Analysis of *relod.net*, a basic implementation of the RELOAD protocol for peer-to-peer networks

Marcos López Samaniego*, Isaias Martinez-Yelmo †, Roberto Gonzalez-Sanchez*

* Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
Avda. Universidad, 30. 28911 Leganés (Madrid). Spain
marcos.lopez@uc3m.es, rgonza1@it.uc3m.es

† Escuela Politécnica - Dpto. Automática
Universidad de Alcalá
Ctra. Madrid-Barcelona, km 33,600. 28871 Alcalá de Henares (Madrid). Spain
isaias.martinezy@uah.es

Abstract- **The P2PSIP Working Group is chartered to develop protocols and mechanisms for the use of SIP in distributed environments, thus minimizing the need for centralized servers. Under this premise, the RELOAD protocol was created, whose design was generalized to accept other applications with similar requirements, and which is currently in process of standardization by the IETF. In this paper, we present a basic implementation and an analysis of this protocol proposed standard, given the great interest displayed in recent years by the scientific and business community in issues related to peer-to-peer networks. Later, we conduct several experiments in order to validate its correct operation in real scenarios and provide feedback in relation with the current specification.**

Palabras Clave- **RELOAD, peer-to-peer, Chord**

I. INTRODUCTION

Peer-to-peer networks emerged last decade to make possible the replacement and recovery of resources over the Internet in a distributed way by creating overlay networks. Nevertheless, although peer-to-peer applications are popular nowadays, some open issues have not been addressed yet. One of the most challenging issues is the incompatibility between overlay algorithms, because their development was isolated. The Internet community is making some efforts to define mechanisms that allow the interoperability among different peer-to-peer networks. [1]

The P2PSIP Working Group is developing a protocol: RELOAD (acronym for REsource LOcation And Discovery), which allows the implementation of any peer-to-peer network, defining a common architecture and format. This protocol will become in the next months a new standard (RFC 6940) by the Internet Engineering Task Force (IETF), whose purpose is to provide support to applications that can work in distributed environments.

Even though it was linked to SIP from the beginning, RELOAD has not been developed only as a VoIP protocol, but rather its field of application has been extended so that it can be used by any other protocol with similar requirements.

This protocol acts by establishing an overlay network. Its modular design makes it possible to use it with any type of P2P network which is previously defined in the standard. A Chord algorithm is mandatory to implement.

In order to carry out the implementation, an object-oriented design based on a functional subset of the protocol was first carried out, and was then coded in such a way that the application can be easily modified and reused.

The structure of the paper is as follows. Section II presents the state of the art in relation with Pee-to-Peer networks. Later, section III introduces the main concepts of the RELOAD specification. Afterwards, section IV summarizes the main aspects of ***relod.net***. In section V, some results from our current implementation are shown to illustrate the operability of our design. Finally, a summary of our experiences and feedback from this work is provided in section VI. To conclude, conclusions can be found in section VI and future work in section VII.

II. STATE OF THE ART

A. Overlay peer-to-peer networks

A peer-to-peer (P2P) overlay network is a distributed collection of autonomous computers called *peers* that form a set of interconnections. These peers self-organize the overlay and have symmetric roles: they act as client and server simultaneously. Any peer can store objects, support queries and performs routing of messages [2].

These overlay networks have the following principles: self-organization, role symmetry, resource sharing, scalability, peer autonomy and resiliency [2].

There are three main types of P2P overlay networks:

1. Structured

Peers and, sometimes, resources are organized following specific criteria and algorithms, which lead to overlays with specific topologies and properties. They typically use distributed hash table-based (DHT) indexing [3]. This kind of networks is touted for their abilities to scale, tolerate failures, and self-manage, making them well-suited for Internet-scale distributed applications [4]. Some examples for DHT algorithms DHT are: CAN, Chord, Kademlia, or Pastry.

2. Unstructured

Unstructured peer to peer networks do not provide any algorithm for organization or optimization of network

connections. There are three models: the pure P2P systems or decentralized, like Gnutella and Freenet, the entire network consists solely of equipotent peers; there are no preferred nodes with any special infrastructure function. Hybrid peer-to-peer systems, like Kazaa, allow such infrastructure nodes to exist often called super nodes. Finally, centralized peer-to-peer systems, such as Napster, where a central server is used for indexing functions and to bootstrap the entire system. [3].

3. *Hierarchical*

A hierarchical overlay network is one in which several overlay networks of different types are connected to each other by means of another overlay network. This interconnection is usually performed by means of nodes known as super-peers, which are simultaneously part of two overlay networks. One example is H-P2PSIP [5].

III. RELOAD

A. *Introduction to the protocol*

The internet draft followed during design and implementation is *REsource LOcation And Discovery (RELOAD) Base Protocol draft-ietf-p2psip-base-26*, which expires on August 28, 2013 [6]. In this document the protocol is summarized as follows:

RELOAD is «a peer-to-peer (P2P) signaling protocol for use on the Internet. A P2P signaling protocol provides its clients with an abstract storage and messaging service between a set of cooperating peers that form the overlay network. RELOAD is designed to support a P2P Session Initiation Protocol (P2PSIP) network, but can be utilized by other applications with similar requirements by defining new usages that specify the kinds of data that needs to be stored for a particular application. RELOAD defines a security model based on a certificate enrollment service that provides unique identities. NAT traversal is a fundamental service of the protocol. RELOAD also allows access from "client" nodes that do not need to route traffic or store data for others.» [1]

B. *Basic concepts*

RELOAD is defined as a protocol for the application layer, the highest level of the TCP/IP protocol suite. As a transport level, it makes it possible to use the "secure" TLS (connection-oriented) or DTLS protocols (connectionless).

