IPv6 Hash-Based Addresses for Simple Network Deployment

Renzo Davoli
Computer Science and Engineering Department
University of Bologna
Bologna, Italy
Email: renzo@cs.unibo.it

***Abstract*—The configuration of an IPv6 network is a rather daunting and error-prone procedure for system administrators. Each node must be provided with its own (128 bit long) IPv6 address and with a domain name manageable by human beings. Autoconfiguration methods can give addresses to interfaces but do not provide any means of configuring the DNS. This paper presents some methods based on hash functions which highly simplify the network configuration process. System administrators just need to define the fully qualified domain names of all the networking nodes (servers and clients) and the networking prefixes for each LAN or subnet. Each node will acquire its own IPv6 address and the DNS server will be automatically configured to support name resolution for all the nodes. The whole process does not require system administrators to type any IPv6 addresses, and it is fully compatible with existing protocols for autoconfiguration and name resolution.**

***Index Terms*—IP networks; TCPIP; Domain Name System; Next generation networking;**

I. Introduction

IPv6 standard provides two different ways for auto-configuration - stateless and stateful. In stateless auto-configuration [1] each networking node broadcasts a router solicitation request to obtain a list of address prefixes active on that local area network. The node then self-assigns an IPv6 address by combining each prefix it receives with the EUI-64 interface hardware address. Stateless auto-configuration provides the node with valid addresses and routing information but it does not configure the DNS (Domain Name Service) server address, nor does it configure its own DNS entries.

As described in [2] and [3], this kind of auto-configuration can cause privacy problems, since the hardware address, usually the Ethernet MAC (Media Access Control) address, is part of the final IPv6 address. So, it is possible to track the movements of personal computers, laptops, tablets, smartphones etc.. The more these devices are personal *digital extensions*, the more this permits the tracking of people, their physical locations and habits. Stateless auto-configuration changes the IPv6 address of a node in cases of network adapter substitution. On the other hand, stateful configuration [4] is based on DHCPv6 (Dynamic Host Configuration Protocol). In stateful configuration, a node broadcasts a DHCP request using its own link-local IPv6 address. The server then replies, providing the node with its own global IPv6 address (or addresses).

Stateful auto-configuration is not self-configuration. In fact, whilst no configuration effort is required from the client, the mapping between hardware and IP addresses must be configured at the server, by hand.

Another problem related to address configuration is DNS mapping. Forward and reverse DNS mapping is compulsory for servers, but it is useful to give symbolic names to clients, too. Numeric addresses, especially 128 bit IPv6 addresses, are hard to use: symbolic names facilitate the management, e.g., the tracking of networking problems.

Both DHCP and DNS configuration involve the typing of several IPv6 addresses: sequences of 32 hexadecimal numbers. Naturally this is highly prone to error.

The results presented in this paper introduce a set of methods that can help the system and network administrators to set up an IPv6 network in a simple and effective way. These methods provide the networking nodes of a local area network with their addresses, given an IPv6 prefix, a domain name of the LAN (Local Area Network) and the list of host names to be configured. Each node just needs to know its own Fully Qualified Domain Name (FQDN). The only hand typed IPv6 address is the prefix of the LAN. Those methods also provide forward name resolution and, if required, reverse name resolution.

This paper is organized as follows. The next section introduces the idea of hash-based address by presenting some implementation scenarios. Section III discusses the limits of the proposed approach and section IV analyzes the cases of address collision. The final part of the paper include sections about a proof-of-concept implementation, a comparison with the related work available in the literature, and final remarks, also discussing the future developments of this project.

II. Hash-based Addresses

The core idea is to compute a 64 bit encoding of the FQDN of each node and to use it as its host address, following the 64 bit prefix fixed for each specific LAN. This idea can be implemented in many ways. Each one has pros and cons.

A. Assisted DHCP and DNS management.

Given the list of nodes, a script computes both DHCP and DNS tables by generating a hash-based address for each FQDN (see Fig. 1). The router must be configured for stateful

