DHCPv6 for Data Centers

Felix Hamme

August 31, 2020

Bachelorarbeit

für die Prüfung zum Bachelor of Science

im Studiengang angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

bei der United Internet AG

zum Abgabedatum 31. August 2020

student	Felix Hamme
created during	June 2020 to August 2020
company	United Internet Sourcing & Apprenticeship GmbH
university	Baden-Württemberg Cooperative State University Karlsruhe
license of this work	Creative Commons Attribution 4.0 International License

"Every networking problem always takes longer to solve than it seems like it should."

fundamental networking truth number 9a, RFC 1925 [1]

Abstract

The IPv4 address exhaustion is the main motivation to migrate from IPv4 to the successor protocol IPv6. Like with IPv4, computers need to be supplied with addresses and configuration information about the network, to be able to communicate using IPv6.

The *United Internet AG* operates multiple data centers with a five digit number of servers, which all need to be supplied with the aforementioned information. The infrastructure to do that needs to be fast enough, highly available and extensible.

This bachelor thesis studies how available configuration mechanisms for IPv6 nodes (particularly SLAAC and DHCPv6) work. Several scenarios are analyzed in which multiple redundant stateful DHCPv6 servers that perform dynamic DNS updates can produce inconsistent information about the addresses in the network. A list of open source DHCPv6 server implementations is compiled. The existing DHCPv4 infrastructure of the *United Internet AG* is explained and a new infrastructure for configuring IPv6 nodes is proposed.

Zusammenfassung

Die IPv4-Adress-Knappheit ist der wichtigste Grund für eine Migration von IPv4 zum Nachfolger-Protokoll IPv6. Wie bei IPv4, müssen Computer mit Adressen und Konfigurations-Informationen bezüglich des Netzwerks ausgestattet werden, damit sie mittels IPv6 kommunizieren können.

Die *United Internet AG* betreibt mehrere Rechenzentren mit einer fünfstelligen Anzahl an Servern, die alle mit den zuvor genannten Informationen versorgt werden müssen. Die Infrastruktur zum Konfigurieren dieser Informationen muss schnell genug, hochverfügbar, und erweiterbar sein.

Diese Bachelorarbeit untersucht, wie Konfigurations-Mechanismen für IPv6-Knoten (insbesondere SLAAC und DHCPv6) funktionieren. Verschiedene Szenarien werden analysiert, in denen mehrere redundante, zustandsbehaftete DHCPv6-Server, die dynamische DNS-Updates machen, inkonsistente Informationen über die Adressen im Netzwerk produzieren können. Eine Liste mit Open-Source Implementationen von DHCPv6-Servern wird zusammengestellt. Die existierende DHCPv4-Infrastruktur der *United Internet AG* wird erklärt, und eine neue Infrastruktur zum Konfigurieren von IPv6-Knoten wird vorgeschlagen.

Contents

1 Introduction

The internet is a big thing. Figuratively and literally, as it is used by over 4.1 billion people in the world [2] and connects even more computers. It continues to grow.

But what *is* "the internet"? It makes exchanging data between many computers possible, via so-called *networks*. The *International Data Corporation* wrote this explanation [3]: "The global Internet is an amalgam of separate, but semiautonomous networks. Each network and server provider is an independent entity with its own policies, services, and customer targets. The binding element of the global Internet is that networks share a common IP addressing and global BGP routing framework that allows all networks to interconnect with each other directly or indirectly." All participants agree upon using a standard called *Internet Protocol* (IP) for exchanging data. Every possible target and source of data traffic needs an address, in order to track where the data is from and where it is meant to go.

The IP is standardized in the versions IPv4 [4] and IPv6 [5].[1] The former uses an address space of 32 bits, which allows to form about 4.3 billion distinct addresses. The amount of 'usable' addresses is fewer than that, due to reservations for special purposes [7]. Held against the 4.1 billion internet users, it is obvious that not everybody can get a unique address. Despite some workarounds, this is a serious problem and has been labeled 'IPv4 Address Exhaustion'. This problem is elaborated in section 3.1.

The *United Internet AG* operates a five digit number of computers in multiple data centers. This work analyzes how those computers are currently supplied with IPv4 addresses and related configuration information, and how this could be improved by migrating to the successor protocol IPv6. The *Dynamic Host Configuration Protocol* (DHCP) is a configuration mechanism standardized for IPv4 and IPv6 that can provide the addresses and configuration information (see [8], [9]). Hence, the title of this work is "DHCPv6 for Data Centers".

1.1 Practical Usecases Considered by This Thesis

A goal of this work is to investigate mechanisms to configure IPv6 nodes. Those nodes may be divided into two groups, which are described in the following sections. Nodes with other requirements may be added in the future, but they are out of scope for this work.

1.1.1 PXE-DHCP

This group of nodes is expected to boot over the network using the *Preboot Execution Environment* protocol (PXE [10]). Booting over the network means to download an operating system image from some other node. To reach that other node and the image, the booting node needs to acquire a usable IP addresses, the address of the node where to download the image from, the name of the image and usually more information (e.g. addresses of recursive DNS servers).

A network boot begins with a network interface card making a DHCP request. Using the received information, a small image is fetched over the network and booted. This booted image

[1] The version numbers 1, 2, 3 and 5 were used by standardized protocols, which now all have historic status. The version numbers are still reserved to avoid confusion. (See [6]).

can continue to use the information, especially the IP address obtained previously via DHCP. The image is usually run to fetch and boot the actually desired, bigger operating system image. This operating system typically gets its network configuration independently of the already executed DHCP message exchange. The operating system can make its own DHCP request, but it could also use any other configuration mechanism.

1.1.2 RMC-DHCP

The nodes in this group are mostly *remote management controllers* (RMCs) of physical servers, hence the name of this group. They need usable IP addresses and, as the major difference to PXE-DHCP nodes, DNS records that point to that addresses. The DNS records allow administrators and appropriate software to access the RMCs.

More precisely, each assigned address should get an `A`- (IPv4) or `AAAA`-record (IPv6) and a `PTR` record to an appropriate FQDN.[2] A `CNAME` record should be added if possible to point to this FQDN from another FQDN that is formed of the hostname of the configured node. Those records should be available soon after the address got assigned. FQDNs should not point to the wrong node, to reduce the risk of mistakenly instructing the wrong server (e.g. shutting a machine down that needs to stay online).

The RMC-DHCP nodes may but usually do not need to use network boot information.

Currently, a DHCPv4 server leases addresses to those RMCs and triggers, using a chain of company-internally developed software, updates in the authoritative DNS servers.

There is a diversity of manufacturers, models and firmware versions of RMCs in use. Most of the RMCs are *Dell iDRACs*.

There is currently no comprehensive database that keeps track of models and firmware versions of the RMCs owned by the company. However, the company uses an internally developed asset management software, which keeps records of all servers. This includes the server type and a URL to the RMC.

URLs of iDRACs are formed by including the Dell Service Tag in a well-known FQDN. URLs of *HP iLOs* and *IBM Integrated Management Modules* (IMM) are likewise formed using other well-known FQDNs. To itemize the number of iDRACs based on their major firmware version, the matching server model codes are retrieved from the asset management software. Those codes indicate the major version of the iDRACs: *Gen12* Dell servers have version 7, *Gen13* have version 8 and *Gen14* and *Gen15* have version 9.

To simplify the scope of this work, the group of expected RMCs is reduced to iDRACs of version 7, 8 and 9.

1.2 The Scope and Aim of This Thesis

The motivation for this work is to improve the configuration of IP addresses and related configuration information for the nodes categorized in section 1.1 into PXE-DHCP and RMC-DHCP.

This thesis first explains the relevant technologies in section 2.

[2] *FQDN* stands for Fully Qualified Domain Name. For this and the DNS record types, see [11].

iDRAC version	number of servers
9	5726
8	12478
7	5937
total	24141

Tab. 1: The count of *Dell iDRACs* by version that are owned by the *United Internet AG* as of 2020-07-28.

Section 3 evaluates advantages and disadvantages of a migration from IPv4 to IPv6 in general and with respect to the considered usecases. The migration to networks that use only IPv6 was expected to be the desirable, which is why the remaining work focuses on how an appropriate configuration service architecture can be built.

It was expected that a stateful DHCPv6 server will be necessary. To fulfill company requirements, this service has to be build resilient. Section 4 analyzes what state is maintained by a stateful DHCPv6 server, what issues might occur if that state is not synchronized across all server instances and how the state could be synchronized.

Section 5 summarizes the required features of DHCPv6 server implementations. A selection of DHCPv6 server implementations that run on Linux and could be suitable with respect to the aforementioned requirements is presented.