This protocol forms basically an overlay network in which internet, transport, and application levels are redefined, as well as a new routing level.

It can work with structured and unstructured overlay networks, which work as genuinely exchangeable plugins. The Topology Plugin is the RELOAD module that provides this functionality

The RELOAD node identifier is the Node-ID. It is composed of a variable number of bits – 128 in Chord. The Resource-ID occupies the same space as Node-ID, and identifies the resources stored in the overlay.

Each node is responsible for those Resource-IDs which are equal to or lower than their own Node-ID, and which in turn are higher than the immediately previous Node-ID.

Internet Model	Internet Model Equivalent in Overlay	RELOAD Architecture
Application	Application	SIP Usage, ShaRe Usage
	Transport	Message Transport, Storage
	(Routing)	Topology Plugin
	Internet	Forwarding & Link Managament
Transport	Link	TLS, DTLS

Fig. 1. RELOAD architecture.

C. *Functional modules*

1. *Message Transport*

This layer is responsible for end-to-end message transactions. It communicates with the use layer and with the storage component, and must be able to deliver messages to the destination node, whether it is a Node-ID or a Resource-ID.

Likewise, in the opposite direction, when Message Transport receives a new message, it delivers it to the relevant module, depending on the message type.

2. *Storage*

One of the features of RELOAD is the fact that is not only a messaging network; it is also a storage network. The overlay nodes keep available data which the usage needs. The storage component is in charge of storing the data and returning it when requested.

A Node-ID will necessarily store the resources of which is responsible, but it will also be sent requests to store Resources-ID which another node is in charge of. In this way, replicated data can be stored throughout the entire overlay, and a certain degree of redundancy against a node failure exists.

The kind of data that can be stored by a node is known as their Kind and is identified by its Kind-ID, which is a 32-bit integer assigned by IANA (or else belonging to a private range). A Resource-ID can contain several Kind-IDs.

The model for the data that can be stored in a Kind-ID is known as a Data Model. In principle, three Data Models are considered, although future usages might define new models. The possible types to be stored can be: an individual value; an array, multiple values indexed by a number; or a dictionary, several values indexed by a key or String.

3. *Topology Plugin*

The Topology Plugin defines a generic structure on which several structured and unstructured peer-to-peer overlay algorithms can work. The functions of these plugins are basically defining the network topology and routing the messages through the nodes.

However, not any type of pre-existing algorithm can work, but rather different algorithms will be defined or redefined in the future so that they can work on RELOAD. The only plugin currently defined is Chord, a DHT algorithm, which has been slightly modified with respect to its original design so it can work on RELOAD. It must be possible to replace Chord by another algorithm without

affecting the rest of levels to continue to work without being modified in any way.

The topology plugin is responsible for the routing and connection table maintenance. The configuration of these tables depends on the algorithm in use. The Topology Plugin must be queried to make decisions about the packets routing.

4. Forwarding & Link Management

This layer communicates with the Topology Plugin to obtain connection tables and routes, and thus deliver the message to the following node. It is responsible for establishing connections with new nodes and passes through NAT and firewalls using ICE.

Forwarding & Link Management has access to the TCP/IP transport level, and is in charge of maintaining connections, so it is responsible for receiving overlay messages and place new packets on the network at the request of higher layers.

5. Link layer

Link provides an extra header known as the Framing Header (FH), which only makes sense in the context of the link and is removed at each hop.

On the one hand, when a reliable protocol (e.g.: TCP) is selected, it is used to frame messages and to provide timing. On the other hand, due to the unreliable nature of UDP, when DTLS is chosen at the link level of the overlay network, use of the Simple Reliability protocol is required. This protocol makes use of the Framing Header to provide congestion control and semi-reliability.

D. Basic fields

Unlike the Internet architecture, RELOAD levels do not define their own headers. Rather, there is a common message to all overlay network levels, which has three parts: Forwarding Header, Message Contents, and Security Block.

1. Forwarding Header

This header includes fields that identify the RELOAD protocol (RELO token), version, overlay name (e.g.: Chord-RELOAD), number of the sequence identifying the configuration file, TTL (time-to-life in number of hops), fragmentation, transaction identifier (a random number), maximum response length, routing addresses, and options.

Addresses include two fields. The first one is the DESTINATION LIST, an array of destination addresses. A message will be routed in strict order through the nodes that appear in the list.

The other relevant field is the VIA LIST, a second array where nodes are being crossed by the message in every hop are gradually added. When a node forwards a message, it places the peer that delivered the packet at the end of the VIA LIST.

RELOAD works with recursive symmetrical routing, which consists in the fact that, when a destination node receives a request which must be answered, it generates a DESTINATION LIST in the answering message, turning the VIA LIST around.

