we need is that Dynamo can be built for the destination flat labels based routing architecture (if it is so functionally over IP, why not over Disco too?).

6.2 Name System. Conceptually the name system that should help the *main DHT* to improve FLIP performance is not different from the actual DNS; we might talk about it as *a DNS extension* because what should be in the new system would not change anything in the present one, while adding more potential features. To simplify the explanation you might reduce the concept of Internet DNS to the management of the following match: domain name <-> IPv4 address, as shown in **table 1**.

Domain name	IPv4 address
www.uniroma2.eu	160.80.1.246

Table 1. A simple domain name <-> IPv4 address match in actual DNS.

The matches needed to an NS redesigned for FLIP would instead be five, as shown in the example **table 2** where you maintain other domain features for each NS record. Of course, new name system may have different semantics.

NS name	FLIP address	Architecture type	Address	Resource or gateway type	Next gateway
www.uniroma2.eu	*FL1*	IPv4	160.80.1.246		
gw1.ipv4v6.net	*FL2*	IPv4	123.123.123.123	IPv4v6	
gw2.ipv4v6.net	*FL3*		2001:0DB8:0000:0000:0000:0000:0000:0001	IPv6v4	
www.ipv6.com	*FL4*		4002:1EF8:0000:0000:0000:0000:0000:0002	IPv6v4	*FL3*
www.flip.net	*FL5*	Disco		DiscoIPv4	*FL7*
gw3.ipv4flip.net	*FL6*		124.124.124.124	IPv4Disco	
gw4.ipv4flip.net	*FL7*			DiscoIPv4	

Table 2. Examples of the new NS record to FLIP; being very long, FLIP addresses are here represented by tags that begin to FL.

Coexistence of different network architectures is made possible by gateways; each peer can begin a gateway if has at least two concurrent connections to different architectures and has updated its status in the name system. In this regard, we would like to notice that the need of a gateway exists only in condition of having different architectures using FLIP: in the perspective of a complete migration to a flat label based routing architecture the name system (if used) would only act as an helper to improve performance since, at the migration end, its records will not contain more data about gateways, the only ones whose specifications aren't recoverable from the peer-to-peer FLIP network but from NS before or while migrating. In **table 2** it is possible to understand the design of the architecture of a *new Internet* with only (but not necessarily) three architectures that could be similar to that shown in the following **figure 4** . It's considerable the fact that in **table 2** there is a simple key-value storing scheme and that's the only thing FLIP need to work: there is no complex query management to achieve and so it's possible to think to the *DNS extension* not as a hierarchical system built on centralized servers in which unique key is a NS name, but as a scalable peer-to-peer key value store like Dynamo where the unique key is a FLIP address. According to that, using Dynamo instead of FNS or similar for example will guarantee that FLIP will not relay on a centralized authority.

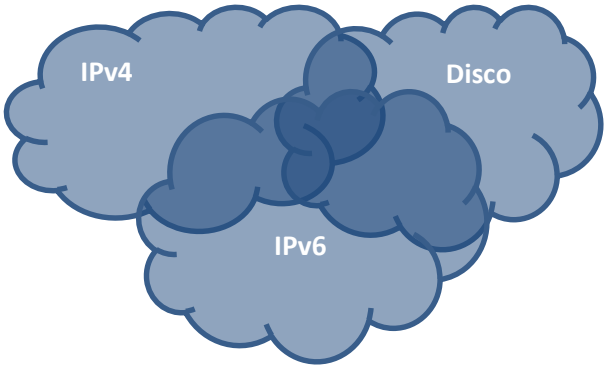


Figure 4. State of migration from IP to Disco for coexisting architectures using FLIP.

Intersections of clouds contain hosts connected to multiple architectures; in that example central intersection shows hosts that can be elected as gateway for all architectures.

Algorithm 4. select_next_gateway() function.

- If (n.ARCH is not qual to n_1.ARCH)
  - For A in (n, ARCH)
    - if A is equal to n_1.ARCH
      - return n;
  - For A in (n, ARCH)
    - gw = nsget_next_gateway(n_1, A);
    - if (gw is not nil)
      - return gw;
  - return nil;
- return n;