 ISBN: 978-1-61208-300-1

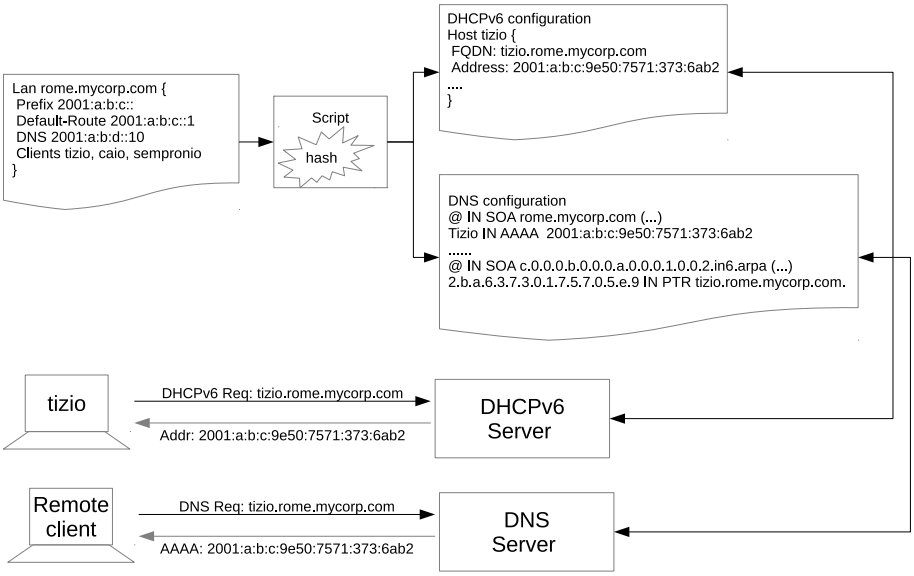

Fig. 1. Assisted DHCP and DNS management.

auto-configuration, both DHCP servers and clients must use the fully qualified host names as client identifiers. When a node starts, it learns from a router advertisement packet that stateful auto-configuration is required. Then it sends a DHCPv6 request using its FQDN as its identifier. The DHCP server assigns to each node its specific address. This method uses standard DHCP and DNS servers and clients. Node addresses are not related to hardware addresses, so they do not change in case of network adapter substitution. DHCP and DNS tables must be recomputed if a name gets modified or a new node is added. Given an IPv6 prefix, a LAN domain name and the list of host names to configure, this method provides the networking nodes of a local area network with their addresses. The use of hash-based addresses should be preferred to other assignment rules, for instance, by a script which enumerates the hosts, as there is no need for system administrators to store the mapping between each host name and its address. The address can be retrieved by running the hash function when needed. The management of the accountancy for unused and reassigned host addresses is also unnecessary.

B. Hash address self-assignment and automatic DNS forward resolution.

This method does not even require a list of nodes. Each node autonomously compute its own address by combining the prefix learned from the router advertisement and the hash of its FQDN (see Fig. 2). A custom DNS server for the subnet provides a forward resolution for any name within the domain of the local area network. This method simplifies the deployment of local area networks composed of client computers. Provided no nodes share the same FQDN and there are no hash collisions, the IPv6 address assignment and the DNS forward resolution require no configuration. Name

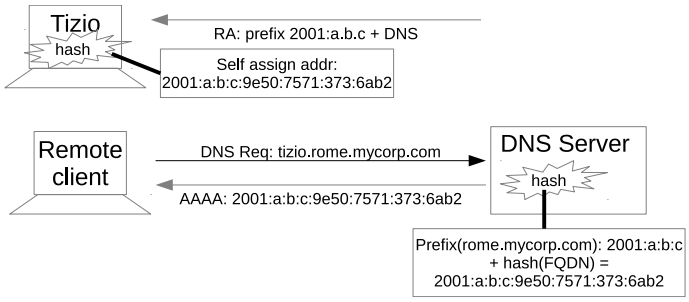

Fig. 2. Hash address self-assignment.

collisions are very rare events, as shown in section IV. This solution requires a specific DNS server and a program to self-assign the address.

C. Hash address self assignment plus DNS forward and reverse resolutions

By itself, the DNS can obtain the address of any host in the subnet by computing the hash of the FQDN, but it is not possible to perform the reverse resolution automatically. The hash function is not injective. In any case, it is possible for the DNS to cache the mapping between names and their addresses so as to be able to give correct answers to reverse DNS resolution, requests. The DNS should not store any DNS resolutions as it may refer to non-existent nodes: an attack based on DNS requests for large numbers of random host names could cause table overflows and delays. The simplest solution is to store the reverse mapping of a host only if the request comes from the address which is being resolved. This means that a host can request its own reverse resolution by introducing itself to the DNS server: it sends a DNS forward resolution of its own FQDN. The DNS server gets the request,

