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.1749v1 | CC-BY 4.0 Open Access | rec: 16 Feb 2016, publ: 16 Feb 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

1. INTRODUCTION

<< No one knows the future, then you have to be ready for anything >>. With this sentence I would explain in two lines my membership at the school of thought favourable to general purpose architectures. My theory is that a future Internet architecture must preserve the attributes of independence and decentralization and lend itself to any kind of utilization that will be universally accepted as the current one; only after you have ensured that, it will be possible to improve performance, increase security, add new features and do anything else that might help and/or be innovative. Second school of thought is the one that comes closest to my theory on a global network infrastructure in a first time universally accepted and only in a second time performing, secure and innovative. FLIP (Flat Labels Internet Protocol), ROFL, Disco [13] and similar ones catch almost fully its axioms but before writing about them it's better to introduce the two schools of thought, some of the major researches and related projects, and GVN [2] as possible "vehicle" for an easy large scale experimentation of many of these studies.

1.1 The first school of thought looks for points where the actual Internet Network should be strengthened because most commonly used (Web content files sharing through P2P, mobile devices, etc.), and offer high performance architectures specifically for the above steps. Among the existing proposals, in this paper I cite CNN and CONET, both adopting division between *identity* and *location*. **CCN** (Content Centric Network [3]) is about content names routing and is very performant in terms of contents access speed. Content security also is remarkable even considering that CCN is actually a stack of protocols and security is postponed to a different layer from the *network layer* (the only one having to be universally acknowledged), called *security layer*; the strategy layer instead deals with the optimization of connections based on changes of some conditions (very useful for mobile devices). CCN stack can be layered over any other layer, including IP; its modus operandi is relatively simple: a request is forwarded from one node (Interest packet) through routers and you have an answer (Data packet, the requested content) that during the return path is stored in very large caches by routers as they consider beneficial to do so. CCN is a complex high-performance architecture for findability of contents but requires huge hardware resources considering the size of the cache requested from each router; some ISPs may not endure so cheaply expensive investment and that can make the "new Internet" *not for everyone*. To work around this problem, **CONET** [4] introduces a template named "Lookup and cache" that allows the various cache sizes of routers to be reasonably reduced by splitting the entire domain architecture and by providing a centralized cache update engine for each domain, exclusive to the same routers. There are some possibilities to implement Lookup and cache such as making it depending on a naming service or building it on the basis of a protocol similar to BGP or OSPF; the first of these two examples of implementation seems to be the most popular and, if so, in the future the CONET would become an architecture depending on the authority of a service. In relation to the above described condition whereby **a universally accepted**

worldwide network should be independent of any centralized authority the question of success or failure of a proposal as stated above does not depend on the performance or the number of added features: in this specific case, if a naming system will act as engine for the Lookup and cache then you might meet several political problems as for example: 1) a domain with its centralized engine might prevent or restrict any communication from or to another domain for political reasons or for a choice of single administrator of the engine; 2) the authority that would manage the naming system, even if internationally, might impose payments as already happened for the DNS; 3) you could meet the implementation of other parallel naming systems as engine in certain domains splitting up so the Internet in several "non-interconnected networks". Still looking at CCN and CONET, it's to note that being mainly directed towards a name based architecture, they provide for the names mutable semantics, and this could be a problem for a global agreement, for example, on which characters admit and which are not in namespace syntax (and this is a problem for a routing that depends in part or totally on names, while it is not so for simple name resolution). In addition, both proposed architectures do not have a well-defined transition from the old to the new architecture, and that could be an obstacle to the effective globalization of each since it will not be very clear to you whether investing on a new architecture will give up the old but pretty well working one. Collecting the peculiarity in common to all proposals in the first school of thought I might add that most of them are not designed for point-to-point communications, and the latter, though still implementable, would be very inefficient. Voluntary exclusion of point to point communication precludes the possibility of architecture to be general purpose; both CCN and CONET for example do not allow not publishing nodes to be accessed from other nodes to start any data streaming. A acceptable solution in my opinion and belonging to this school of thought is that you should not change the current architecture (based on IP routing) but concentrate instead on improving the HTTP protocol, in such a way as not to affect the true soul of the Internet and at the same time as to approach a network based on content; the idea of improving the Web protocol (called "Narrow waist of the future Internet" [5]) is probably a good compromise for those who see in the ICN (Information Centric Networking) the right answer but still want the characteristic of general purpose, not introducing a new architecture but by improving the existing one. **Table 1** shows a summary of the new architectures I've just quoted, related (in order): 1) to the existing three fundamental characteristics, 2) to the new fourth one, 3) and to other non-critical ones.