2. Message Contents

The fields in this block are the message code, the message body, and the extensions.

The message code is an integer that identifies the content of the message (Store, Fetch, Join, Leave…, and whether it is a request or an answer). Once identified, the message body is delivered to the responsible module (Storage, Topology…), which will decode it and generate an answer if required.

3. Security Block

This last block includes a number of certificates required to verify signatures (they can be of various types, e.g.: X.509), it specifies the hash and signature algorithms and, finally, includes the certificate hash and the value of the signature.

E. Usage layer

RELOAD cannot work by itself, it is designed to support a variety of applications. The usages that can be given to RELOAD are precisely known as *usages*. A usage defines how an application can store its information in the overlay; it may define its own data types, along with the rules for their use. One single application may require multiple usages.

A vast number of usages are being currently defined to work on RELOAD, in addition to SIP. Some of them are Distributed Conference Control (DisCo) [10], Shared Resources (ShaRe) [11], Constrained Application Protocol (CoAP) [12], Simple Network Management Protocol (SNMP) [13], Service Discovery [14], Public Switched Telephone Network (PSTN) Verification [15]…

F. Encapsulation example

Fig.-2 shows an example of how a Join Request packet is processed. Link layer delivers the message to Forwarding & Link Management but this module does not contribute with any header. Afterwards, it comes to Message Transport, which will be able to decode the structure. The Message Body field, which is opaque to Message Transport module, is delivered to the responsible layer: Topology Plugin in the case of a Join message. Header and Footer do not represent any particular field in the message, just the rest of the structure after and before Message Body.

G. Working diagram

It can be seen in fig.-3 how the different modules communicate with each other. This diagram of course does not represent all classes in detail; this is an extremely simplified scheme that represents how the implementation is designed, to validate RELOAD architecture in the standard.

Link, Forwarding & Link Management and Storage are symbolized by a single class, this class has in fact all the main code from these modules, but other secondary classes exist and are not represented in the diagram.

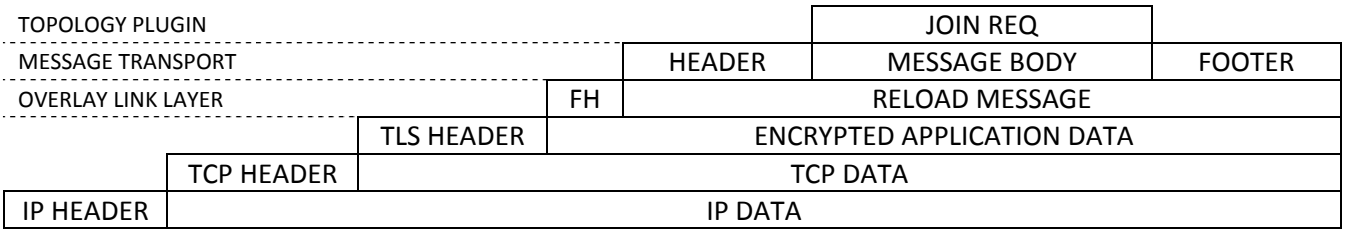


Fig. 2. Encapsulation example in RELOAD.