 ISBN: 978-1-61208-300-1

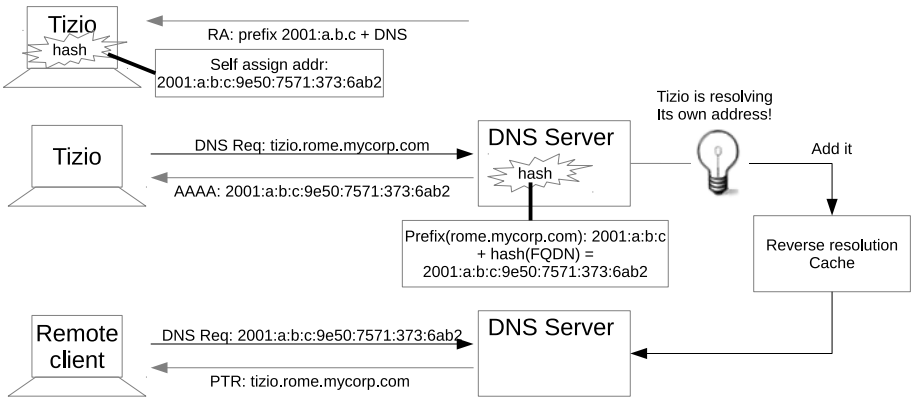

Fig. 3. Hash address self-assignment supporting reverse DNS mapping

uses the hash mapping and recognizes that the request comes from the same address, so it saves the mapping for the reverse resolution (Fig. 3). In this way, it is not possible for external DNS clients to add fake reverse mapping items. However, some assumptions are needed to grant this security property: the internal nodes and the routing must be trusted. Otherwise, misbehaving internal nodes, or nodes able to forge source addresses of the local network, can attack the DNS and add fake entries.

D. Further methods

The methods described in II-B and II-C have been designed for an environment where each LAN is assigned its specific prefix and all the nodes on the LAN belong to the same domain. Although this is quite a common situation, hash-based addresses can also support different and more general scenarios.

- Several Domains, one prefix. The hosts of several domains use the same 64 bit prefix for their IP addresses. This scenario is already supported using the methods described in II-B and II-C: the hash-based address is computed on the FQDN, which is different, so the addresses will not interfere with each other (except for the address collision problem, see IV).
- One or Several Domains, Several Prefixes. Hosts of several domains are working on the same LAN but each domain has its own prefix, routing rules, etc. While the previous methods are still valid for the node resolution by remote nodes, it is not possible for nodes to self-assign their addresses. In this situation each node receives advertisements from the router/routers for all the prefixes and cannot autonomously choose the right one for its domain. In this case, the hash-based assignment can be implemented in the DHCPv6 server (see Fig.4). Each host is able to acquire its address using a standard stateful configuration interaction.

III. Limits

The hierarchical structures of fully qualified names and IP addresses are not necessarily related. Symbolic names are useful for humans to reach servers and services, whilst IP addresses are machine oriented representations used by routers to dispatch packets towards the right networking nodes. The methods introduced in this paper require direct mapping between a DNS domain and a prefix. In real applications, this is quite usual for small business firms, client nodes in computer labs, or office LANs. A company may have several networks, and then it would need to use several (sub) domains. e.g., the company `famouscorp.com` may have several IPv6 sub-nets. Hash configuration requires the subnets to be named as sub-domains, e.g., `lab.famouscorp.com`, `adm.famouscorp.com`, `londonoffice.famouscorp.com`, etc. In general, there is no problem assigning FQDN to clients, as client names are for internal use, for system and network administrators, so the more specific they are, the clearer they are. On the other hand, institutional servers (e.g., `www.famouscorp.com`) should not refer to the internal sub-netting structure. It is always possible to assign names to externally visible servers using DNS CNAME entries, e.g., define `www.famouscorp.com` as a CNAME of `www.londonoffice.famouscorp.com`. CNAME entries can provide network administrators with a means of defining all the symbolic names needed. Using the hash address assignment, the administrator just writes symbolic names. Thus, no IPv6 address typing is required, reducing the probability of configuration errors.