Next, the current infrastructure of the *United Internet AG* is analyzed in section 6. The requirements get specified further and a service architecture for IPv6 is proposed.

Finally, section 7 summarizes the findings, discusses the methodology and recommends future work.

7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link
1	Physical

Tab. 2: The Internet Protocol is by far not the only standard used for data exchange between computers. IP is transferred in the payload of a lower layer protocol and has itself a payload in which it carries higher layer protocols. For each layer, there is a variety of protocols to choose from. Lower layer protocols should not depend on higher layer protocols. The layers are standardized by the *OSI model* [13] and shown here. Examples for network layer protocols are IPv4 [4] and IPv6 [5]. Examples for transport layer protocols are UDP [14] and TCP [15].

2 Background: How the Internet Works

This section provides the relevant background on network protocols. Ordered by the OSI model layers (see table 2), the protocols Ethernet (section 2.1), IPv4 (section 2.2), IPv6 (section 2.3), ICMPv6 (section 2.4) and UDP (section 2.5) are presented.

Section 2.6 gives an overview of possible configuration mechanisms for IPv4 and especially IPv6 nodes. The configuration mechanisms for IPv6 nodes SLAAC and DHCPv6 are explained in section 2.7 and section 2.8. Some terms and protocols for operating multiple redundant servers are introduced in section 2.9.

Unless otherwise mentioned, IPv6 terminology is used (see especially the "Terminology" sections of [5], [9], [12]).

2.1 Layer 1 and 2: Ethernet

IPv6 sends its messages over so-called *links*. The IPv6 standard [5] defines a link to be "a communication facility or medium over which nodes can communicate at the link layer, i.e., the layer immediately below IPv6. Examples are Ethernets [...]". A link can have any number of interfaces attached to it. Each interface belongs to a node and has a link-layer address. IPv6 packets are transmitted in the payload of the link-layer protocol of a link, from an interface to another, each identified by a link-layer address. This terminology is illustrated in figure 1.

The *Institute of Electrical and Electronics Engineers, Inc* (IEEE) standardized the network protocol "Ethernet" [16], which implements the *physical* and *data link* OSI model layers (see table 2). Despite the existence of alternative implementations, this work assumes Ethernet to be practically the only implementation, to reflect the considered networks of the *United Internet AG*.

The interaction between IPv6 and link-layer implementations is subject to implementation-dependent standards. RFC2464 [17] describes how IPv6 can be transported over Ethernet.

MAC addresses are a type of link-layer addresses. Every Ethernet network interface has a unique MAC address. In case of physical network interfaces, those are assigned by the hardware

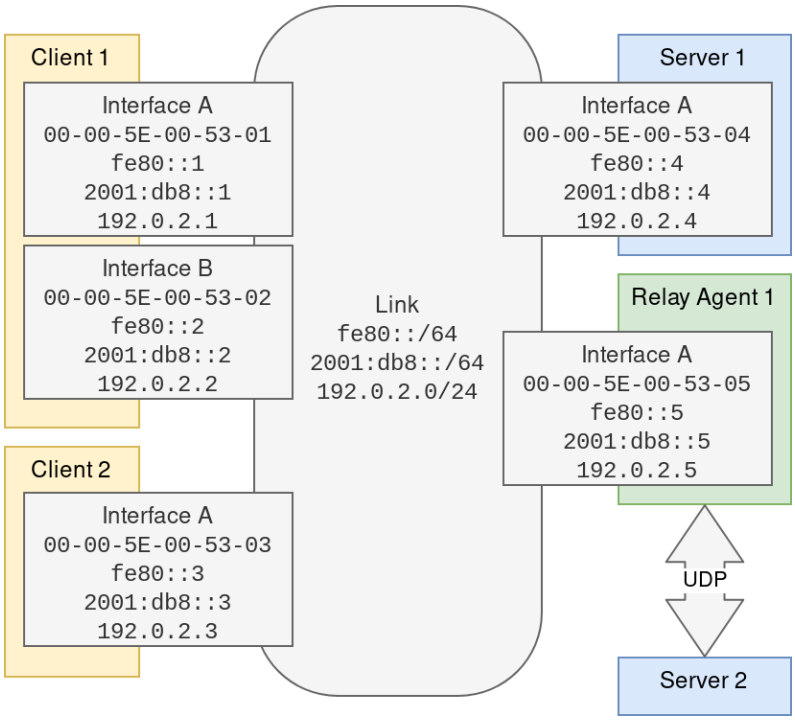


Fig. 1: This example shows how IP *nodes* (colored boxes) are connected to other nodes (e.g. DHCP servers). Each interface connects a node to a link and has one link-layer address and several IP addresses. In each of these examples, the first address is an Ethernet, the second an IPv6 of link-local scope, the third an IPv6 of global scope and the fourth an IPv4 address. The IP addresses are taken from the address prefixes that are *on-link*, which are listed here below "Link". The communication between *Relay Agent 1* and *Server 2* is abstracted to the layer 4 protocol UDP and could be implemented using one or a chain of links.

vendor.

Volume "Section One", section 3.2.3 of the standard [16] defines the format of *media access control addresses* (MAC addresses) to be 48 bit long. The first two bits in canonical bit order have special meaning. The first denotes whether the address refers to a single (value `0`) or multiple (value `1`) targets. The second bit indicates whether the address was assigned by a global authority (value `0`).

2.2 Layer 3: IPv4

The Internet Protocol version 4 (IPv4) was standardized 1981 in RFC 791 [4]. It implements the *network* layer of the *OSI model* by sending "packets" that consist of a header and a payload to other nodes. The packets are transmitted in the payload of link-layer protocols, like Ethernet. IPv4 (like all versions of IP) enables to transmit data between hosts that are not at the same link, but are connected via a chain of links. So-called *routers* transfer packets between links and thereby interconnect the networks, hence the name "Internet".

The addresses used to define the source and target of an IPv4 packet ("IPv4 address") are 32 bit long. Its representation in text has not been specified in the IPv4 standard. However, other documents specify a format. RFC 952 [18], which was published about four years after

prefix	purpose
`0.0.0.0/8`	"this host on this network"
`10.0.0.0/8`	private addresses
`127.0.0.0/8`	loopback (same host)
`169.254.0.0/16`	link-local addresses
`172.16.0.0/12`	private addresses
`192.168.0.0/16`	private addresses
`240.0.0.0/4`	multicast addresses

Tab. 3: Some reserved IPv4 prefixes, see RFC 6890 [7]. The listed prefixes account for 7.45 % of the IPv4 address space.

RFC 791 defines the format that is common nowadays. The 32 bit are split into four bytes which are each written in decimal and then joined together using dots ("."), like such:

$$\begin{array}{c} \texttt{01111111 00000000 00000000 00000001} \\ \downarrow \\ \texttt{127.0.0.1} \end{array} \tag{1}$$

Each network interface can usually be assigned at most one IPv4 address. All those IPv4 addresses at the same link are assigned from the same prefix. A prefix is a range of addresses that start with a common sequence of bits. A prefix is represented in text by adding as much zeroes to the bit sequence as needed to from an address, representing that in text, appending a slash ("/") and appending the length of the bit sequence (without added zeroes). For example, `10.0.0.0/16` describes addresses from `10.0.0.0` to `10.0.255.255`. (See [19]).

The first and last IPv4 address of a prefix at a link can not be assigned to interfaces, because they are reserved to respectively identify the network and broadcast packets to all nodes at the link. There are several prefixes which are reserved for special purposes, some of which are listed in table 3.

2.2.1 Mapping IPv4 Addresses to Link-Layer Addresses

Whenever an IPv4 packet is sent to a different node, it needs to be encapsulated in the link-layer protocol payload. This requires to know the link-layer address of the next hop (a node at the same link, the destination node or a router).

The *Address Resolution Protocol* (ARP, see RFC 826 [20]) can be used to query a link-layer address using an IPv4 address. It is transmitted in the payload of the link-layer protocol and can be used for other protocols than IPv4. (IPv6 uses the *Neighbor Discovery Protocol* instead of ARP, see section 2.4.1). To find out a link-layer address, a node broadcasts (see figure 2) an ARP message to all nodes at the link. If a node sees such a message and has the IPv4 address configured on its interface, it replies with a message containing the link-layer address.

2.2.2 Predecessors of DHCPv4

The *Reverse Address Resolution Protocol* (RARP, see RFC 903 [21]) servers, as the name suggests, the reverse purpose of ARP: It translates link-layer addresses into IPv4 addresses. This can be

used by nodes that have no IPv4 address yet and want to find out which they can use.