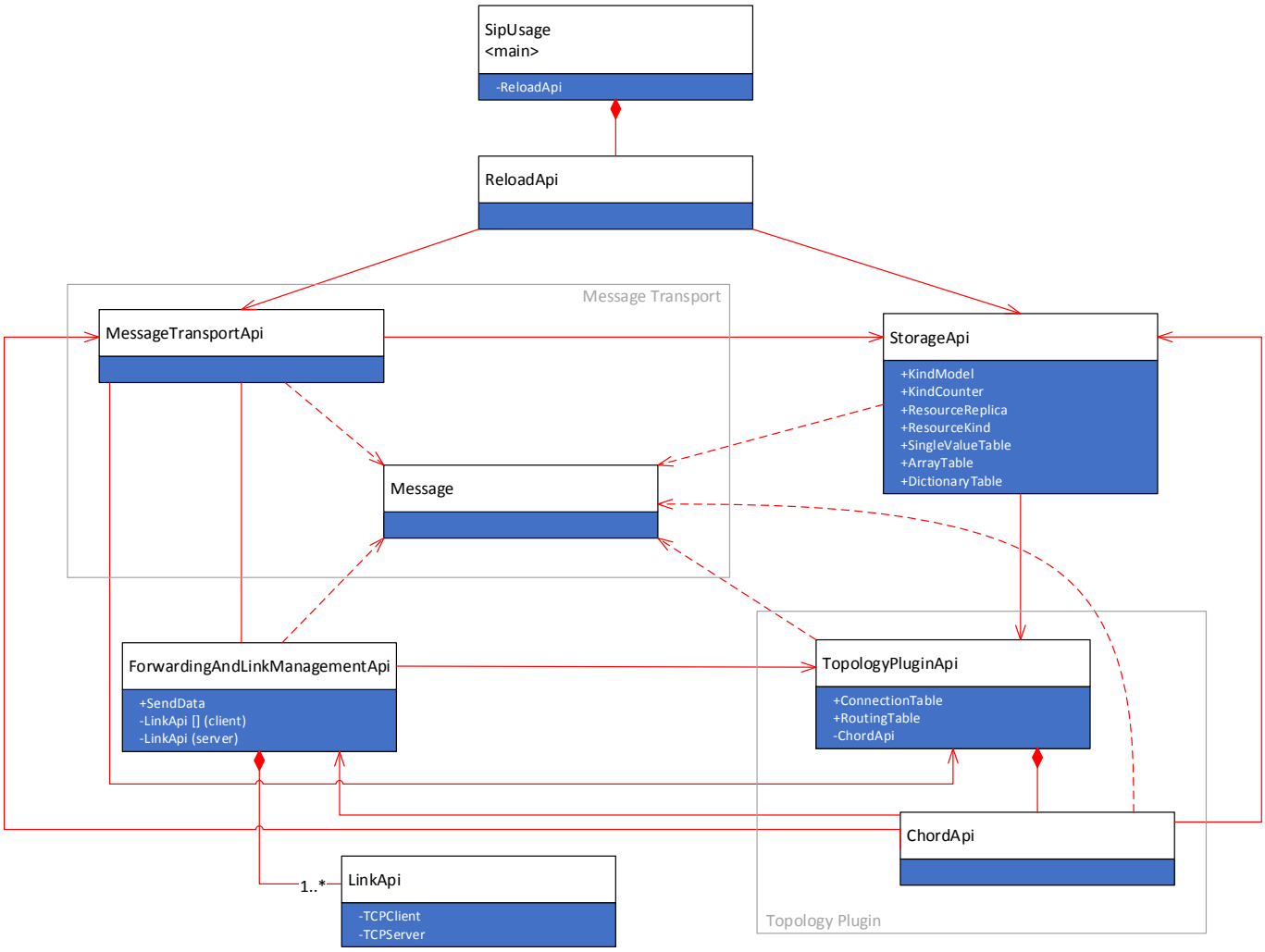


Fig. 3. UML Diagram.

By the other hand, Message Transport and Topology Plugin have two classes.

The Topology Plugin module will have an additional API-class for every algorithm implemented. While ChordApi has the code from Chord, the TopologyPluginApi is a standard API which has all the methods that can be called by the rest of the layers, thus, it is impossible for the other modules know the algorithm in use.

MessageTransportApi is the API for the Message module, which includes all the needed methods in this layer; while Message class is just an object containing the message structure in RELOAD, which will be instantiated by the most of the modules.

Finally, RelodApi and SipUsage are real classes, where any other usage can be implemented instead of SIP.

IV. Implementation

A. Introduction

Our work consists in an implementation and benchmark for RELOAD: ***relod.net***. Due the specification length, a subset with basic functionality is described here.

In addition, a reduced version of SIP has been implemented to check the operation of the entire system, taking as its base the existing draft: *A SIP Usage for RELOAD draft-ietf-p2psip-sip-09* [7].

The chosen programming language is Java. So far, no Java implementation of the protocol is known – only C and C++ implementations have been carried out and they are not publicly available yet. In addition to previous reasons, Java is chosen due to its object orientation and its multi-platform nature.

B. Code structure

There are seven main Java packages in each of the modules that form RELOAD: the link level, Forwarding and Link Management, Topology Plugin, Message Transport, Storage, one more for common classes to various modules, and finally, one for the specific packages of the usage in question, in this case SIP. In turn, these may contain more subpackages.

C. Design considerations

A modular design was developed, emphasizing the independence of the various layers in the overlay network.

Each package has a class, which is the Application Programming Interface (API) – they are the only means for access from other modules. This fact simplifies operation and ensures that, for example, all requests that can be made from the Storage module are addressed to the StorageApi class only. However, internally within each layer, there are no restrictions on how classes communicate with each other. This design makes possible for each module to be independent, retaining control over how some parts of the code communicate with others.

Likewise, there is a class that groups all protocol modules: ReloadApi. These APIs mask the complexity of each of the modules and the application itself.

The code seeks to make maximum use of Java's object orientation. Therefore, each structure defined in the draft is encoded in a different class. This implementation is innovative, since instead of using classic mechanisms such as parsers to encode and decode structures, a new procedure has been designed.

D. Object structure

Each structure is modeled as simple as possible: each one of the fields is encoded by means of an attribute, which may be a basic Java type or another structure (i.e. an object). Each structure usually has two builders: the first one, whose arguments are the same as the class attributes and whose assignation is thus trivial; and a second builder whose only argument is a byte array which also fills in the attributes after the structure is decoded.

The purpose of the first builder is to generate a message that will be delivered through the overlay, whereas the second builder is used when a packet is received from the network and we need to know its contents. Decoding is made recursively: each class decodes its structure and, if it contains attributes that are not primitive types, it calls the builders of those structures in turn. Coding is performed by calling the *getBytes()* method and it is also recursive.

V. Results

A. Testing environment

Several tests are documented here. Various aspects of the implementation are tested, such as the overlay registration – when they access the overlay for the first time –, storage, or the way in which they exchange routing information by means of Update messages.

A Peer-to-Peer overlay network forms with twelve peers is used in our tests. All peers are in the same LAN network or they are reachable through public IP addresses, NAT traversal support has not been implemented yet.