If in **pseudo-code 1** node address no_1 had not belonged to the IPv4 class then it would be necessary to select an intermediate node denominated: "gateway" with ability to communicate with both IPv4 and the architecture of the target node. Each node can be elected as gateway or give up

system; therefore nodes can exist with gateway function that are not published. The publication and the election are borne by the node itself which can also ask for money to carry out this service for other nodes (see below). In the **pseudo-code 1** example a particular case is shown too, i.e. one in which the source node architecture finds no gateway to reach the destination node architecture: this occurs when both architectures have no common gateway and must resort to nodes in intermediate architectures. The need for intermediate architectures is rare and usually occurs when one of the two architectures is not widespread or is in a small "neighboring" network, as in a scheme similar to that in **Figure 5** only where a small LAN with IPv6 addresses and an IPv4v6 gateway tries to reach a node with an IPv6 Internet address. To handle situations of intermediate architectures there are two possible solutions: first, that we suggest, is to manually set the best gateways path so that every node in the LAN can reach the destination without travelling through too many intermediate nodes; second would be to make the name system processing a shortest path through various gateways, dramatically increasing the workload of servers. Since these situations as mentioned above may be rare, evaluating the manual selection of the gateways path would be a better choice. On success **algorithm 4** returns the FLIP address of the gateway to insert in the *gateway address* field of the protocol. You can immediately deduce that often, when the packet is distributed among the hosts on a single architecture, the value of the *gateway address* is the same as that of the *destination address*; this redundancy may be deleted with an improvement of the protocol, for example adding a control bit. As first step, select_next_gateway() function checks the existence of the latter case before starting a first control cycle between the architectures that are actually connected to n and so returning still n; the next step involves the execution of a last cycle during which for each architecture *to* which n is linked, a name server will be queried about destination node architectures (here calling a custom function *nsget_next_gateway()* whose implementation is easy and so not described) in order to find a match and get the gateway which will then be returned to the calling function. The worst case is, therefore, when the second cycle fails finding an intermediate host which can act as a gateway: this case is rare and as mentioned above occurs more in configurations like that illustrated in **Figure 5** .

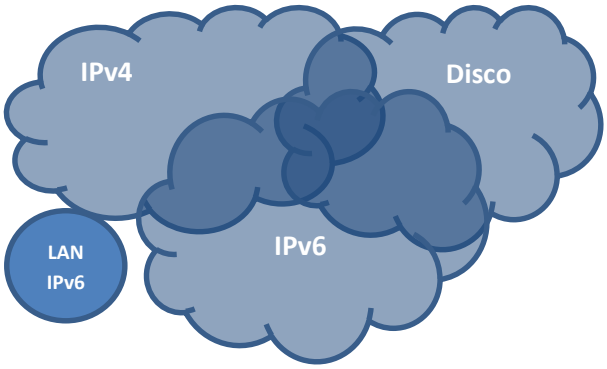


Figure 5. Example of a LAN with IPv6 addresses that tries to contact an IPv6 Internet node but that is only connected to an IPv4 Internet host. In this case, you must manually set the FLIP gateway for the LAN which then will call nsselect_next_gateway() function.

6.3 Crypto-coins and cash flows. Acting as a gateway is a service that could be sold: the money used to purchase it are the ones provided by peer to peer systems of crypto-currencies, first among them Bitcoin [12] created from a.k.a. Satoshi Nakamoto in 2009 and become popular worldwide. Base of this system is also asymmetric encryption and its peers make wide use of it to support a mathematically hard to break and authority independent monetary system. Implementing this system to be used on FLIP is relatively simple: every time you want a payment, for example to allow the continuation of a gateway service or to sell resources (see below), the requesting host adds the necessary data in first ready to be sent packet 1) setting *payment session type* field to a value that depends on the type of required payment, 2) indicating the code of the crypto-currency needed in *crypto-currency code*, 3) setting the amount of money in units and cents in their respective fields *payment units* and *payment cents*, 4) filling the fields *Payment address* with the address of own electronic wallet and *Payment ID* with a value other than zero, then 5) awaiting for the packet carrying money from the host that will (or will not) send them. The latter will in turn decide whether or not to pay and to risk an interruption of the data flow of data or less; in the latter case, the host must make a regular payment and somehow recover the transaction string shown in **Figure 2** (that is not difficult: crypto-currencies software is open source). The retrieved string will be part of the next FLIP packet that is sent to the requesting host using *payment address* field as proof of payment; in addition, *Payment ID* field must contain the same value as that in the packet containing the request citing a reference for the transaction. In the case of fixed-term payments (*payment session = 10*), the same value of *payment ID* filed will be sent through multiple packets at regular intervals along with always different transaction strings without need of further requests by the receiver. Check of the transaction strings is borne by the receiver; at present, in various crypto-coins

systems a transaction takes from a few seconds to several minutes to be confirmed: it follows that the recipient will have to wait a certain amount of time before you receive the money, so the transaction string is more useful as a payment notification rather than as an immediate proof of the same.

7. IMPLEMENTATION: THE RESOURCES

Except for the gateway which as mentioned can be published in name servers, all FLIP resources are first published in peer-to-peer network using DHT and later in the name system in order to improve the performance of the protocol and therefore of the architecture that the latter uses.