IV. Addresses' Birthdays

It is possible that multiple FQDNs generate the same hash code, so that they collide, pretending to use the same IPv6 address. The discussion is focussed on one network where all the hosts share the same networking prefix, as no collision can take place if the prefixes are different. This section shows that there is a very low probability of this event, so it can safely be ignored. Should it happen, it is possible to avoid it by changing one of the colliding names. If we consider that the hash keys follow a uniform distribution, the probability of collision becomes an instance of the Birthday Problem (also known as Birthday Paradox, see page 507 in [5]). The probability of

 ISBN: 978-1-61208-300-1

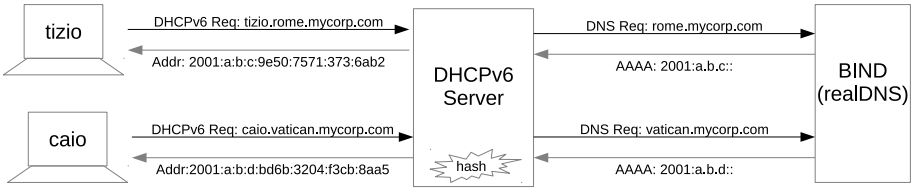

Fig. 4. Hash address assignment using a modified DHCPv6 server for multiple-prefixes LAN

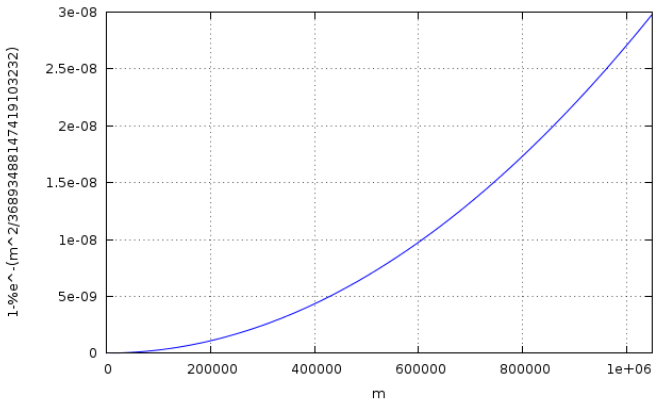

Fig. 5. Probability of address collision in a 64 bits hash

m nodes choosing the same random key, amongst n possible keys, is

$$Pr[(n,m)] = 1 - \frac{n!\binom{m}{n}}{m^n} \quad (1)$$

Using a Taylor series approximation (as $m/n << 1$) the probability can be estimated as follows:

$$Pr[(n,m)] \approx 1 - e^{-\frac{m^2}{2n}} \quad (2)$$

Figure 5 shows the probability function (2) plotted for up to 1Mi (i.e., 2^{20}) computers. The probability of two addresses colliding is less than one in 30 million for a LAN connecting more than 1 million hosts. In more realistic cases, network connecting less than one thousand nodes, the probability has the order of magnitude of one in $3 \cdot 10^{14}$.

Anyway, it is possible to implement methods in order to warn system administrators if such a collision should occur. Using the technique described in II-A, the address assignment script can take care of the collision detection. For II-C the updating function of the address cache for reverse resolution can reveal the collision problem and start some warning procedure.

V. Proof-of-concept implementation

Some software tools have been implemented to test the effectiveness of the approaches proposed in this paper. A first tool is a command to compute the address of a host. This command, named `hashaddr`, uses the MD5 algorithm [6] to compute the hash of the FQDN.

`hashaddr` takes two arguments: the first is a prefix (or a base address, as explained in the following) P and the second is the FQDN.

The final address is computed as follows:

$$hash = md5hash(FQDN) \quad (3)$$

$$address = P \oplus hi64bits(hash) \oplus lo64bits(hash) \quad (4)$$

where the function $hi64bits$ returns the 64 most significant bits, and $lo64bits$ returns the 64 least significant bits.

For example, a test running of `hashaddr` together with its output follows:

```
$ ./hashaddr 2001:a:b:c::  tizio.rome.mycorp.com
2001:a:b:c:9e50:7571:373:6ab2
```

The MD5 hash of `tizio.rome.mycorp.com` is `a2ea0c2518c2756b3cba79541bb11fd9`. The final address is computed as follows (the IPv6 encoding of 128 bit numbers is used for clarity):

$$\begin{aligned} addr \quad = \quad & 2001:a:b:c:: \oplus :: a2ea:0c25:18c2:756b \\ & \oplus :: 3cba:7954:1bb1:1fd9 \end{aligned}$$

`hashaddr` can be used in the scripts to compute the tables for the DHCPv6 and DNS servers as presented in II-A. These scripts must be run each time there is a change in the set of managed hosts to update the tables. No modifications are needed for DHCPv6 and DNS servers, because they simply use automatically generated tables instead of manually inserted data. Clearly the performance of the stateful auto-configuration and name resolution operations is not affected by the usage of hash computed addresses.