This configuration mechanism was later obsoleted by the *Bootstrap Protocol* (BOOTP, see [22]–[24]). Using the BOOTP, a client can request configuration from a server. Among IPv4 addresses, the BOOTP can also configure arbitrary other information, like router IPv4 addresses or addresses of recursive DNS nameservers. The Bootstrap Protocol got its name because it enables nodes to enough information to fetch a file from another node and use that as boot image.

Because the BOOTP is transported in the payload of UDP (see section 2.5), which in turn is (in this case) transported in IPv4, a client initially needs to fill addresses in IPv4 packet headers before knowing them. This is solved by using the unspecified address `0.0.0.0` as source and broadcasting the message to the well-known address `255.255.255.255`. The reply to that is either sent to the new address of the client, or to the mentioned broadcast address. In both cases, the client needs to receive that message before having an IPv4 address assigned to the interface.

Because the broadcasts reach only nodes at the same link, the BOOTP introduced "forwarding agents" which are also known as "relay agents". Those are placed at the link of the client and pass the BOOTP messages along between clients and servers.

2.2.3 DCHPv4

The *Dynamic Host Configuration Protocol* (DHCP, see RFC2131 [8]), named *DHCPv4* here for disambiguation, is a protocol that provides IPv4 nodes with IPv4 addresses and other configuration information. It was designed as a mechanism that eliminates the need for manual and individual configuration of clients. DHCPv4 was developed to be interoperable with the BOOTP.

Like BOOTP, DHCPv4 offers assignment of IPv4 addresses and other configuration information to clients using servers. A DHCPv4 server can assign addresses to clients permanently or for a limited time ("lifetime"). The latter is motivated by networks in which addresses are scarce.

BOOTP identifies clients based on their link-layer address, which is still common practice with DHCPv4, but the latter introduces an opaque value that can be provided by a client to identify itself. Nonetheless, DHCPv4 can assign exactly one IPv4 address per link-layer address.

To initially obtain configuration information, the following four-way message exchange is used. This needs to be repeated for each interface to which the client wants to assign an address. The communication prior to assigning an address to a client takes place like with BOOTP as described above, if needed via a relay agent.

1. The client locates servers by broadcasting a `DHCPDISCOVER` message to all nodes at the link using the well-known IPv4 address `255.255.255.255`. This broadcast address is used for all broadcast messages in DHCPv4.

2. Any servers can offer configuration information by sending a `DHCPOFFER` message to the client. A server ought to probe (e.g. using an ICMPv4 Echo Request message) whether the address it is offering is currently not in use.

3. The client selects a server and declines offers of the others by broadcasting a `DHCPREQUEST` message. The servers know which was selected because the client includes the server's IPv4 address in the message.

4. The server confirms the configuration information using a `DHCPACK` message. If during the time of the message exchange the server became unable to provide the configuration information as written in the `DHCPREQUEST` message, it refuses that by broadcasting a `DHCPNAK` message.

IPv4 has no mechanism to prevent duplicate addresses, so a DHCPv4 server has the responsibility to ensure the addresses in the network are unique. RFC 2131 recommends clients to check the configuration information they received. If a client detects at this occasion (e.g. using ARP) that the address it got assigned is already in use, it must notify the server using a `DHCPDECLINE` message and restart the configuration process.

A client can request to extend the lifetime of an address by first *renewing* and then *rebinding*. First, a client attempts to renew its address by unicasting a `DHCPREQUEST` message to the server from which it got the address. The server can grant the the lifetime extension by sending a `DHCPACK` message to the client. If the server does not reply after some time (perhaps because it was shut down), the client proceeds to *rebinding*. When doing so, it broadcasts the `DHCPREQUEST` message in the hope that another server can extend the lifetime. This is intended to work with multiple DHCPv4 servers that synchronize their information about the addresses they have assigned.

If a client receives no answer to a message, the client retransmits the message. RFC 2131 requires an "randomized exponential backoff algorithm" to determine the delay between such retransmissions, but leaves further details up to the implementation of the client.

A client can use a `DHCPRELEASE` message to inform a server that it is no longer using an address and the server may reallocate it. If a client wants no address but other configuration information, it can query a server using a `DHCPINFORM` message.

2.3 Layer 3: IPv6

The *Internet Protocol version 6* (IPv6) is defined 1998 in RFC 2460 [25] which was replaced in 2017 by RFC 8200 [5]. It implements the network layer of the OSI model (see table 2), like IPv4. The following sections describe this protocol and those built upon it. Within this section, IPv6 addresses are described. Much of the behavior of IPv6 nodes is implemented using the *ICMPv6* protocol, which is described in section 2.4. The *Stateless Address Auto Configuration* is also part of IPv6 and explained in section 2.7.

2.3.1 Addresses and Prefixes

An IPv6 *address* is a 128 bit long number. In text it can be represented in different forms, as defined in RFC 4291 [26]:

- In the "preferred form" the IPv6 address is split up into eight 16 bit long fields which are written in hexadecimal and joined by colons. Each field must have at least one digit,

leading zeroes may be omitted.
Example: `2001:db8:0:0:0:1234:c0a8:1`

- The "compressed form" is a variation of the preferred form in which consecutive fields with the value 0 are abbreviated using a double colon ("::"). To avoid ambiguity, this must be done no more than once per address.
  Example: `2001:db8::1234:c0a8:1`

- To ease migration from IPv4, the last two fields (32 bit) may be written in the syntax of an IPv4 address:
  Example: `2001:db8::1234:192.168.0.1`

The examples above all refer to the same address.

An IPv6 *prefix* is an IPv6 address together with an integer between 0 and 128. In textual representation, address and integer are joined by a slash ("/"), e.g.: `2001:db8::/64` (see [26]). This is similar to the CIDR notation of IPv4 subnets (see [19]), because it has a similar meaning.

The prefix specifies a range of addresses by defining a common string of bits they start with. The length of that bit string is given by the integer. The actual bits are taken from the address part of the prefix.

The notation may either refer to a specific address within a prefix or to a prefix as such. In the latter case, the "unspecified" bits after the bit string should be set to 0. Such addresses are also reserved for anycasting traffic to a router on the link (see [26]).

2.3.2 Address Scopes

RFC 4007 [27] defines: "Every IPv6 address other than the unspecified address has a specific scope; that is, a topological span within which the address may be used as a unique identifier for an interface or set of interfaces. The scope of an address is encoded as part of the address [...]."

This means, that the scope sets a limit to where an IPv6 packet may be routed. The scope is identified by standardized prefixes. A scope is a general concept, instances of a scope are called *zones*. Since a node might be connected to multiple different zones of the same scope, there is a need to disambiguate addresses. RFC 4007 defines that an implementation-dependent string that identifies a zone should be written behind an address, delimited by a percent ("%") character. Together with a prefix length, an example may look like this: `2001:db8::%1/64`.

Scopes can be sorted by the topological size they cover. The smallest scope is called *interface-local* because it contains only a single interface. Addresses of that scope might be useful for node-internal traffic. The *link-local* scope comprises all nodes that can reach each other on the link layer, meaning routing is prohibited. The *global* scope includes the whole world. The single zone with global scope is the entire internet. The boundaries of zones of other scopes have to be chosen by network administrators.

2.3.3 Delivery Schemes: {Uni,Any,Multi,Broad}cast

Usually an IPv6 address identifies a specific interface on a specific host. But this is not always the case – there are multiple delivery schemes. Among them are the ones depicted and described

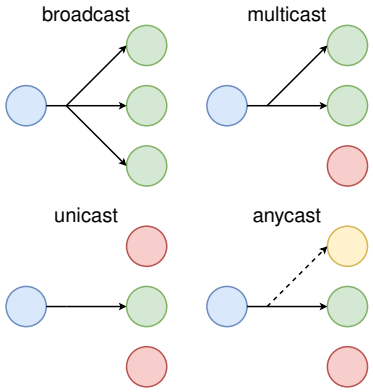


Fig. 2: Different delivery schemes govern where traffic is directed. With unicast, the destination is always exactly one target. With broadcast, traffic goes to all targets (within a given scope). Multicast traffic is routed to a specific set of targets. Anycast is like unicast, but the single target may change at the behest of the responsible routing protocol.

in figure 2.

IPv4 has a dedicated broadcast address in each subnet [28]. In IPv6, there is no such broadcast. Instead, there are standardized multicast addresses which are bound to scopes, so that there is no need for allocating multicast or broadcast addresses per network.

2.4 Layer 3: ICMPv6

The *Internet Control Message Protocol for IPv6* (ICMPv6) [29] is intertwined with IPv6: It is transported in the payload of IPv6 and IPv6 cannot function properly without it. ICMPv6 serves similar purposes like ICMPv4 [30], ARP [20], RARP [21] and IGMP [31] together.