7.1 User resources. The most important resource is undoubtedly the user: it owns a FLIP address (and therefore the user owns a private key) and its publication occurs before in peer to peer, and then in name system by the call to a joining function like that in **algorithm 1**. The fact that a user and a host have two FLIP addresses but the same host as destination modifies the search time of Chord algorithm used in this paper: if *u* is the number of users and *h* is the number of hosts in the *main DHT* then finding one of these two resources takes O (log (*h*)) hop (where each hop is a FLIP packet that starting from source arrives to destination) because the hop from a host to the user published by the latter actually does not exist. Until migration to Disco will not be complete, however, in the previous calculation you have to add the number of eventually traversed gateways while searching: in a drastic hypothesis where the maximum number of publishable and intermediate gateways between source and destination is 1 and where all the *h* nodes have another default gateway (for example, because each of them is in a LAN with different architecture connections as in **Figure 5**) needed time *is* O (2 (log (*h*) -1) + log (*h*)) hop where for each hop you must travel through 2 gateways. Implementing a system of calculation of shortest paths in a name system to manage more than one FLIP gateway between source and target is therefore not recommended also to avoid a considerable decline in the FLIP packets distribution performances when looking for a resource that is in a drastic hypothesis already assuming the existence of a single gateway: (2 (log (*h*) -1) + log (*h*)), without considering the packets distribution of the underlying architectures, because the latter still hardly could improve the result. Of course, using the name system instead of using Chord for the search with the maximum number of intermediate and publishable gateways set to 1 and according to the LAN example in **Figure 5,** then the number of nodes used to reach the name server is Θ (2). In case of [illegible] results and name system ones, firsts are to be considered reliable and that explains how FLIP has the feature to be before partially, and in the migration end totally, independent from central authorities.

7.2 Other resources. The *main DHT* is composed of three types of resources: the first two, those of the hosts and users, have already been discussed. The third type is actually a *meta-resource*: its function is to identify another DHT and the host that publics it is its first node and must remain on the net as much as possible, like a beacon for anyone who wanted to use it but don't know where to join. Let p be the node that wants to publish a resource: first of all p creates the resource using the DHT **algorithm 5**, a version that takes cue from the Chord join algorithm [11].

Algorithm 5. newDHT function algorithm that instantiates a DHT for a resource. Node n_1 belongs to *main DHT* as well as node n that creates a new DHT and joins to it, owning the same *resource_flip_addr* addres in both DHTs.

- n = new Node(hash(*resource_flip_addr*));
- n.n = newNode(hash(*resource_flip_addr*));
- n.flipaddr = *resource_flip_addr*;
- n.n.flipaddr = n.flipaddr;
- for i = 1 to n.n.m
    - n.n.finger[i].node = n;
- n.n.predecessor = n;
- n.join(n_1);
- updateNS();

The new resource becomes a node of the *main DHT* and the host that joins it become a reference node for accessing or searching on the new DHT; the type of the new resource is also identifiable through its FLIP address (in **algorithms 5** a resource of type *resource_flip_addr*). All nodes belonging to the new DHT will have their FLIP addresses to be identified: therefore it is impossible for a node in a *resource DHT* not to belong to the *main DHT*, as you can see in **algorithm 6** in which a FLIP node joins a *resource DHT*. After setting various finger of the new DHT, the n node joins the *main DHT* and updates a name server to improve the search performance of the resource type giving to the latter a name associated to its FLIP address in the name system.

Algorithm 6. FLIP node n joins a resource DHT of type *resource_flip_addr*.

- n.n = newNode(hash(n.flipaddr));
- n.n.flipaddr = n.flipaddr;
- n_1 = rsqueryNS(*resource_flip_addr*);
- n.n.join(n_1);
- updateNS(n.flipaddr, resource_flip_addr);

In the previous algorithm n node is a "container" of another node that will be created and attached to *resource_flip_addr* resource DHT. To join, the node queries a name server to locate a FLIP node that already joined the DHT of *resource_flip_addr* type which will allow joining through a normal call to join Chord algorithm. Finally, n updates the name system via an ad hoc function: rsupdateNS adding a resource record that indicates its promotion as node that has joined the *resource_flip_addr* resource type; now it is possible to take advantage from that resource type from the last joined node too. NS records will be stored as well as in the example in **table 3** that completes the **table 2** used for the examples with gateways.

NS name	FLIP address	Architecture type	Address	Resource or gateway type	Next gateway
www.uniroma2.eu	*FL1*	IPv4	160.80.1.246		
gw1.ipv4v6.net	*FL2*	IPv4	123.123.123.12	IPv4v6	
gw2.ipv4v6.net	*FL3*		2001:0DB8:0000:0000:0000:0000:0000:0001	IPv6v4	
www.ipv6.com	*FL4*		4002:1EF8:0000:0000:0000:0000:0000:0002	IPv6v4	*FL3*
www.flip.net	*FL5*	Disco		DiscoIPv4	*FL7*
gw3.ipv4flip.net	*FL6*		124.124.124.12	IPv4Disco	
gw4.ipv4flip.net	*FL7*			DiscoIPv4	
www.fileshare.net	*FL8*	IPv4	125.125.125.12	*FL8*	
	FL9	IPv4	126.126.126.12	*FL8*	

Table 3. A node with FL9 address joins a FL8 type DHT and is added to the example in the previous table.