	CCN	CONET
Decentralized	☺Yes	😐Maybe: it depends on the engine implementation (centralized in each domain?)
Does not relay on centralized authority	☺Yes	😐Maybe: it depends on the implementation (NS, BGP, etc.)
General purpose	☹No: 1) Not publishing nodes aren't contactable to start a data streaming; 2) there aren't other service except the content accessing one	
Gradual migration	☹No	☹No
Secured	☺Yes: using security layer	☺Yes
Pro mobile	☺Yes: using strategy layer	☺Yes
Normal amount of resource requested	☹No: router caches must be very large	☺Yes
Not mutable address semantic	☹No if it is name based	☹No if it is name based
Native crypto-currencies support	☹No	☹No
Communication between network with different technologies	☹No	☹No

Table 1: Comparison between some proposed architecture of the first school of thought.

1.2 The second school of thought instead incorporates architectures designed for *general purpose* without privileging in any way their contents. To speak about it I've chosed ROFL and Internames, two elements in the collection that I think are the best. Internames [6] is the architectural model that, as we'll see, more approaches FLIP: in that 1) each name is used to identify all identities involved in a communication (e.g. content, users, both physical and logical devices, etc.); 2) the identities are separate from the location being so really

mobile friendly; 3) there is a communication between different network technologies; 4) it is provided for a gradual migration from current IP-based technology. A perhaps negative note, always according to my theory, is to be reported for the necessity of a powerful name service called NRS in Internames. The dependence on the authority that manages that service, for some of the reasons already expressed in the case of CONET with its centralized engine, may assign this architecture a potentially approval rating equal to zero for some users. FLIP instead, whether layered on IP or on a flat label based protocol (ROFL, VRR [14], Disco, etc.), is only optionally supported by an ad hoc name service (see next paragraph) and that support does not affect packet routing or the full efficiency of one (i.e.: IPv4) or another (i.e.: ROFL) underlying architecture: the name service is only a second and faster choice in *solving answers* already available in first but slower choice using P2P network communication primitives that FLIP could implement. In addition, unlike Internames FLIP is not name based, but uses flat labels with unchanging semantics acting as addresses for its resources. ROFL and Disco do routing on flat labels too: ROFL suffers of the relatively large cache problem in routers, but the average size needed (host identity cache) is not comparable, because asymptotically lower *(O (n) where n = "host" expected value)*, to the one necessary to CCN *(O (n * m) where m = "expected value of content in a host")*; Disco, instead, with its compact routing has very smaller states (authors of [13] analyze a state in terms of the number of entries in the protocol's routing tables [13]). FLIP may instead suffer of excessive size of the cache for its routing based on resources (flat labels as not only hosts addresses like in ROFL) and its expected value would be asymptotically greater than CCN: *O (n * m * (R-m))*, where R is the expected value of resources that include contents, published services, the amount of available hardware, and everything else is definable as resource for an user; this is why FLIP is optionally supported by a name service that can attach to any flat label any kind of metadata (including useful information regarding routes). An alternative or complemental solution to a name system is a smart partitioning system to store *R* between FLIP nodes, like Dynamo (see next) to provide an high availability of such data. Returning to ROFL, the only negative note is that it does **not support a gradual migration from the current Internet architecture**, which, in this case, might not provide any liking by some users. **Table 2** summarizes how the just described architectures of the second school of thought "approach" to the *true soul of the Internet*; as you can see only **FLIP layered over a flat names based protocol like ROFL** (or layered over IP) **possesses all the requirements to be universally accepted** among those cited, because 1) decentralized, 2) independent from authority, 3) designed for general purposes and 4) prepared for a gradual migration (the latter is the only characteristic that such flat label based protocols are missing and that maybe makes them only a fantastic utopia).