ICMPv6 messages are divided into error messages and information messages. The former are to inform other nodes about errors while processing IPv6 packets. The information messages serve multiple purposes, which are not all relevant for this work. In the following, *Neighbor Discovery*, *Multicast Listener Discovery* and *Stateless Address Autoconfiguration* will be explained, which are all protocols based on ICMPv6 information messages.

2.4.1 Neighbor Discovery Protocol (ND)

Two nodes (hosts or routers) that are connected to the same link, are called neighbors. In order to use a link, a node needs to become acquainted with its neighbors. The *Neighbor Discovery* (ND) protocol [32], which is part of ICMPv6, solves this problem. More precisely, it offers solutions for the following problems:

address autoconfiguration	What addresses may a node choose?
address resolution	What is the link-layer address of an on-link IP address?
duplicate address detection (DAD)	Is a given on-link address already in use in this zone?
neighbor unreachability detection	Are the neighbors still reachable?
next-hop determination	Where should traffic be sent?
parameter discovery	What parameters should be known about the link (e.g. MTU) and the IP-layer (e.g. hop limit)?
prefix discovery	Which prefixes are on-link? (Meaning which addresses are reachable without leaving the link?)
redirect	Is there a better first hop?
router discovery	Which neighbors route traffic to what other links?

The address autoconfiguration is for this work the most relevant problem to look at. It aims to let nodes configure their addresses on their own. This involves almost all of the other problems. To solve this, the ND offers *Stateless Address Autoconfiguration* (SLAAC), which will be investigated in section 2.7.

The ND facilitates its solutions mostly through these four different ICMPv6 packet types:

- The *Neighbor Solicitation* is a query to trigger *Neighbor Advertisements*. It is used for address resolution, DAD and neighbor unreachability detection.
- The *Neighbor Advertisement* is sent as a reply to a Neighbor Solicitation. The message states for an IP address, that it is reachable and at which link-layer address it is reachable. As a performance optimization, a node may send Neighbor Advertisements without being asked in case its link-layer address changes.
- The *Router Solicitation* is a request for *Router Advertisements*.
- The *Router Advertisement* can only be sent by routers. It implies that the sending node is a router and delivers information about what prefixes are on-link, what prefixes the router will route and more. This message conveys whether SLAAC may be used and whether DHCPv6 is available.

Using *Neighbor Solicitation* and *Neighbor Advertisement* messages, a node can perform address resolution. With IPv4, this has been done using ARP [20]. The same can be said for inverse address resolution and RARP [21].

Link-Scope Addresses: Avoiding Dependencies on Link-Layer Protocols In contrast to IPv4, IPv6 aims to be more independent of the link-layer it is built upon. (IPv4 has a quite strong dependency on Ethernet.) The *Neighbor Discovery* (ND) protocol (see section 2.4.1) relies on multicast at the link-layer, which limits the types of links with which ND is compatible. However, there are standards that define ND even for such links (e.g. for cellular links [33]).

To loosen the dependency on link-layer addresses, IPv6 introduces link-scope addresses. These addresses are "normal" IPv6 addresses, but they are only usable and unique within the boundaries of a link. Hence they are somewhat like link-layer addresses but at the IP-layer. There are unicast and multicast link-scope addresses. Every node generates a link-scope unicast address per interface once it becomes online. This address is often derived from the corresponding link-layer address (e.g. using EUI-64). Before using the generated address, the node assures its uniqueness using *Duplicate Address Detection* (DAD).[3]

Duplicate Address Detection (DAD) To avoid using addresses that are already in use, the ND comes with an algorithm called *Duplicate Address Detection* (DAD), which is specified in [12]. This is performed every time before assigning an address to an interface, regardless of the mechanism that triggered the assignment (SLAAC, DHCPv6, ...).[4]

An address whose uniqueness is to be determined is called a *tentative* address. Usage of tentative addresses is limited to what is necessary for DAD. There are two cases in which the lack of uniqueness will be detected:

- The node sends a Neighbor Solicitation with its tentative address as target. If another node responds with a Neighbor Advertisement, the address is already in use. To increase certainty, the transmission of Neighbor Solicitations is repeated. The number of transmissions and the time to wait after them can be configured using variables.[5]
- There could be another node which simultaneously performs DAD for the same tentative address. In this case, the node would receive a Neighbor Solicitation that is not from itself, originates from the unspecified address and has the tentative address as target. This indicates that the address is not unique.

To avoid congestion, the first MLD message and Neighbor Solicitation is delayed by a random time.

If a duplicate address is detected and the tentative address has been generated from the link-layer address, the chances are that the link-layer address is not unique. It is recommended to stop using the interface at all.

2.4.2 Multicast Listener Discovery (MLD)

IPv6 makes heavy use of multicast (e.g. for ND [32] and DHCPv6 [9]). Nodes that want to receive traffic for a specific multicast address are called *listeners* [34]. This section explains how multicast traffic is sent to listeners.

Within a link, an IPv6 multicast address will be mapped to a link-layer multicast address. The link-layer protocol is then responsible for sending the traffic to at least all listeners. (The ND protocol assumes that the link-layer supports multicast. If not, other solutions may be

[3] If DAD fails, the node may try a different address. If the failure suggests that the link-layer address is not unique, the node should abort using that interface.

[4] There are two exceptions where DAD will not be done: When DAD is disabled by a per-interface variable, and when the address to be assigned is an anycast address.

[5] The protocol default is a single Neighbor Solicitation and 1 second waiting after transmission.

defined. [32]) For example, the link-layer protocol Ethernet constructs Ethernet-addresses from IPv6 multicast addresses by appending the last 32 bit of the IPv6 address to the well-known prefix `0x3333` (e.g. `ff02::1:2` becomes `33:33:00:01:00:02`) [17].

For IPv6 multicast addresses with a scope greater than link-local, delegating multicasting to the link-layer is not sufficient, because it could be necessary to route the multicast traffic across multiple links. If a router knows at which links the listeners are, it can send the traffic there using the link-layer multicast mentioned before. Hence, routers need a mechanism to find out on which links there are listeners for which IPv6 multicast addresses.

For IPv4, this problem has been solved with the *Internet Group Management Protocol* (IGMP) [31]. For IPv6, a protocol called *Multicast Listener Discovery* (MLD) has been defined as a part of ICMPv6 (whose implementation is mandatory).

Currently, the MLD protocol is specified in two versions: MLDv1 [34] and MLDv2 [35]. Both versions serve the purpose of providing routers with the information whether there are any listeners for a given IPv6 multicast address present on a given link. Neither the number of listeners nor their unicast addresses is conveyed, just the presence of at least one listener.

MLDv1 does that by using three types of messages:

- A *Report* message is sent by a node to indicate that it wants to receive traffic for a specific multicast address.
- A *Done* message is sent by a listener to express it no longer wants to receive traffic for a specific multicast address. There may be other listeners that still want to receive traffic for that multicast address, so the routers validate the absence of listeners before stopping routing traffic to for the multicast address to the link.
- A *Query* message is sent by a router to trigger Report messages. To avoid unnecessary traffic, only the router with the lowest link-local IPv6 address sends Query messages.

MLDv2 is interoperable with MLDv1 and extends its functionality by filtering multicast traffic based on its source address. In a MLDv2 Report message, a listener can whitelist or blacklist source addresses from which it wants to receive traffic for a specific multicast address.

2.5 Layer 4: UDP

The User Datagram Protocol (UDP) is mentioned here, because DHCPv6 builds upon it. It is fairly simple: Its definition, RFC 768 [14], is only three pages long.

The *header* together with an arbitrary payload is called a *datagram* and is carried in the payload of IPv4 or IPv6. The header contains a source and destination address, which are called ports, the length of the datagram and a checksum. Source port and destination port are each 16 bit long and are meant to identify individual applications on a single node.

The checksum incorporates the datagram and parts of the IP header. IPv4 already has a checksum, so the UDP checksum is optional when using UDP over IPv4. Because IPv6 has shifted the responsibility of data integrity away from itself, the UDP checksum has become mandatory for UDP over IPv6 [5].

2.6 Overview of Configuration Mechanisms for IP Nodes

IPv6 nodes need at least a link-local address to communicate with other nodes. To communicate with nodes at other links, they need at least one address with a greater scope. IPv4 nodes require a single IPv4 address to communicate with other nodes. If the scope of that address is not sufficient, this is usually solved with NAT (see section 3.1.1).

To resolve FQDNs, IPv6 nodes as well as IPv4 nodes need to know addresses of recursive DNS servers (RDNSS), commonly in conjunction with a list of zones to append hostnames to (DNSSL). This information is here named *DNS configuration*. Often, even more information is required (e.g. network boot information), which is vaguely summarized by RFC 8415 [9] as "other configuration information". It covers a variety of information and is designed to be extensible, which justifies the nebulous term. The *DNS configuration* is part of the *other configuration information* but can be treated separately.