The new name system thus becomes a resource system for FLIP too; in **Figure 6** you can see a model that describes how the new resource DHTs are distributed within the *main DHT*. The name system offers strong performance increase but the system that uses the FLIP protocol is autonomous and allows searching for a resource type node knowing the FLIP address of the latter, which is actually the address of the publishing node; if there wasn't the name system, that node would be borne by all search requests. By examining in detail the search time without the support of a name system you note a duplication of hops: let *m* be the number of nodes that have the resource; the publishing node will be first contacted as explained in the previous case drastic during (2 (log (*h*) -1) + log (*h*)) hop, where h is the number of hosts on the *main DHT*. Next, the publishing node will search for a node (besides itself) that offers the type of the published resource in O (2 (log (*m*) -1) + log (*m*)) hops because the fingers of the various nodes in the DHT will not refer to nodes in the *main DHT* not providing that resource, and the whole operation will be so ended in (2 (log (*h*) -1) + log (*h*) +2 (log (*m*) -1) + log (*m*))) hops. Even for not standard type resources (not users, neither hosts type resources), in case of inconsistency between the various DHTs nodes and the name system records, only data in the DHT nodes are reliable. We conclude this chapter finally reporting that each resource is accessible only via FLIP protocol and then each data stream that takes advantage of such resource may be subject to payment using crypto-currencies as described above for gateway services; requiring or not requiring a payment is a choice of the node that offers that resource.

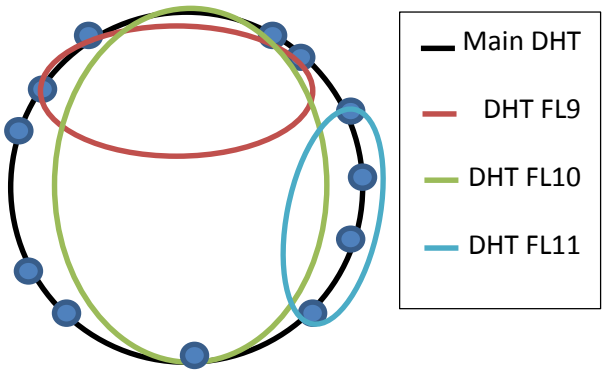


Figure 6. Main DHT nodes joined to other FL9, FL10 and FL11 type DHTs.

8. IMPLEMENTAION: FLIP ON DISCO

Now that I've discussed about FLIP giving an idea of how generally layering it over IP, let's see what we think should be the real destination of IP to flat labels migration: Disco. As about ROFL, we don't know what can be Disco network level protocol, but we don't care about that: maybe it will be FLIP, in the other case we would wait for GVN's authors to use libraries built for it, if tomorrow Disco were a reality. In both cases, important things are two: 1) how to match FLIP host addresses with Disco host addresses? obviously, an easy way is that both addresses of a node are the same and I'll take that as an assumption because both are flat names and Disco does not set limits their length or semantic. 2) How to add what we defined "multi-dimensional routing" in Disco? Since Disco has addresses for internal use only because such addresses are flat labels containing portions of paths and are updated dynamically, we can think to them as IP addresses: **the idea is to associate FLIP addresses to Disco internal addresses**, not to its real addresses (we may

presume that such real addresses could be FLIP addresses and in **table 3,** for example, a Disco real address and a FLIP address are the same), so we can build FLIP's main DHT. **Algorithm 7** is a Disco version of **Algorithm 1**.

Algorithm 7. Aggregation of a node *n* connected to the Internet with FLIP address *fl_addr* and Disco internal address *di_addr* having an already known node n_1. (Pseudo-code)

- n = new Node(hash(*fl_addr*));
- n.diaddr = *di_addr;*
- n.flipaddr = *fl_addr;*
- n.join(n_1);
- updateNS(); *//optional*

A difference from IP is that Disco internal addresses have dynamic labels, so one of them can change its label when already a FLIP node has joined the main DHT; this is not a problem because the primary key of the main DHT is a FLIP address, not a Disco internal address, so the change can be propagated through the peer-to-peer FLIP network as a normal attribute associated with the primary key (that, if using a Disco version of Dynamo, will be very fast). On this way Disco name resolution needed by Disco is done at "FLIP level" instead of at "Disco level", aided by the faster Dynamo. Of course, another choice is to leave to *NDDisco* this work removing second line from **Algorithm 7** so that *NDDisco* can update its consistent hashing database, but that maybe will be slower. In both cases however, all other algorithms shown for IP are the same for Disco, maybe with little and easy to discover differences. That's all.