	Internames	**ROFL, Disco**	**FLIP (in GVN) over IPv4 using Dynamo**	**FLIP on ROFL, Disco**
Decentralized	☺Yes	☺Yes	☺Yes	☺Yes
Does not relay on centralized authority	☹No: it depends on NRS authority	☺Yes	☺Yes	☺Yes
General purpose	☺Yes	☺Yes	☺Yes	☺Yes
Gradual migration	☺Yes	☹No	☺Yes	☺Yes
Secured	☺Yes	☺Yes, using asymmetric cryptography (a flat label is a public key too)		
Pro mobile	☺Yes	☺Yes	☹No	☺Yes
Normal amount of resource requested	☺Yes	☺Yes	☺Yes	☹No, if not supported by a name service
Not mutable address semantic	☹No	☺Yes	☺Yes	☺Yes
Native crypto-currencies support	☹No	☹No	☺Yes	☺Yes
Communication between network with	☺Yes	☹No	☺Yes	☺Yes

	Internames	ROFL, Disco	FLIP (in GVN) over IPv4 using Dynamo	FLIP on ROFL, Disco
different technologies				

Table 2: Comparison between some proposed architectures of the second school of thought.

2. OVERVIEW

As I said, I am convinced that a flat labels based protocol is the right way in order to start and end a **complete redesign of the architecture of Internet** with the consensus and the interests of all its users, without exclusions. The difference, however, between a beautiful utopia and a new reality in this case is binary and Internet could be *moved from 0 to 1* by implementing gradually the FLIP protocol before on IPv4, IPv6, and then on flat labels architectures. Now let's see FLIP Protocol applied to two different architectures using GVN before over IP (as a 3.5 layer level) and then, only theoretically, over ROFL. In the end, I'll discuss about FLIP and GVN layered over Disco, that is a very performant flat labels compact routing and maybe the final target of a complete migration from IPs addresses to flat labels ones.

2.1 A model of Generalized Virtual Network. The IP protocol is not designed to allow the layering of the protocols and services, so successful applications have approached the overlaying to overcome this limitation (e.g. Skype, CDNs, P2P file sharing system, etc.). The GVN began as *an institutionalization of cross layer and overlay networking* to facilitate a simple and efficient way to find the necessary information in the packet header, instead of conducting a thorough inspection of the protocol; the interesting thing is that it is completely independent from the under laying protocol (i.e.: IP) and from all previous lower level ones proving in the occasion a potential layer 2.5 level, good for a transition from the old Internet architecture to any new one. With these features, GVN allows the coexistence of different architectures (or, more in general, process logics) by assigning a *process logic number* (p.l.n.) to each of them: a logic or logical process, most suitably programmed as kernel modules and present in each GVN capable node, assures the node can handle the architecture (also experimental) chosen by the user; each GVN node can upload logic modules chosen by the owner of the node that manage data flows of new protocols without having to problems with the underlying architecture, such as IPv4 and IPv6. The best feature according to my opinion is that not GVN aware nodes simply manage the packet with the GVN header like other packets of its architectures (i.e.: IPv4 packets) making possible that a GVN packet can pass through not GVN aware nodes to reach a GVN aware one.