There are several configuration mechanisms to tell a node all that information:

- It is always possible to enter the information by hand or using a custom automation. This is called "manual configuration". It might be appropriate where no dependency on the other mechanisms is desired, for example for an interface of a DHCPv6 server. RFC 2322 [36] specifies a protocol that can centrally (statefully) assign configuration to arbitrary IP nodes by using clothes-pegs and humans as transport mechanism. In most cases, another mechanism is preferred.

- For IPv4 nodes, DHCPv4 is a mechanism to configure the mentioned information. It is a server-client protocol that is built on UDP. The DHCPv4 server is stateful, because it tracks the leased addresses. Besides exactly one IPv4 address per link-layer address, it can also supply other configuration information.

For IPv6, there are two configuration mechanisms: SLAAC and DHCPv6. Either one of them or both together may be used.

The distinction between *stateful* and *stateless* operation is based on whether or not a single service needs to "maintain any dynamic state for individual clients", as clarified by RFC 3736 [37]. SLAAC is always stateless. When DHCPv6 is used for IPv6 address and/or IPv6 prefix allocation, it has the sole authority on that and therefore needs to maintain such an individual state, which makes it stateful. However this feature is not compulsory, which enables stateless operation of DHCPv6.

In either case ICMPv6 Router Advertisements are used to locate routers.

- When doing *stateful* configuration, DHCPv6 is used "just like" DHCPv4. The DHCPv6 server leases addresses to its clients and provides other configuration information.

- When doing *stateless* configuration, SLAAC is used. This implies that each node chooses its address on its own. The DNS configuration can be provided in Router Advertisements [38]. They can also hint that stateless DHCPv6 is available. For handling conflicting configuration information, see the following bullet point. Providing stateless DHCPv6 is optional.

- *Stateful* (DHCPv6) *and stateless* (SLAAC) configuration may be provided simultaneously. This applies to all configurable information, including addresses. In this case, hosts accept the union of all available information. The information should be consistent, but may differ. To solve a conflict, a host should prefer more recent information and may prefer information that was obtained from a more secure source. (see [12])

Table 4 provides an overview over the configuration mechanisms. In the following, SLAAC (section 2.7) and DHCPv6 (section 2.8) will be explained in more detail.

2.6.1 Separation of Concerns in IPv6

The DNS configuration and other configuration information does not have to be known to use IP. However, such information is usually of interest shortly after the IP connectivity was established. Especially the DNS configuration is practically an inevitable requirement for IP nodes (see [38]).

SLAAC can supply all necessary information for the IP layer, without "irrelevant" information overhead. Stateless DHCPv6 can be added if needed to provide configuration beyond that. This achieves the clearest separation of concerns using the presented configuration mechanisms.

The DNS configuration is mentioned explicitly, because it is so important, that SLAAC was extended to provide it, as section 2.7.3 describes. On nodes that also operate IPv4 using DHCPv6 it can be sufficient to solely use SLAAC and do DNS queries using IPv4 and the recursive DNS servers learned from DHCPv4.

Stateful DHCPv6 extends stateless DHCPv6 by address assignment and prefix delegation. The former can also be done using SLAAC, the latter not. DHCPv6 creates the impression to be a standalone configuration mechanism, but it depends even in stateful operation on ICMPv6 features like SLAAC for link-local address generation and Router Advertisements for router discovery.

In contrast, DHCPv4 violates the boundaries of the OSI model layers, because it defines special behavior at the network layer (e.g. the all-zero IPv4 address), whilst building upon a transport layer protocol. The DHCPv4 standard defines in section 2 of RFC 2131 [8]: "In the case of a client using DHCP for initial configuration (before the client's TCP/IP software has been completely configured), DHCP requires creative use of the client's TCP/IP software and liberal interpretation of RFC 1122."

2.7 Stateless Address Auto Configuration (SLAAC)

The protocol *Stateless Address Auto Configuration* (SLAAC) is defined by RFC 4862 [12] and part of ICMPv6. It supplies nodes with IPv6 addresses without the need for a central, stateful service.

As noted by RFC 4862 the design goals were:

1. There should be no need to configure a node manually before connecting it to the network.
2. Using the network should not require a central configuration service (DHCP). This should apply to single links as well as to networks comprised of multiple links.

	IPv4	IPv6		
	stateful	stateful	stateless	
	DHCPv4	stateful DHCPv6	ICMPv6	stateless DHCPv6
addresses for assignment	yes	yes	yes (SLAAC)	no
delegated prefixes	no	yes	no	no
router addresses	yes	no	yes (RA)	no
DNS configuration	yes	yes	yes (RA)	yes
other configuration information	yes	yes	no	yes

Tab. 4: For each configuration mechanism it is noted which information can be supplied. Each information may also be supplied via manual configuration. For more details, see section 2.6.

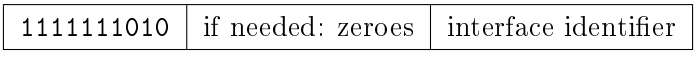
3. It should be possible to gracefully change addresses. Gracefully means, that there is a timespan where both old and new addresses are valid.

To fulfill the first goal, SLAAC was built as an integral part of IPv6, its implementation is mandatory. Furthermore, it has to be enabled by default. The default configuration of the protocol is intended to be sufficient to establish connectivity. By providing a way to generate addresses in a stateless manner (no single service assigns single addresses), the second goal is achieved. SLAAC introduces lifetimes of addresses that govern the usage of them, as described in figure 3. The usage of ICMPv6 *Router Advertisements* to set and update lifetimes of generated addresses accomplishes the third goal.

The generation of link-local addresses and global addresses differs, as described in the following sections. SLAAC always generates a link-local address per interface. The generation of global addresses is only possible if routers send appropriate messages. Those messages are also used to indicate the presence of the routers and optionally, to provide a list DNS configuration.

2.7.1 Generation of Link-Local Addresses

SLAAC generates a link-local scope address for each interface of a node by combining the well-known prefix `fe80::/10` with an interface identifier. The length of the interface identifier is subject to link-specific standards. The interface identifier is written in the very right bits of the IPv6 address Bits that are not covered by this or the well-known prefix are set to zero. Such a link-local address will look like this:



The prefix length that is used for the link is changed as appropriate with respect to the interface identifier length. In the case of Ethernet links, RFC 2464 [17] specifies how a 64 bit long interface identifier is generated from a 48 bit long MAC address. The according prefix is `fe80::/64`, because 128 bit minus the interface identifier length is 64 bit.

Link-local addresses get infinite lifetimes.

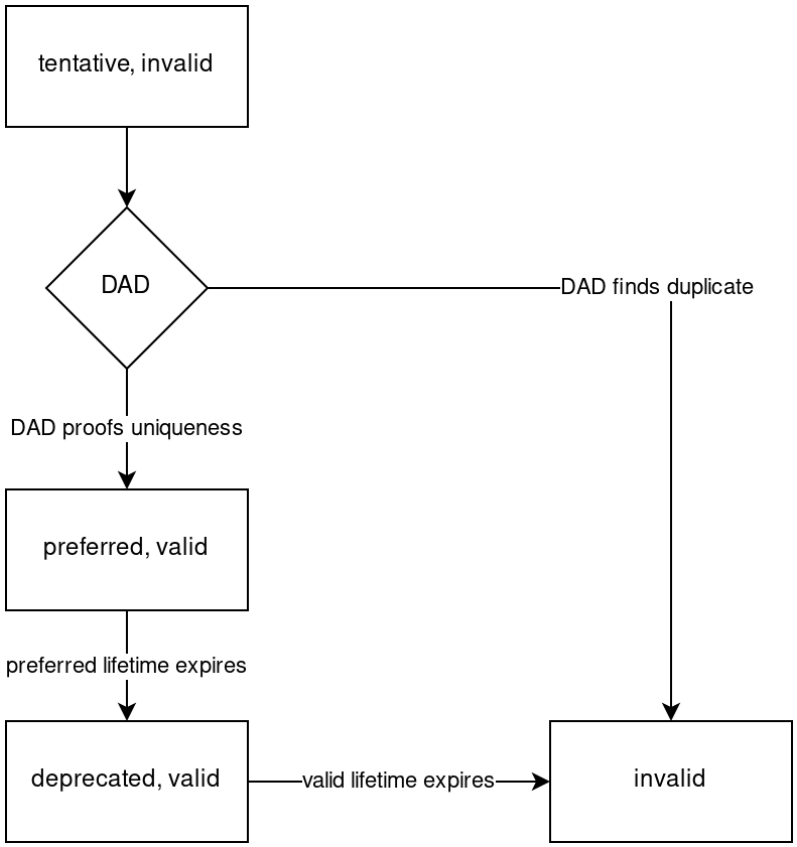


Fig. 3: SLAAC introduces some states of IPv6 addresses, as shown here with the rectangles. Addresses and prefixes provided via DHCPv6 also go through these states.
Once an address gets scheduled for assignment to an interface, it becomes *tentative*. In this state, it must only be used for Duplicate Address Detection (DAD, see 2.4.1).
If DAD succeeds, the address becomes *preferred*. The preferred state is a *valid* state, which is defined by permitting using the address for sending and receiving IPv6 packets. After the *preferred lifetime* expires, the address becomes *deprecated*, but remains valid. The deprecated address is distinguished from a preferred address by discouraging its usage. A deprecated address should not be used for new upper layer connections. It may be used to continue established connections.
After the *valid lifetime* expires, the address becomes *invalid* and can no longer be used. The valid lifetime is counted from the same start time as the preferred lifetime, which is why the valid lifetime must be greater than the preferred lifetime.