9. EVALUATION OVER IP

We implemented our FLIP by using the OMNET++ [20]: it is a general-purpose modeling framework that is currently heavily used in the simulation of networked systems and distributed algorithms by the academic research. It is characterized by a modular and extensible architecture of C++ modules, with a Tcl graphical interface, recently ex- tended by providing an Eclipse-enhanced IDE. The simulator comes with several analysis tools that allows the user to study the statistical features of the performed experiments. Due to its success, there are several third-party models that featured academic projects made available through the web page of the simulator and users can import in their models. Specifically, in this work we have used the following libraries for OMNET++:

1. INET framework has provided the means to model the networking devices as routers and switches, however, we decided to do not specify the networking failures as parameters of such models. The issue is that the path-by-path characterization is not possible since the on-field measurements of loss patterns are only possible on end-to-end basis (we can detect that a given subscriber lost a message, but not along which path such loss happened). Therefore we have simplified the topology by having all the underlying routing architecture abstracted by few routing devices.

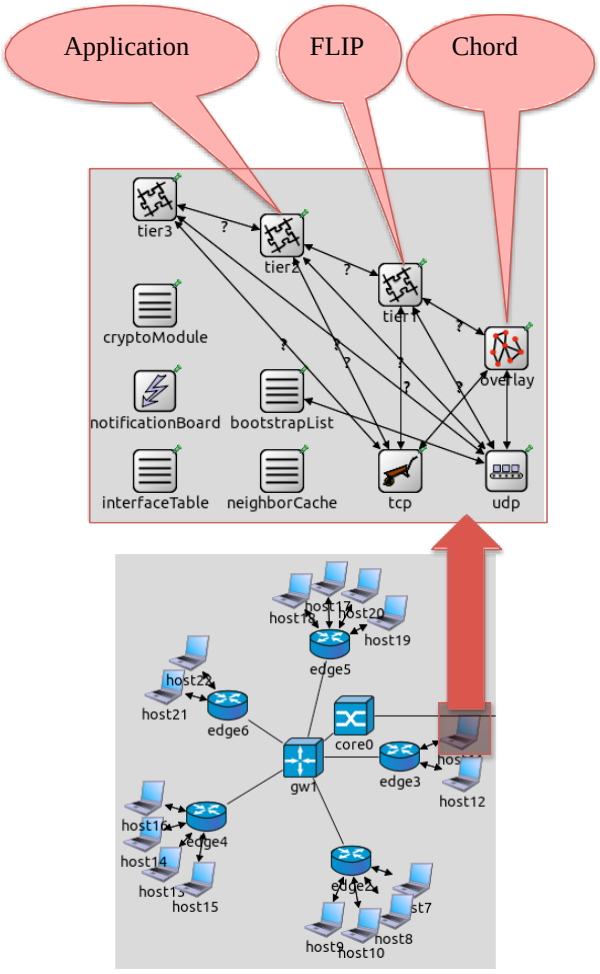


Figure 7. FLIP implementation on top of the Chord module provided by OverSim within the OMNET++ simulator.

2. These has been used to generate the interconnection topology among our abstract routing devices and the end hosts. The correct model of the Internet topology, i.e., the connectivity graph among its nodes [21], is one of the key aspect to address when aiming at making realistic simulations of systems that use Internet to convey information. Internet is structures as an inter- connection of distinct routing domains, also known in literature as Autonomous Systems (AS), which adopts an interior gateway protocol to internally route packets while uses an exterior gateway protocol to forward packets towards others ASes [22]. Therefore, the topology of Internet can be characterized using two distinct abstraction levels: Inter-AS topology, also called AS-level topology, having nodes representing a single AS and edges being the BGP peering, and Intra- AS topology, also called Router-level topology, having nodes representing end-hosts or hardware devices, while edges being physical connections among them. In this work we have limited to model Intra-AS topology, as shown in **Figure 7**, leaving for future work more complex two-level topologies. The network behavior has 50ms as link delay, and the network encompasses 500 hosts.

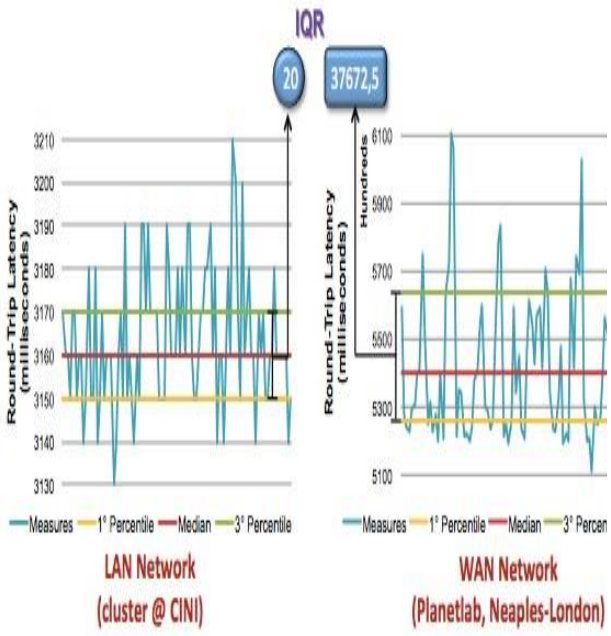


Figure 8. Comparison of Latencies on a LAN and on Planetlab, taken from [23]