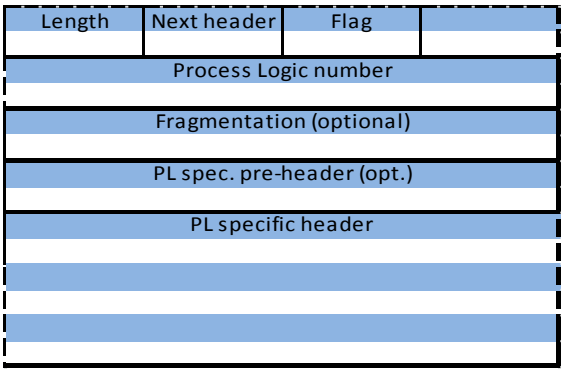

Figure 1: the header of Generalized Virtual Network

Besides the fact that all logics can be programmed freely, GVN also provides a library of functions to interact with GVN data streaming without having to worry about place them in layer 3 or lower protocols; FLIP uses this library that one of the keys of its gradual migration: there will be no work to do to pass to a new architecture (ROFL, Disco, etc.) that will provide the same GVN library. About process logic, FLIP uses p.l.n. 2006 with two specific process logic pre-headers to divide its inter-architectural GVN incoming packets (i.e.: from IPv4 to ROFL) from intra-architectural ones (i.e.: from IPv4 to IPv4): the firsts will be moved to a different network device of the receiving node (that in this case act as a gateway between two different architectures, see below) while the seconds will be managed because arrived to the effective end-node. **Figure 1** shows a GVN header. The fragmentation field has been a my co-working experience with GVN team because while developing FLIP primitives I noticed that a real FLIP packet was often greater than common 1500 bytes MTU of Internet: needing fragmentation, I helped the team to develop it on GVN instead. The fragmentation field now let FLIP and whatever other protocol to have packets length great enough. Thanks to GVN each FLIP node won't have to worry about the architecture on which "will travel" its

protocol because it will be totally transparent.

2.2 FLIP over GVN routing on IP. 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 the flat labels, FLIP uses addresses suitable to peer-to-peer based architectures, allows the 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 since fourth or equivalent OSI level 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 numeric 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 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** 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, it is my opinion that the birth of crypto-currencies was a real revolution for the Internet, perhaps having greater importance than Web 2.0 and social networks; the 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 after having sent or received money**; it is a payment that can be made at protocol level and this is to say that: **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.

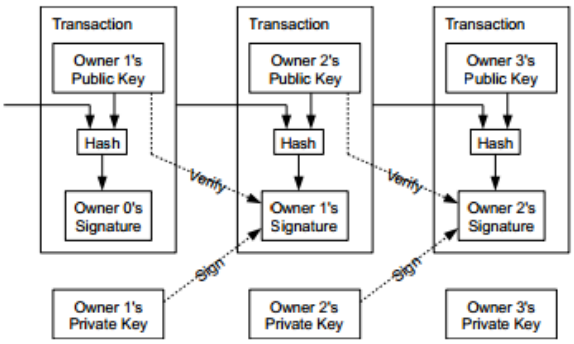

Figure 2: Chain template blocks of a crypto-coin. Every client of crypto-coins has a copy. On FLIP viaggerebbe the last rectangle "transaction" for each new payment.

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*, or between 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 (in case of DHT based primitives). 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.