As with all unicast addresses, link-local addresses have to pass DAD (see 2.4.1) before getting assigned to an interface.[6] If DAD fails and the address was generated from an interface identifier, this may be because the link-layer address is not unique. In such a case, RFC 4862 recommends to disable the interface entirely, because communication would fail on the link-layer.

2.7.2 Generation of Global Addresses

The following method of SLAAC to generate global addresses can be disabled, but is required to be enabled by default.

ICMPv6 *Router Advertisement* messages can carry *Prefix Information* options. Those contain a prefix, a *Valid Lifetime*, a *Preferred Lifetime*, the *on-link* flag and the *autonomous address-configuration* flag. The latter, if set (value 1), permits generating an address using the prefix. If the on-link flag is set, traffic to addresses within the prefix will be sent directly without using a router. The on-link flag does not alter address configuration.

Like with link-local addresses a global address can be generated by combining the prefix with an interface identifier:

prefix	interface identifier

The length of the prefix must fit to the length of the interface identifier. The preferred and valid lifetime of the generated address are taken from the respective values in the Router Advertisement. Those values can by updated later by new Router Advertisements. As mentioned earlier, such an address must pass DAD before getting assigned to an interface.

2.7.3 DNS Configuration

RFC 8106 [38] specifies two *DNS RA* options for ICMPv6 *Router Advertisement* messages, which allow to transmit IPv6 addresses of *recursive DNS servers* (RDNSS) and a *Domain Search List* (DNSSL). Both options contain a single lifetime and at least one entry. The options may be repeated to specify multiple entries with varying lifetimes.

The RFC claims that the need for DNS configuration (RDNSS addresses and a DNSSL) is inevitable in any "practical network". On nodes that operate both IPv4 and IPv6, the DNS configuration can be obtained via DHCPv4. Because nodes that only use IPv6 cannot rely on this, there is a need for a configuration mechanism in IPv6. DHCPv6 can provide the DNS configuration, but the configuration mechanism introduced by RFC 8106 makes "practical networks" without DHCPv6 possible.

If DNS configuration is obtained from DNS RA options and DHCPv6, RFC 8106 recommends to use all information available and give the DNS configuration from DHCPv6 precedence. If the DNS configuration was supplied using the *Secure Neighbor Discovery* (SEND), that configuration must be preferred.

[6] Performing DAD can be manually disabled by setting the SLAAC protocol variable `DupAddrDetectTransmits` of an interface to zero.

Atlasis and Rey [39] tested in 2015 the behavior of six different operating systems when confronted with DNS configuration from Router Advertisement messages and DHCPv6. They found that different operating systems and versions of operating systems used different sources for their DNS configuration. They concluded that the flexibility of these configuration mechanisms comes with noticeable complexity and the undefined, implementation-specific behavior burdens administrators. They also identified vulnerability of some clients to rogue Router Advertisement messages, which could be abused to distribute malicious DNS configuration. This security issue is also described in RFC 6104 [40].

2.8 DHCPv6

The *Dynamic Host Configuration Protocol for IPv6* (DHCPv6) was originally defined in RFC 3315 [41] but later replaced by RFC 8415 [9] to incorporate other RFCs that amended RFC 3315. DHCPv6 is introduced as "an extensible mechanism for configuring nodes with network configuration parameters, IP addresses, and prefixes. Parameters can be provided statelessly, or in combination with stateful assignment of one or more IPv6 addresses and/or IPv6 prefixes. DHCPv6 can operate either in place of or in addition to Stateless Address Autoconfiguration (SLAAC)".

DHCPv6 is a client-server-protocol that exchanges messages between clients, relay agents and servers. *Clients* are IPv6 nodes that want to receive configuration information, *servers* are those who provide the information and *relay agents* are sometimes needed for message transportation.

The DHCPv6 messages are transported in the payload of UDP datagrams which in turn are transported in the payload of IPv6 packets. This implies that sender and receiver of those messages need valid IPv6 addresses. Using SLAAC, IPv6 nodes will always generate link-local addresses on their own (see section 2.7.1), so for link-local scope, this problem is solved. Unfortunately and usually where clients appear at multiple links, servers may not be attached to the same link as the clients. In this case, clients would need valid addresses of greater scope than link-local. DHCPv6 solves this problem using relay agents.

2.8.1 Relay Agents

One, multiple in chain or no *relay agent* at all may be used. This applies to stateless as well as stateful DHCPv6. If client and server are not connected to the same link, relay agents are necessary. They are placed to be reached by clients using link-scope addresses and have connections to servers or other relay agents that may be unreachable for the client.

A client does not have to be aware of whether it is communicating with a server or a relay agent.

If a client does not know a reachable unicast destination, it sends its messages to the well-known link-scope multicast address `ff02::1:2`. All DHCPv6 servers and relay agents at the link receive traffic to this multicast address. This eliminates the need to configure server addresses on clients. As distinct from DHCPv4 and due to multicasting, nodes that are not interested in the DHCPv6 messages do not receive them.

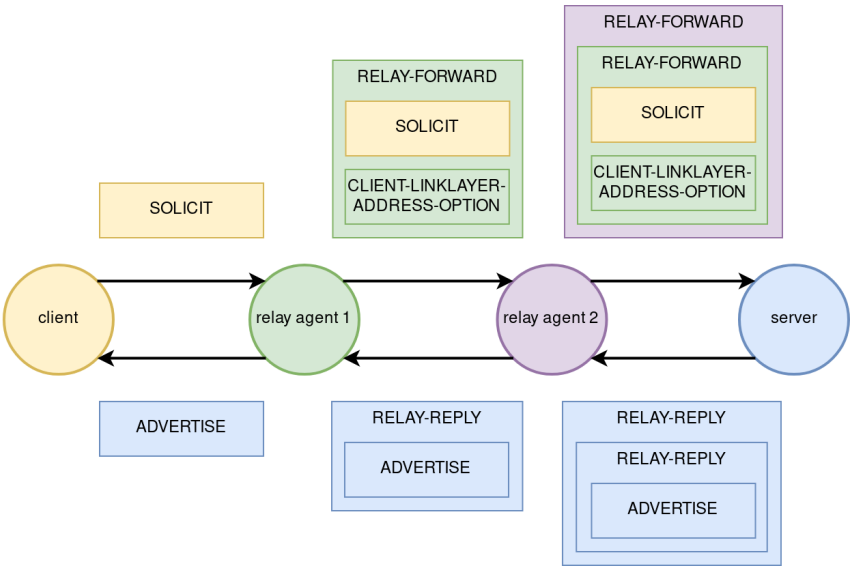


Fig. 4: DHCPv6 *relay agents* are forwarding messages between clients and servers. The rectangles here represent DHCPv6 messages or parts of them and are color-coded by whom they are created. In this example, the client sends a `SOLICIT` message via two relay agents to the server. The first relay agent in chain appends an option. The server uses the `RELAY-FORWARD` messages to figure out how the reply, in this example an `ADVERTISE` message, needs to be nested within `RELAY-REPLY` messages.

The relay agent passes all messages along unchanged, but adds additional information for the server to facilitate the relaying. A relay agent may add further information (e.g. the link-layer address of the client, see [42]) to a `RELAY-FORWARD` message, to aid the server. A relay agent sends by default all messages from clients to the well-known site-scope multicast address `ff05::1:3`, which is meant to target all servers. This default address may be changed into a custom list of unicast and multicast addresses.