A number higher than ten is selected in order to be big enough in order to validate the behavior of routing tables based on fingers in Chord. Therefore, it will typically be one or two fingers, and some nodes will not be connected to each other, so messages must be routed through the overlay.

A basic implementation of SIP as usage [7] is being used in the tests, so when each node is registered its overlay location information is saved in the peer-to-peer network, so any overlay node participating in the peer-to-peer overlay can later retrieve it.

B. Basic operation sequence

Even though all the overlay nodes are only identified by their Resource-ID / Node-ID, their IP addresses are mentioned so that figures illustrating the text can be more easily understood. In this section, diagrams show the theorist working from the RFC and they can be compared to the real-application Wireshark captures.

1. Overlay registration (Fig.-4)

When a node accesses the overlay – Joining Peer (JP), with IP 192.168.1.15 –, it makes a connection to a Bootstrap Node (BN), whose IP address it previously knows, in this case: 192.168.1.70. This is the fifth node to initialize, so it contacts 4 other peers, which are added to the neighbor table.

Then an Attach message is sent to the relevant Resource-ID – its own Node-ID plus one –, which is routed via the bootstrap to its Admitting Peer (AP), whose IP address is 192.168.1.13.

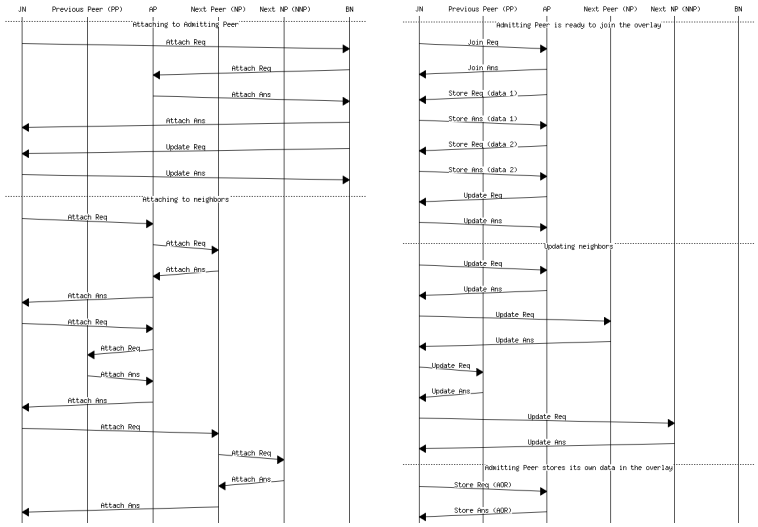
After receiving the Attach message, a direct connection is established between the Joining Peer and the Admitting Peer, direct messages are sent between these two nodes, and by this means the JP receives information about the AP's neighbors, which are also its own neighbors. It then sends an Attach message to the three other overlay nodes (two of which were routed through the AP), and, as it is now ready to become a part of the overlay, it sends a Join message to its Admitting Peer.

After this, the JP sends three Store messages in which it provides information about three Resource-IDs of which the AP will now be responsible. The JP then sends an Update to the AP, stating that it is now one more node in the overlay. The AP now sends Updates to its entire routing table to make it known that it is ready to route for them. Finally, SIP usage registered its Address Of Record (AOR) in the overlay by means of a Store message (the fact that the peer responsible for storing this information is the AP is coincidental, and any other peer in the overlay might have played this role).

2. Data procurement and remove (Fig.-5)

Once all the nodes have been initialized a mapping of an AOR in another AOR is stored in the overlay by sending a Store message and receiving the corresponding answer.

Then, a couple of queries for different AORs are made



Time	Source	Destination	Protocol	Length	Info
0.022979	192.168.1.15	192.168.1.70	RELOAD	1269	Attach Request
0.158876	192.168.1.70	192.168.1.15	RELOAD	1304	Attach Response
0.199647	192.168.1.13	192.168.1.15	RELOAD	1304	Update Request
0.255607	192.168.1.15	192.168.1.13	RELOAD	1246	Update Response
0.399271	192.168.1.15	192.168.1.13	RELOAD	1268	Attach Request
0.420789	192.168.1.13	192.168.1.15	RELOAD	1286	Attach Response
0.443881	192.168.1.15	192.168.1.13	RELOAD	1268	Attach Request
0.462110	192.168.1.13	192.168.1.15	RELOAD	1286	Attach Response
0.478593	192.168.1.15	192.168.1.14	RELOAD	1268	Attach Request
0.562027	192.168.1.14	192.168.1.15	RELOAD	1286	Attach Response
0.625787	192.168.1.15	192.168.1.13	RELOAD	1261	Join Request
0.643610	192.168.1.13	192.168.1.15	RELOAD	1245	Join Response
0.772796	192.168.1.13	192.168.1.15	RELOAD	1453	Store Request
0.786422	192.168.1.15	192.168.1.13	RELOAD	1261	Store Response
0.791981	192.168.1.13	192.168.1.15	RELOAD	1453	Store Request
0.797329	192.168.1.15	192.168.1.13	RELOAD	1261	Store Response
0.811386	192.168.1.13	192.168.1.15	RELOAD	1320	Update Request
0.844412	192.168.1.15	192.168.1.13	RELOAD	1246	Update Response
1.005322	192.168.1.15	192.168.1.13	RELOAD	1250	Update Request
1.011821	192.168.1.13	192.168.1.15	RELOAD	1246	Update Response
1.030923	192.168.1.15	192.168.1.14	RELOAD	1250	Update Request
1.040872	192.168.1.14	192.168.1.15	RELOAD	1246	Update Response
1.045741	192.168.1.15	192.168.1.12	RELOAD	1250	Update Request
1.053522	192.168.1.12	192.168.1.15	RELOAD	1246	Update Response
1.063275	192.168.1.15	192.168.1.70	RELOAD	1250	Update Request
1.112422	192.168.1.70	192.168.1.15	RELOAD	1250	Update Request
1.118390	192.168.1.15	192.168.1.70	RELOAD	1246	Update Response
1.123321	192.168.1.70	192.168.1.15	RELOAD	1246	Update Response
1.140494	192.168.1.15	192.168.1.13	RELOAD	1454	Store Request
1.166302	192.168.1.13	192.168.1.15	RELOAD	1311	Store Response