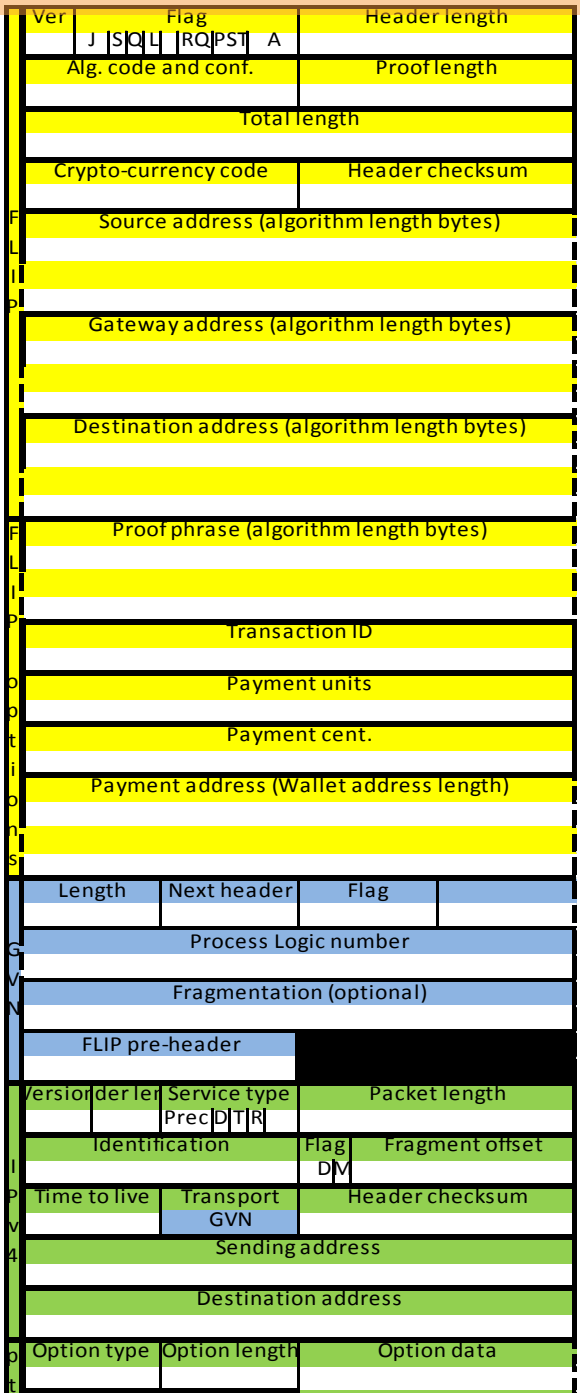

Figure 3: FLIP over GVN (over IP).

Figure 3 shows FLIP protocol over GVN layered on the version 4 IP protocol. For a correct usage of the protocol following 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 existing routing functions are not affected in any way prior to migrating to destination flat label architecture. 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 cryptographic algorithm 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):** all data in this area are encrypted with the private key of the sender (whether it is a device, a user or otherwise).

2.3 FLIP over GVN routing on flat labels. 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 VRR and 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 (as Disco does too) preserves all three fundamental characteristics described in the previous paragraph and with FLIP is able to guarantee fourth (slow migration) still required; 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 DHT 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.

3. FLAT LABELS

Flat labels are bit strings of variable and finite length. Unlike in IP addresses, their variable size allows an additional flexibility for DHT-based architecture or protocol that uses them (e.g.: ROFL, VRR); this flexibility belongs to the use of hash functions that "normalize" the size of the flat label to one length while retaining the uniqueness, thus making them usable in DHTs. 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: 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. From the perspective of a complete migration to Disco anyway, the size of a new version of the protocol should be reduced because it will no longer require the next gateway field. For simplicity in this paper I will use, where not otherwise specified, to 1024-bit FLIP addresses and the asymmetric encryption algorithm will be RSA. The only limit to the FLIP protocol based on DHTs with asymmetric cryptography is the choice of the length of the resulting string of the hash function and the type of DHT to be used for the *main DHT*: once you have made these choices, they can no longer be changed.

4. ROUTING ON IP

The *heart* of the flat label system that uses current routing Internet architecture is the DHT of *standard labels* which includes the addresses of hosts and users, 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; the first demonstrations that follow relate to the management of the *main DHT*.

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 ∈ (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 will 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.

4.1 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 I need two assumption: first is that it is available worldwide. Data is partitioned and replicated using consistent hashing and consistency is facilitated by object versioning and among replicas is maintained by a quorum-like technique and a decentralized replica synchronization protocol. Second assumption that I 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?).

4.2 Name System. Conceptually the name system that should help the *main DHT* to improve FLIP performance is not different from the actual DNS; I 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, I 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.