3. Oversim [24] has been used to incorporate Chord simulation modules within our simulations and to build FLIP on top of it. Oversim adopts a modular design and that use of the Common API to facilitate the ex- tension with new features or protocols. Specifically, the library provide a skeleton simulation organized in tiers: tier 0 is made of the routing architecture, taken from the INET framework; tier 1 can host any possible overlay, and we have selected Chord, while the other tiers are applications and protocols built on top of the lower tier, in particular we have implemented FLIP on tier 2 and an application on tier 3. This is illustrated in the magnification of a node in **Figure 7**.

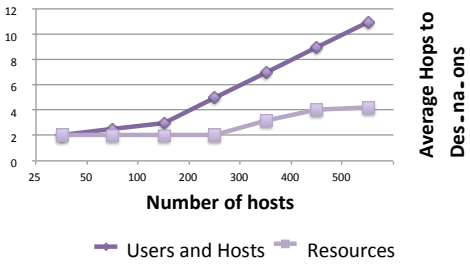


Figure 8. Number of hops

We decided to not use any real wide-area networks, such as PlanetLab[25], due to the uncontrollable loss patterns that make the obtained results non reproducible. For a concrete example, let us consider the study in [23], where some comparisons between a cluster of computers interconnected by a dedicated LAN, and some nodes in Planetlab. On the nodes an exchange of ping pong messages is made through a communication protocol, and the latency needed by a message to go to the destination and backward was measured, and **Figure 8** shows the registered latencies: measures on LAN exhibit lower variability (as demonstrated by an Interquartile Distance of 20), while on Planetlab we have higher fluctuations (as demonstrated by an Interquartile Distance of 37672.5). This allows us to say that Planet- lab is not a controllable testbed, so that it is tough to understand if a variation within the obtained measures is due to some unexpected behavior of the protocol or to uncontrollable phenomena within the testbed. Our simulations has been run ten times, and the average over these runs has been considered in the following discussion (we did not observe standard deviation above 5% of reported values having most of our measures of merit within the 95% confidence interval, thus they are not plot- ted on the curves). The payload of the packets is as specified in FLIP specification [2], by encrypting the messages as above described. In order to quantify the costs of encryption we have measured the overhead to encrypt the FLIP packets by using the Java open source library of the Legion of the Bouncy Castle1 . Specifically, we used the symmetric AES and asymmetric RSA cryptography schemes. We obtained that the mean overhead is 268.96ms, while the standard deviation is 183.39ms.

The workload of our experiments consists of joining and then leaving the overlay built by the FLIP protocol, and registering and unregistering a number of resources during the joining period. More complex application scenarios have been determined, but we leave their assessment as future work. The measure of merit that we have considered in our experiments are the following ones:

1. Needed time and number of packets to join and leave the FLIP overlay;
2. Mean number of hops to reach a destination;
3. Needed time and number of needed packets to find a resource;
4. Needed time and number of needed packets to publish a resource and a type of resource.

Charts from **Figure 9(a) to 9(c)** illustrate the performance of FLIP to add and remove a node within the main DHT. Such value are higher than the traditional Chord overlay due to the authentication needed when the node state has to be varied. We can notice that the joining operation is more expensive of the leave one with respect of all the measures of merit, and all the trends increases as the number of hosts augments. Publishing a resource has a needed time that is lower than joining the main DHT, but higher than leaving it, as depicted in **Figure 10(a)**. This is due to the fact that the required authentication steps are slightly lower, despite the cost to pay for establishing the resource DHT. In fact, the number of needed packets is closer to the ones for joining rather than leaving, while the average time has an opposite trend, as shown in **Figure 10(b)**. Also, the standard deviation reflects such a consideration, as evident in **Figure 10(c)**, since is quite small, meaning that very few verifications, and for close nodes, have been done.

The time for finding a resource is very small compared to the other operations, and also in this case the reason is the absence of any verification means, and the latency, both in the average shown in **Figure 11(a)** and the standard deviation in **Figure 11(b)**, only depends on the performances of the Chord DHT and our approach of the multiple DHT hosts the resources of interest. Enlarging the horizontal scale of the system, i.e., the number of hosts, increases all the measures of merit, but the incremental factor is smaller than in the other cases, thanks to the partitioning of the lookup exploited by our multiple DHTs. The benefit effect of the multiple DHT can be noticed also by observing the number of hops needed to find a given entity, which is lower for resources, thanks to the fact that multiple DHTs have been established for improving such a search.