`hashaddr` uses `getaddrinfo` [7] to get the prefix (or base address), so it can be expressed in a symbolic way (using DNS resolution). For example, if the base address `2001:a:b:c::` for rome.mycorp.com has been manually added to the DNS table, this is the result:

```
$ ./hashaddr rome.mycorp.com  tizio.rome.mycorp.com
2001:a:b:c:9e50:7571:373:6ab2
```

In the examples, we have used a prefix as the second argument, i.e., the 64 least significant bits of the address were zeroes. When a general address is used instead, the 64 low order bits and the hash of the FQDN are combined by a XOR operation. In this case we name this *base address*, as it allows us to provide a specific mapping.

 ISBN: 978-1-61208-300-1

TABLE I
PROXY IMPLEMENTATION PERFORMANCE

Test #1: 100 DNS lookups

	average	standard deviation
DNS(no proxy)	35.3	5.22
Proxy	75.6	11.27

Test #2: `scp` of 1k file

	average	standard deviation
DNS(no proxy)	156,6	4.72
Proxy	158.9	5.89

This idea can be used to implement an extension of the method to preserve some address privacy. IPv6 architecture and philosophy does not include the idea of masqueraded addresses for clients (unique local addresses [8] are site-local and not routed to the Internet). If the hash-based address resolution of local clients and DNS resolution for base addresses are allowed only for local clients, it is harder for a remote attacker to guess the IP addresses. He cannot just compute the hash of some known FQDN. The knowledge of the entire base address is needed to succeed.

The proof-of-concept implementation for the method described in II-B, hash address self-assignment and automatic DNS forward resolution, uses a specific DNS proxy program: `hashdns` (see Fig. 6). This program takes three or more arguments. The first argument is the IP address the proxy DNS server will use; the second is the address of a real existing DNS, where the requests will be forwarded; the third and following arguments are a list of domains managed by the proxy.

```
./hashdns 2001:a:b:c::2 2001:a:b:c::3 rome.mycorp.com
```

All the DNS requests received by the proxy (at the address `2001:a:b:c::2`) get forwarded to the DNS server specified by the second argument (`2001:a:b:c::3`) except those asking for an AAAA record of a host of one of the managed domains. In this case (e.g., `tizio.rome.mycorp.com` in the example above) the proxy forwards the request for the domain (`rome.mycorp.com`) by stripping away the preceding part of the FQDN. When the real DNS server returns the address corresponding to the domain, `hashdns` performs the same process described for `hashaddr` to compute the hash-based address, using the address received by the real DNS server as its base address. The result is returned by the proxy as its reply to the original request.

Table I shows some of the proxy performance figures. The test environment is consists of a dual core 2Ghz CoreDuo2 processor, 3GiB RAM. The proxy is running on the same machine using the user mode stack LWIPv6 [9]. The first test evaluates the time needed for name resolution using a real DNS server and the proxy supporting hash-based addresses. The time in the table is in milliseconds for 100 calls of `getaddrinfo`. Although the overheads are high, the proxy takes about twice the time of the DNS, the second test shows that it is not appreciable for a normal networking usage. In fact, the second test measures the time needed to transfer one 1KB file by `scp` from the same machine: the address returned by the DNS or proxy resolution is one of the addresses of the same computer. If this is not the worst case for an `scp` transfer, it is an unfavorable situation: the transfer of a small file on a very fast line. The cost of name resolution would impact less on large files or slow lines. The computed overhead is less than 2%.

VI. RELATED WORK

Hash-based addresses are very useful when applied to the Internet of Threads. This new idea, presented in [10], changes the concept of communication endpoints in the Internet: each process can be provided with its own IPv6 address and can be a node of the network.

This change of perpective permits the development of many new services and a higher level of flexibility but, at the same time, increases the number of IP addresses to manage.

Cryptographically Generated Address (CGA) protocol, defined in [11], uses a one-way hash function to define the least-significant 64 bits of the IPv6 address. The meaning and the purpose of the operation is completely different from those defined in this paper: CGA computes a hash function on the public key of the sender to enforce a secure communication; hash-based address is computed on the FQDN of the host to ease the network management.

IPv6 privacy extension [2] generates the least-significant 64 bits of the IPv6 address in a random manner to preserve the privacy of the hardware controller MAC address. A new random address is generated if an address collision is detected. This method is for clients and does not provide methods to update a DNS server map.