Fig. 4. Peer initialization messages.

Time	Source	Destination	Protocol	Length	Info
8.7396110	192.168.1.15	192.168.1.2	RELOAD	1350	Store Request
8.7410840	192.168.1.2	192.168.1.15	RELOAD	1261	Store Response
10.674067	192.168.1.15	192.168.1.13	RELOAD	1279	Fetch Request
10.691594	192.168.1.13	192.168.1.15	RELOAD	1471	Fetch Response
15.695138	192.168.1.15	192.168.1.16	RELOAD	1279	Fetch Request
15.706263	192.168.1.16	192.168.1.15	RELOAD	1453	Fetch Response
23.638374	192.168.1.15	192.168.1.13	RELOAD	1330	Store Request
23.655468	192.168.1.13	192.168.1.15	RELOAD	1297	Store Response

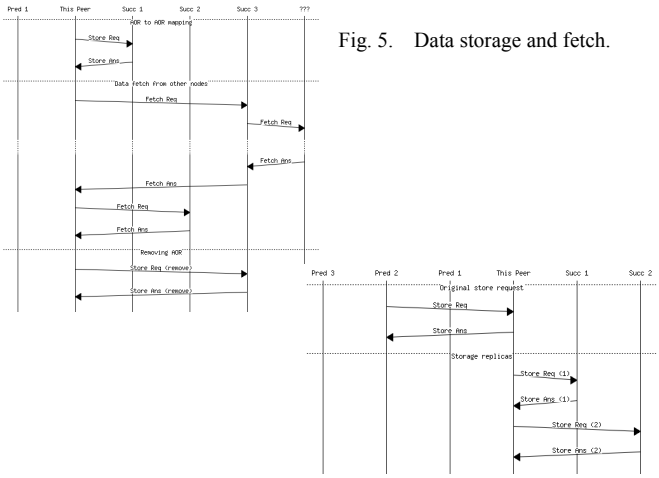


Fig. 5. Data storage and fetch.

Time	Source	Destination	Protocol	Length	Info
1.016677	192.168.1.1	192.168.1.15	RELOAD	1490	Store Request
1.024823	192.168.1.15	192.168.1.1	RELOAD	1329	Store Response
1.025991	192.168.1.15	192.168.1.2	RELOAD	1453	Store Request
1.029554	192.168.1.2	192.168.1.15	RELOAD	1261	Store Response
1.034978	192.168.1.15	192.168.1.16	RELOAD	1453	Store Request
1.043227	192.168.1.16	192.168.1.15	RELOAD	1261	Store Response

Fig. 6. Replica storage.

Fig. 7. Join Request message detail.

via the console. The Fetch requests and answers are shown, which, in this case, provide the Node-ID for the AOR in question. The answer to this Fetch message is shown on the screen via the console.

Finally, we remove our AOR, by means, as we will see, of a Store that crushes the stored information, as there is no specific message for removals.

3. *Replication (Fig.-6)*

Once our peer has become a part of the overlay, another one, whose IP address is 192.168.1.19, wishes to store its information in a Resource-ID that belongs to us.

Firstly, we receive a Store message, but it does not come from the expected node. This happens because, given that the overlay has many nodes, we are not the neighbor or the finger of the node generating the packet, so that the packet is routed through the overlay – in this case, with an intermediate hop, through the peer with IP address 192.168.1.1. Given that routing is symmetrical and recursive, the answer is returned to that same node.

This information must be replicated twice, so it is forwarded to our two first successors. Our immediate successor is the node with IP 192.168.1.2, so that the information is directly delivered. The next successor is 192.168.1.16, so it is delivered in the same way.

C. *Join traffic (Fig.-7)*

This screenshot shows the Join Request message in the previous section (*Overlay registration*). This picture is a detail of the eleventh packet in fig. 4.

The Forwarding Header shows the values mentioned above. The RELOAD version is $0x0A_{16} = 10_{10} = \mathbf{1.0}$. The message is sent to a Node-ID (the Admitting Peer), the message code is 15 (Join Request), and its only content is the Joining Peer's Node-ID.