To let the server know where to send responding traffic, the relay agent encapsulates the message within a `RELAY-FORWARD` message, see figure 4. This encapsulating message also contains two IPv6 addresses that are used in responding traffic from the server: The *peer-address* is used to address the relay agent and the *link-address* is used by the relay agent to identify the link of the client (or the next relay agent). The server copies those values to a `RELAY-REPLY` message in which it nests the actual message for the client. The nesting of `RELAY-FORWARD` and `RELAY-REPLY` messages may gain multiple layers if multiple relay agents are chained.

Whenever a server wants to send a message via a relay agent, it needs to know how to nest the message. This information is taken from the forerun message of the client. If the server wants to send a `RECONFIGURE` message, it initiates the message exchange. For that reason the server may not be able to derive the necessary information from forerun messages. RFC 8415 section 18.3.11 [9] allows the server to get that information "through some external agent".

For certain messages (see figure 5), the server may add a Server Unicast option into its messages. Then, a client may reply using unicast, bypassing any relay agents. Obviously, the relay agents can not supply additional information in this case and the server should not offer to accept unicast messages, if it is interested in the information added by relay agents.

2.8.2 Client Identification and the Relationship Between IPv6 Addresses and Link-Layer Addresses

With IPv4, every interface got a single IPv4 address. More precisely, DHCPv4 assigns a single IPv4 address to a single link-layer address.

This relationship has changed with IPv6: A node may have multiple IPv6 addresses per interface, meaning per link-layer address. Usually IPv6 nodes have at least a link-local and a global scope IPv6 address.

DHCPv6 accommodates this fundamental change by no longer relying on link-layer addresses to identify clients. Instead, each client and server (per node, not per interface) generates a *DHCP Unique Identifier* (DUID). This value may be formed out of a link-layer address, but must be treated as an opaque value, as clarified by RFC 8415 section 11 [9]. Nevertheless, some implementations violate this restriction and extract the link-layer address from the DUID. A DUID may change for the same node when a different implementation or configuration of that is used (e.g. when booting a different operating system).

However, DHCPv6 servers may use the use the link-layer address of the requesting interface when deciding about a lease. If server and client communicate over a single link, the server could obtain the link-layer address directly from the link-layer messages or by using the Neighbor Discovery. If one or more relay agents are between them, the relay agent connected to the client may add the link-layer address of the client to the DHCP message, as standardized by RFC 6939 [42] and illustrated in figure 4.

To satisfy the desire to identify DHCPv4 and DHCPv6 clients with the same identifier, both protocols need to support the same identifier. This can be the link-layer address or the DUID. It should be noted that the former identifies individual interfaces while the latter identifies whole nodes. The link-layer address was originally used by DHCPv4 and its predecessor BOOTP, and can be used with DHCPv6. The DUID is the recommended identifier type for DHCPv6. RFC 4361 [43] specifies how it can be used in DHCPv4.

As a side note, since 2012, Google prominently refuses to implement a DHCPv6 client in the popular operating system *Android*, because they fear DHCPv6 server operators will not provide multiple addresses per interface and thereby break some functionality which depends on that. Despite the possibility to assign multiple addresses per interface with DHCPv6, Google assumes that the operators will limit that to the relationship cardinality of DHCPv4, likely to reduce the number of addresses in the network or to require fewer changes when enabling IPv6. Instead, Android uses SLAAC with the DNS configuration extension (see section 2.7.3) as IPv6 configuration mechanism. (See [44]–[46].)

The lack of a DHCPv6 client in Android does not preclude DHCPv6 as configuration mechanism for the scope of this work, because Android nodes are not among the intended clients. Nonetheless, the decision of Google shows that some usecases would benefit of multiple IPv6 addresses per link-layer address. Also, this might be of interest if the usecases of the anticipated IPv6 configuration mechanism grow by new kinds of nodes.

message format

message-type	transaction-id	options
1 byte	3 byte	variable

option format

option-code	option-len	option-data
2 byte	2 byte	variable

Tab. 5: The left table shows the format of a DHCPv6 message as it is sent over the network. The options field contains a variable number of options, whose format is shown in the right table. The length of each field is denoted below them.

DHCPv4	DHCPv6
DHCPACK	REPLY
DHCPDECLINE	DECLINE
DHCPDISCOVER	SOLICIT
DHCPINFORM	INFORMATION-REQUEST
DHCPNAK	REPLY
DHCPOFFER	ADVERTISE
DHCPRELEASE	RELEASE
DHCPREQUEST	REQUEST
DHCPREQUEST	RENEW
DHCPREQUEST	REBIND
-	CONFIRM
-	RECONFIGURE
-	RELAY-FORWARD
-	RELAY-REPLY

Tab. 6: Comparison of the DHCPv4 and DHCPv6 messages types defined in RFC 2131 [8] and RFC 8415 [9]. There are more message types defined in other RFCs. The bottom four message types do not have (dedicated) equivalents in the considered DHCPv4 message types.

2.8.3 DHCPv6 Messages

All DHCPv6 messages are built very modular. The only fixed fields are the message type and the transaction identifier (see table 5). Everything else is contained within "options". Some options may contain other options. Options consist of fields for the type, the length and option-specific data (see table 5). The total length of a message can be obtained from the *length* field of the UDP datagram.

This architecture avoids transmitting unnecessary information and remains extensible.

All message types defined by RFC 8415 are shown in figure 5. The message types of DHCPv6 are compared to DHCPv4 message types in table 6.

2.8.4 Retransmission of DHCPv6 Messages

After a client sends a message, it waits for an answer. If it does not receive any, it retransmits its message. RFC 8415 defines for each message type how often, for how long at what times a message should be transmitted. The retransmission mechanism is meant to account for any messages that did not make it to the server. To inform a server of how long the client has attempted a particular message exchange, the client adds this time in hundredths of a second to each message using the Elapsed Time option. The client updates this time every time it

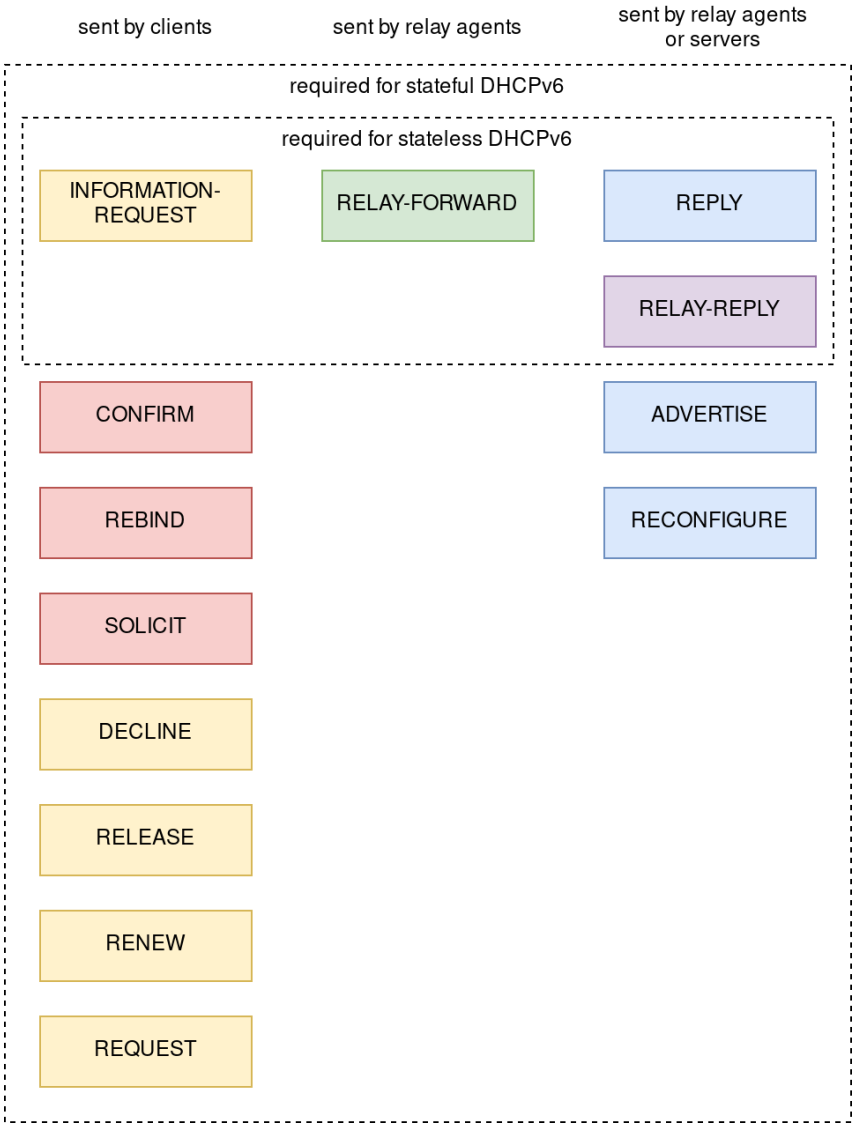


Fig. 5: These DHCPv6 message types are defined by RFC 8415 [9]. The red messages are multicasted to all servers and relay agents. The yellow messages are either multicasted to all servers and relay agents or, if the server allowed that using a *Server Unicast* option, unicasted to a server.[8] The `RELAY-FORWARD` message is unicasted to a server or multicasted to all servers. The `RELAY-REPLY` message is unicasted to the appropriate relay agent. The blue messages are unicasted to clients.