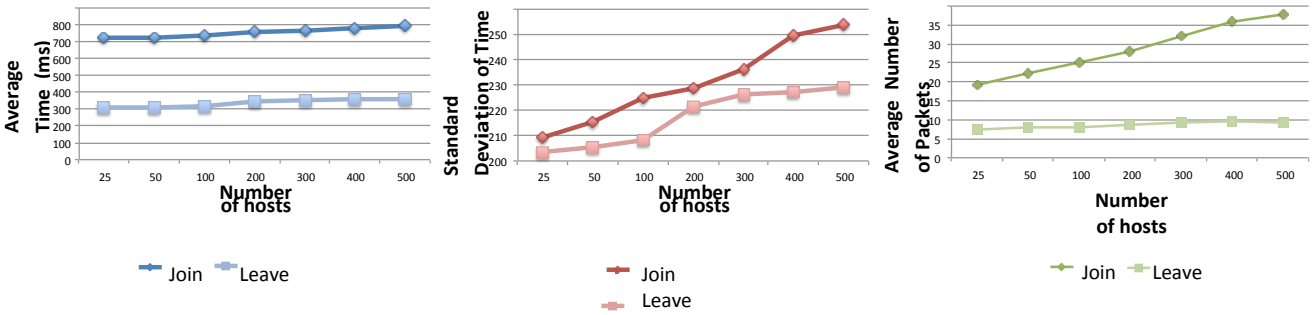


Figure 9. Experiments Results to study the time and overhead of the join and leave operations.

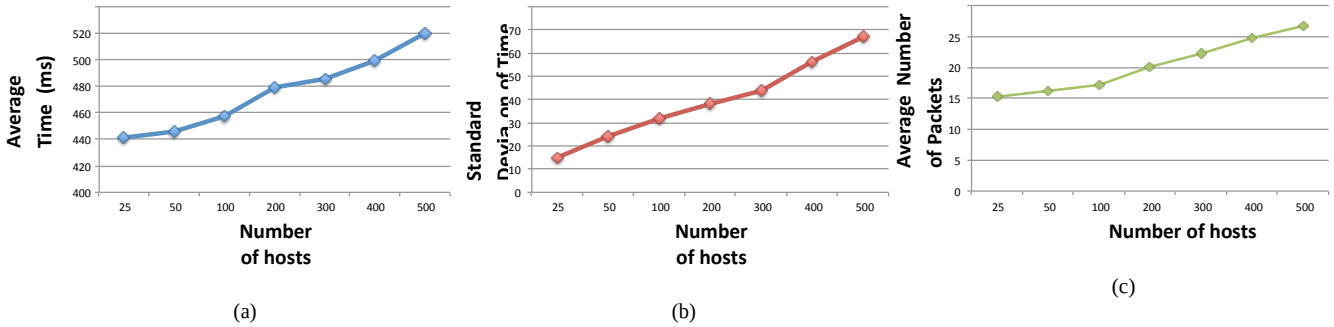
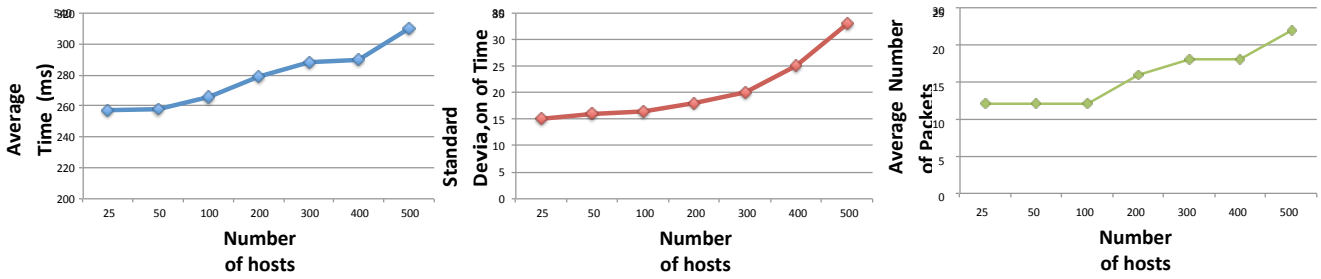


Figure 10. Experiments Results to study the time and overhead of publishing a resource.



(a) (b) (c)

Figure 11. Experiments Results to study the time and finding overhead of a resource.