VII. CONCLUSION AND FUTURE WORK

This paper has explained how to use hash-based IPv6 addresses to simplify the network deployment by system administrators. The proposed methods open a new perspective on IPv6 network configuration. The use of the MD5 algorithm as well as the proxy based implementation are just examples to support the effectiveness of these methods. It is possible to envision several further developments for this project. A non-exhaustive list of ideas by the author at the time of writing this paper follows:

- Per site defined hash function: each company, institution, etc., can design its own hash function. All the methods work provided that the same hash function is shared between the node, for address self assignment, and the DNS resolution.
- Native support of hash-based addresses in DNS servers. This enhancement would provide better performance to the name resolution process.
- DHCPv6 proxy for multiple domain support. An implementation of the method proposed in section II-D is still missing.
- It is possible to support the co-existence of static IPv6 addresses and hash-based ones. For example, the DNS proxy can forward the request for the entire FQDN to

 ISBN: 978-1-61208-300-1

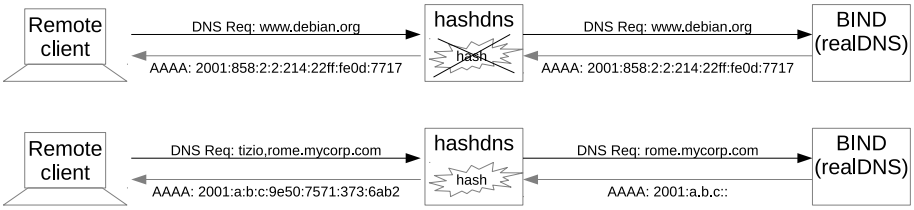

Fig. 6. hashaddr: DNS proxy proof-of-concept implementation

the real DNS server and perform the hash-based name resolution only when the FQDN resolution fails.

- It is common for companies to have different DNS servers to provide different views of their networking structure for local and external users. For example, customers should be able to access the company's web server, but it is useless (and potentially dangerous) to provide the name resolution of the accountancy office personal computer. An extension for the DNS proxy can provide filters to decide the visibility boundary of the hash-based address resolution for each domain, i.e., which source IP addresses are allowed to know the resolution for the hosts in each domain.

The source code to test the experiments presented in this paper can be downloaded from `svn://svn.code.sf.net/p/view-os/code/branches/hashaddrtest` and has been released under the GNU General Public License (GPL) v. 2 or newer. The programs are intended as just a proof-of-concept to show the effectiveness of the ideas introduced here.

ACKNOWLEDGMENT

I would like to thank the anonymous reviewers for their valuable suggestions and their comments.

REFERENCES

[1] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," RFC 4862 (Draft Standard), Internet Engineering Task Force, Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4862.txt

[2] T. Narten, R. Draves, and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," RFC 4941 (Draft Standard), Internet Engineering Task Force, Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4941.txt (Retrieved: June 15, 2013)

[3] M. Tortonesi and R. Davoli, "User untraceability in next-generation internet: a proposal," in *Proceeding of Communication and Computer Networks 2002 (CCN 2002)*, IASTED, Ed., November 2002, pp. 177 – 182.

[4] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," RFC 3315 (Proposed Standard), Internet Engineering Task Force, Jul. 2003, updated by RFCs 4361, 5494, 6221, 6422, 6644. [Online]. Available: http://www.ietf.org/rfc/rfc3315.txt (Retrieved: June 15, 2013)

[5] D. E. Knuth, *The art of computer programming, volume 3: sorting and searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1973.

[6] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321 (Informational), Internet Engineering Task Force, Apr. 1992, updated by RFC 6151. [Online]. Available: http://www.ietf.org/rfc/rfc1321.txt (Retrieved: June 15, 2013)

[7] R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens, "Basic Socket Interface Extensions for IPv6," RFC 3493 (Informational), Internet Engineering Task Force, Feb. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3493.txt (Retrieved: June 15, 2013)

[8] R. Hinden and B. Haberman, "Unique Local IPv6 Unicast Addresses," RFC 4193 (Proposed Standard), Internet Engineering Task Force, Oct. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4193.txt (Retrieved: June 15, 2013)

[9] R. Davoli and M.Goldweber, "Virtual square: Users, programmers and developers guide," 2011. [Online]. Available: http://www.cs.unibo.it/ renzo/virtualsquare/ (Retrieved: June 15, 2013)

[10] R. Davoli, "Internet of threads," in *Proc. of the The Eighth International Conference on Internet and Web Applications and Services, ICIW 2013. To appear*, 2013.

[11] T. Aura, "Cryptographically Generated Addresses (CGA)," RFC 3972 (Proposed Standard), Internet Engineering Task Force, Mar. 2005, updated by RFCs 4581, 4982. [Online]. Available: http://www.ietf.org/rfc/rfc3972.txt (Retrieved: June 15, 2013)

 ISBN: 978-1-61208-300-1