D. *Fetch traffic (Fig.-8)*

The following packet to be analyzed is the first Fetch message in fig. 5, from *Data procurement and remove*, in the previous section.

It can be seen how, in the DESTINATION LIST FIELD, the destination is a Resource-ID, so the responsible Node-ID for that resource will receive the message. The message code is number 33 (Join Request), and the message body includes the Resource-ID again. It is a simple request that has only one Kind-ID (number 1: "SIP-REGISTRATION"). A GENERATION COUNTER of 0 specifies that it is an original (not replicated) message, and no dictionary keys are specified (*0 keys*), as SIP usage indicates that requests must be empty, so that all the values stored in that resource will be returned in the answer.

E. *Store traffic (Fig.-9)*

Finally, a Store Request from *Replication* will now be examined. The picture is a detail of the third packet in fig. 6, where a request to store some replicas was made.

Even though it is not shown due to space constraints, the destination of this message is a node, the Node-ID where we want to store the replicated message. The message code is 7 (Store Request), and in the content appears the Resource-ID where the original content is stored – a resource for which the node receiving this message is of course not responsible.

It can be seen that this is a "SIP-REGISTRATION" Kind-ID, the GENERATION COUNTER is 1, which means that this is a replicated message in the first successor, and that a single dictionary entry is stored, with one key and one value. SIP usage defines that the key is a node identifier where the person to be called can be located, while the value is a structure with additional SIP data.

VI. ANALYSIS

A. Implementation performance

These values include the Java Virtual Machine resource consumption. This test was made on an Intel Core i7 950 Quad-core clocked at 3 GHz, on Microsoft Windows 7.

In a small network with 10 or 12 nodes running ***relod.net***, the CPU usage is at 2.1% maximum during the registration in the overlay, after this, the usage drops to a value very close to 0%. The RAM consumption is between 23.8 and 25.8 MB, peaking at the initialization.

B. Feedback and suggested improvements

1. Feedback

Generally speaking, the standard structure is complex. Just taking a look to it, it is not clear to which module messages belong, or how the layers are encapsulated. In this sense, other protocols such as P2PP are easier to understand, and even most of the documents published by the IETF published are less complex.

Further, some of the relationships between modules are not so obvious. In certain messages, such as a Ping Request, when a node receives a message in the link layer of the overlay network, it delivers the message to Forwarding, so that it decides whether it belongs to it or whether it must be forwarded to another node. To do so, it queries the Message Transport layer, as it is unable to understand the structure by itself, given that Forward has no associated header. After checking that the message belongs to it, it is finally sent to the higher layer, which is again Message Transport.

Message analyses the packet, checks that it is a Ping Request and that the responsible module is Forwarding, so the message body is delivered to this layer, which will be able to process its content. This module generates the answer, which will be sent through the opposite path.

It is obvious that it is not easy for such a relevant layer as Forwarding & Link Management not to have any associated header (it would be perfectly possible for it to have access to the data that would enable it to decide whether to forward the message or not). It is in no way understandable the fact that the Forwarding layer delivers a message to a higher layer, which immediately returns the message to this same Forwarding layer.

2. Suggested improvements

The RFC does not define a link layer proper in the overlay network. It merely specifies that the Link level can be TLS or DTLS but then, incoherently, it defines an extra header on that same level: Framing Header.

We suggest creating a new intermediate layer between TLS and Forwarding called “Link”, which would be responsible for the tasks assigned by the document: congestion control, semi-reliability, and timing.

C. Benefits

The creation of a standard protocol such as RELOAD can be regarded as a landmark in the history of the Internet. Its approach involves a shift from the current client-server model to a new model distributed between peers. It can be expected that businesses will tend to gradually assume this paradigm shift, due to the high costs of centralized servers.

This protocol is particularly significant due to the extensibility it allows, as is it not conceived for a specific protocol, but rather it allows multiple usages. It also stands out for its ease in adapting new protocols and turning them into RELOAD usages. Finally, it is highly flexible, as any type of peer-to-peer algorithm can be used.

Fig. 8. Fetch Request message detail.

Fig. 9. Store Request message detail.

This implementation was carried out to promote these features. Its extremely modular design makes it possible to create new topology plugins with no need to change the rest of the code, and even if the source code is missing, as each of the modules has an easily accessible API as a standard access from other modules.

In addition, a general class is given: ReloadApi, which provides a high-level API for the whole protocol. Any usage to be programmed on this implementation of RELOAD will only need to create a ReloadApi-type object and make calls to the methods in this class.

VII. Conclusions

This paper analyzes ***relod.net***, one of the first RELOAD implementations, protocol that will become peer-to-peer networks' new standard. However, it does not only represent a change when it comes to the creation of new software that works in a distributed way; its main challenge is the redefinition of widely accepted protocols by the industry, which once adapted, the Internet will gradually change from the client-server model into a new paradigm that will minimize the need of centralized servers.

This work aims to show how possible is to make a basic implementation of RELOAD extracting the main features of the standard, at the same time that it suggests a modular design that would even allow that two different modules encoded by different companies could work jointly.