10.CONCLUSIONS

The purpose of this paper is to point to a reflection on the questions: “is there a way other than the IP one?” and “can flat labels be a simple answer to cloud computing, code mobility, crowd sourcing and ubiquitous or pervasive computing?”. Protocol here described is intended as a first point from where to start a big and concrete work on flat labels architectures, and not as a promotion of a standard for an improbable new architecture. Having said that we conclude with the hope that the idea of FLIP will not be a flop (computer humor).

BIBLIOGRAPHY

[1] K. Lakshminarayanan, I. Stoica, S. Shenker, “ROFL: Routing on Flat Labels”. In SIGCOMM, 2006.

[2] S. Salsano, “Generalized Virtual Networking”. In Networking Group, department of electronic engineering, University of Rome “Tor Vergata”, 2014.

[3] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, R. Braynard, “Networking Named Content”. In Proc. of ACM CoNEXT, 2009.

[4] A. Detti, N. Blefari-Melazzi, S. Salsano, M. Pomposini, “CONET: A Content Centric Inter-Networking Architecture”. Departement of Electronic Engineering, University of Rome “Tor Vergata”. In Proc. of ACM SIGCOMM, 2011.

[5] L. Popa, A.Ghodsi, I. Stoica, “HTTP as the Narrow Waist of the Future Internet”. U.C. Berkeley. SIGCOMM, 2010.

[6] N. Blefari-Melazzi, A. Detti, M. Arumaithurai, K. Ramakrishnan, “Internames: a name-to-name principle for the future Internet”. University of Rome “Tor Vergata”, 2013.

[7] A. Montresor, “Designing extreme distributed systems: challenges and opportunities”. In CompArch, 2012.

[8] OFELIA: http://www.fp7-ofelia.eu/

[9] G. Bembo, “Free Name System” – Tesi di laurea. Università degli studi di Salerno, 2009. http://www.freenamesystem.it/

[10] B. Chun, S. Ratnasamy, E. Kohler, “NetComplex: A Complexity Metric for Networked System Designs”. In Proceedings of USENIX Networked Systems Design and Implementation (NSDI), 2008.

[11] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”. In ACM SIGCOMM Computer Communication Review 31 (4): 149.

[12] Bitcoin: https://bitcoin.org, Satoshi Nakamoto (pseudonimo), 2009.

[13] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy, "Scalable routing on flat names," in Proceedings of the 6th International Conference, ser. Co-NEXT '10. New York, NY, USA: ACM, 2010.

[14] M. Thorup, U. Zwick, “Compact routing schemes”. In proc. SPAA, 2001.

[15] Giuseppe DeCandia , Deniz Hastorun , Madan Jampani , Gunavardhan Kakulapati , Avinash Lakshman , Alex Pilchin , Swaminathan Sivasubramanian , Peter Vosshall , Werner Vogels, “Dynamo: amazon's highly available key-value store”. Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, October 14-17, 2007, Stevenson, Washington, USA.

[16] Antonio Carzaniga, Gian Pietro Picco, Giovanni Vigna, "Is Code Still Moving Around? Looking Back at a Decade of Code Mobility", ICSECOMPANION, 2007, International Conference on Software Engineering Companion, International Conference on Software Engineering Companion 2007, pp. 9-20, doi:10.1109/ICSECOMPANION.2007.44

[17] Leonard Kleinrock, “An Internet vision: the invisible global infrastructure”, Computer Science Department, UCLA, Los Angeles, CA 90024, USA.

[18] Satyanarayanan Mahadev, “Mobile computing: the next decade”. SIGMOBILE Mob Comput Commun Rev 15:2–10 (2011).

[19] Dipankar Raychaudhuri, Kiran Nagaraja, Arun Venkataramani MobilityFirst: a robust and trustworthy mobility-centric architecture for the future Internet ACM Mobile Comput. Commun. Rev., 16 (3) (2012).

[20] A. Varga and R. Hornig. An Overview of the OMNeT++ Simulation Environment. Proceedings of the 1st Inter- national Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, March 2008.

[21] S. Zhou. Characterizing and Modeling the Internet Topology: the Rich-club Phenomenon and the PFP Model. BT Technology Journal, 24(8):108–115, July 2006.

[22] J. Hawkinson and T. Bates. Guidelines for Creation, Selection, and Registration of an Autonomous System (AS). RFC 1930 (Best Current Practice), March 1996.

[23] C. Esposito. Data Distribution Service (DDS) Limitations for Data Dissemination w.r.t. Large-scale Complex Critical Infrastructures (LCCI). Mobilab Technical Report (www.mobilab.unina.it), March 2011.

[24] I. Baumgart, B. Heep, and S. Krause. OverSim: A Scal- able and Flexible Overlay Framework for

Simulation and Real Network Applications. Proceedings of the 9th Inter- national Conference on Peer-to-Peer Computing (IEEE P2P 09), pages 87–88, September 2009

[25] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the In- ternet. ACM SIGCOMM Computer Communication Review, 33(1):59–64, January 2003.