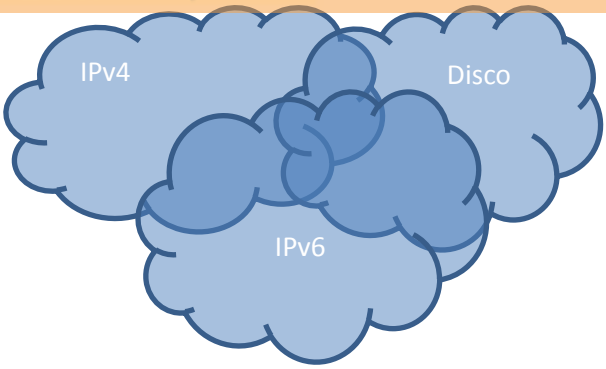

Figure 4. State of migration from IP to Disco for coexisting architectures using FLIP.

Intersections of clouds are 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 this function, and can publish or delete his election in name 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 I 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.

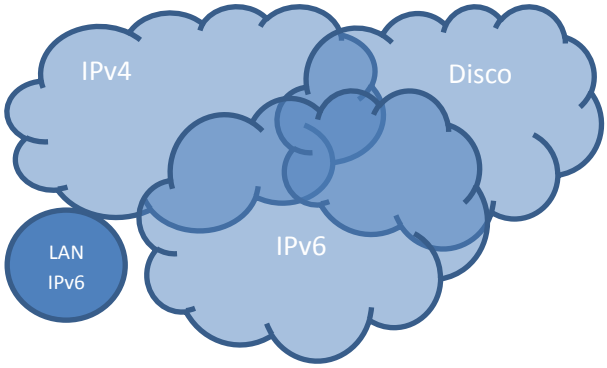

Figure 5. Example of a LAN with IPv6 addresses that tries to contact an IPv6 Internet node but that is only connected to an IPv4 Internet host. In this case, you must manually set the FLIP gateway for the LAN which then will call nsselect_next_gateway() function.

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** .

4.3 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.

5. 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.

5.1 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 temporary inconsistency between peer to peer query 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.

5.2 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);
- rsupdateNS(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* .

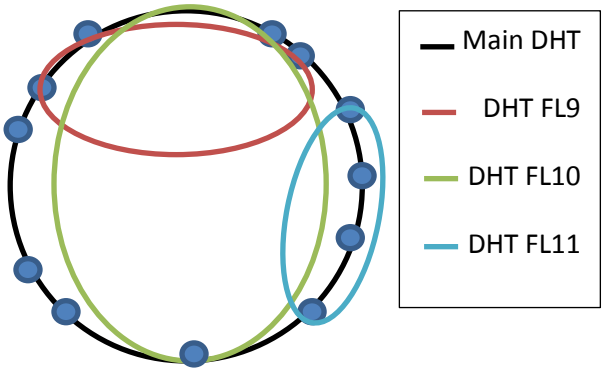

Figure 6. Main DHT nodes joined to other FL9, FL10 and FL11 type DHTs.

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. I 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.

6. ROUTING ON FLAT LABELS

Now that I've discussed about FLIP giving an idea of how generally layering it over IP, let's see what I think should be the real destination of IP to flat labels migration: Disco. After that, I'll talk about how layering FLIP on it.

6.1 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 I 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. Now I think: will be possible to change consistent hashing distributed database used in NDDisco with a flat label version of Dynamo? This will solve previous security issue because: 1) NDDisco utilizes simple key-value queries and 2) Dynamo has gossip-based membership protocol that avoids having centralized registry for storing membership and node liveness information (that's what NDDisco needs in this case).

6.2 FLIP over GVN routing on Disco. As about ROFL, I don't know what can be Disco network level protocol, but I don't care about that: maybe it will be FLIP, in the other case I 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 will 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 I 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.

7. CONCLUSIONS

The purpose of this paper is not to criticize harshly the various architectures to highlight the qualities of FLIP protocol. I wrote this document to raise the problem of the various aspects to consider while designing a new architecture for the Internet, which should not be limited to the performances of most used services or to innovations. Only after that, I propose a solution that could, if not take place between the various proposals, at least give an input to design a policy that considers also and above all the Internet community approval. Having said that I conclude with the hope that my 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.