To provide the interoperability, it's necessary to define certain APIs, which determine what calls are permitted from one module to another. Regarding this issue, it is especially important to reach agreement on the Topology Plugin module, and to set a standardized API as a way to ensure that future overlay algorithms can operate with existing implementations.

Therefore, we have published the Java API documentation and we have also decided to release the source code to the community, which is available in:

http://download.relod.net/

This application has been programmed through a whole year. The program consists of 11,000 code lines, which occupy a total of 554 kilobytes, stored in 163 Java classes, in 28 packages.

Although this is an *alpha* version that, due to their level of maturity, interoperability with other implementations cannot be guaranteed, we believe it could become a reference to other developers, for its proper design and simple code. In addition, the program will be updated while is being completed, until it becomes a fully functional application.

VIII. Future work

The first of our goals is to complete the missing parts, such as transversal NAT, security, and clients. It is also very important to focus on interoperability with other implementations: once the security module is completed, it will be possible to test it using other programmers' software.

The creation of a second algorithm that works jointly with Chord is proposed. No other peer-to-peer has currently been defined yet, so potential work might involve adapting an existing algorithm for usage on RELOAD. Designing a usage from scratch might be equally interesting, modifying an already existing protocol to work in distributed environments, or else designing a new one.

In the storage module, data are stored in RAM memory. This might suffice in many scenarios, but in others it might be preferable for data to be stored in a hard disk or a solid-state drive. For this reason, the use of databases is suggested.

Finally, it would be advisable to test the scalability of the implementation on a larger number of nodes using a network emulator such as ModelNet or PlanetLab. To ensure proper operation, the idea is to try different scenario setups with at least 1,000 peers.

Acknowledgments

This research was supported in part by the Comunidad de Madrid grant S-2009/TIC-1468 (MEDIANET project).

References

[1] Isaias Martinez-Yelmo, Roberto Gonzalez-Sanchez and Carmen Guerrero. "Validation of H-P2PSIP, a scalable solution for interoperability among different overlay networks". *Peer-To-Peer Networking and Applications*, vol. 6. no. 2, pp. 175-193, 2013.

[2] John F. Buford, Heather Yu and Eng Keong Lua, *P2P Networking and Applications*. Burlington, Massachusetts: Elsevier, 2008, pp. 29-31.

[3] D. Vivekanandreddy, Allamprabhu Vastrad and R. M. Nareshkumar, "Implementation of a novel optimized trust based search approach for the peer to peer (P2P) platform", *World Journal of Science and Technology*, vol. 2, no. 10, pp. 129-132, 2012.

[4] Tallat M. Shafaat, Ali Ghodsi and Seif Haridi, "Dealing with network partitions in structured overlay networks", *Peer-To-Peer Networking and Applications*, vol. 2. no. 4, pp. 334-347, 2009.

[5] Isaias Martinez-Yelmo, Alex Bikfalvi, Ruben Cuevas, Carmen Guerrero and Jaime Garcia. "H-P2PSIP: Interconnection of P2PSIP domains for Global Multimedia Services based on a Hierarchical DHT Overlay Network". *Computer Networks: The International Journal of Computer and Telecommunications Networking*. vol. 53 no. 4, pp. 556-568, 2009.

[6] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset and H. Schulzrinne, *REsource LOcation And Discovery (RELOAD) Base Protocol draft-ietf-p2psip-base-26*, February 24, 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-p2psip-base-26

[7] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset and H. Schulzrinne, *A SIP Usage for RELOAD draft-ietf-p2psip-sip-09*, February 25, 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-p2psip-sip-09

[10] A. Knauf, G. Hege and M. Waehlisch, *A RELOAD Usage for Distributed Conference Control (DisCo) draft-ietf-p2psip-disco-00*. October 9, 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-p2psip-disco-00

[11] A. Knauf, T C. Schmidt, G. Hege and M. Waehlisch, *A Usage for Shared Resources in RELOAD (ShaRe) draft-ietf-p2psip-share-01*. February 24, 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-p2psip-share-01

[12] J. Jimenez, J. Lopez-Vega, J. Maenpaa and G. Camarillo, A Constrained Application Protocol (CoAP) *Usage for REsource Location And Discovery (RELOAD) draft-jimenez-p2psip-coap-reload-03*. February 18, 2013. [Online]. Available: http://tools.ietf.org/html/draft-jimenez-p2psip-coap-reload-03

[13] Y. Peng, W. Wang, Z. Hao and Y. Meng, *An SNMP Usage for RELOAD draft-peng-p2psip-snmp-05*. October 18, 2012. [Online]. Available: http://tools.ietf.org/html/draft-peng-p2psip-snmp-05

[14] J. Maenpaa and G. Camarillo, *Service Discovery Usage for REsource LOcation And Discovery (RELOAD) draft-ietf-p2psip-service-discovery-08*. February 23, 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-p2psip-service-discovery-08

[15] M. Petit-Huguenin, J. Rosenberg and C. Jennings, *A Usage of Resource Location and Discovery (RELOAD) for Public Switched Telephone Network (PSTN) Verification draft-petithuguenin-vipr-reload-usage-04*. March 12, 2012. [Online]. Available: http://tools.ietf.org/html/draft-petithuguenin-vipr-reload-usage-04