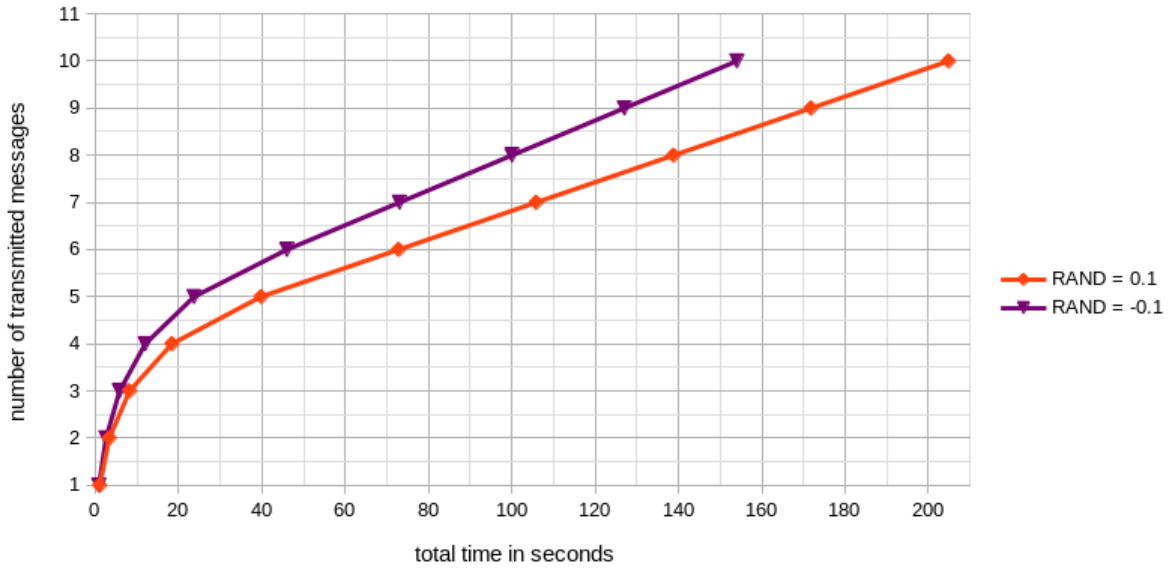


Fig. 6: As per RFC 8415, DHCPv6 `REQUEST` messages are transmitted 10 times if the server does not reply. The delay between those messages is shown here: each mark on a line stands for a message transmission. The red line shows the times for when the randomization factor has the greatest possible value every at every retransmission, the purple line for when it has the smallest possible value every time.

transmits the message.

The duration after which to retransmit a message is calculated after every transmission to incorporate a randomization factor. This is done to avoid congestion when many clients want to perform a message exchange simultaneously. The duration until the next transmission is first increased exponentially, but then linear after a threshold (which depends on the message type). The retransmission is limited for some messages by a number of transmissions or a total duration. Messages of type `SOLICIT` and `INFORMATION-REQUEST` get transmitted indefinitely, but all other message transmissions (as defined in RFC 8415) declare failure eventually.

As an example, figure 6 shows how `REQUEST` messages get retransmitted.

The algorithm for (re)transmission of messages is varied for some message types, e.g. for `SOLICIT` messages, as described in 2.8.7.

Messages sent by servers do not get retransmitted, with the exception of `RECONFIGURE` messages. When sending ADVERTISE or REPLY messages, a server does not need answers from the client. Using a `RECONFIGURE` message, the server can initiate a message exchange (see section 2.8.8), which makes it responsible for assuring the communication partner received the message. A `RECONFIGURE` message is retransmitted for a limited number of times.

[8] RFC 8415 states in section 16 "A server MUST discard any [...] Information-request messages it receives with a Layer 3 unicast destination address.", whilst section 18.4 states "For [...] Information-request [...] messages, [unicast transmission] is allowed only if the Server Unicast option is configured.". It is not clear, whether the standard permits unicast transmission of `INFORMATION-REQUEST` messages. The *Dynamic Host Configuration Working Group* of the *Internet Engineering Task Force*, who developed the standard, is currently discussing under what circumstances unicast transmission of `INFORMATION-REQUEST` messages is allowed, see [47].

2.8.5 Stateless vs Stateful DHCPv6

A DHCPv6 server may operate either in *stateless* or *stateful* mode, as mentioned in section 2.6. In stateless operation, only *other configuration information* is provided. A stateful DHCPv6 server provides *additionally* so-called *Identity Associations*, which convey addresses or prefixes and thereby provide address configuration (see section 2.8.7). Clients and servers that only use stateless DHCPv6 only need to implement a subset of the protocol. The necessary messages for stateless and stateful DHCPv6 are shown in figure 5.

The availability of at least one DHCPv6 server can be indicated in *Router Advertisements* (see section 2.4.1). The "Other Configuration" flag in a Router Advertisement signals whether stateless DHCPv6 is available. The "Managed Address Configuration" flag expresses the availability of a DHCPv6 server that assigns addresses. The latter is more precise than indicating the availability of a stateful DHCPv6 server, because a stateful server could not provide addresses but other Identity Associations (currently only prefixes). (See RFC 4861 [32].)

A node may, but is not required to use DHCPv6 upon receiving Router Advertisements that suggest the availability of a DHCPv6 server. A node can also attempt to use DHCPv6 regardless of any Router Advertisements.

Like with DHCPv4, multiple DHCPv6 servers can offer their service to a client simultaneously.

2.8.6 Stateless DHCPv6

The other configuration information is supplied using merely two messages: The client sends an `INFORMATION-REQUEST` message to the server, which replies with a `REPLY` message. The message of the server includes a lifetime that specifies the time interval after which the client should update the information by querying the server again.

Because the client multicasts the `INFORMATION-REQUEST` message to all servers and relay agents, it could receive multiple `REPLY` messages. RFC 8415 does not specify how to process multiple `REPLY` messages, so this is up to the client implementation.

A client is not required to include a Client Identifier option in the `INFORMATION-REQUEST` message to hide its identity. A server is not required to answer to such a request.

The client uses an *Option Request* option to specify which information it is interested in. It can include further options to specify values it would like to receive.

2.8.7 Stateful DHCPv6

In contrast to stateless DHCPv6, stateful DHCPv6 can be used to assign *Identity Associations* (IAs) to clients and manage the life-cycle of the IAs. An IA contains either addresses or prefixes and thereby facilitates the address configuration fo IPv6 nodes. The permission granted by a DHCPv6 server to use an IA for a certain period of time is called a "lease". A "binding" is a group of *client-specific* information at the server, consisting of IAs and/or *other configuration information*. This terminology is according to RFC 8415 [9].

Finding and Selecting Servers The *Server Discovery* algorithm enables a client to find available DHCPv6 servers and select one of them.

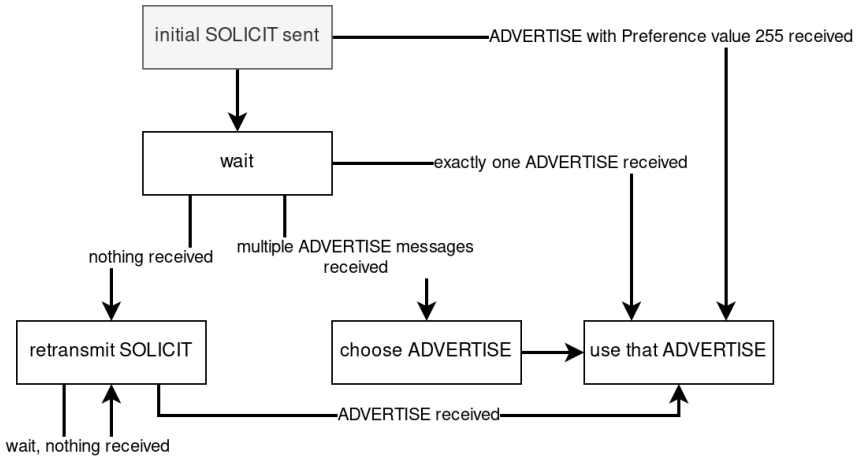


Fig. 7: This flowchart shows how a DHCPv6 client chooses a server. This is described in more detail in 2.8.7.

First, a client locates all available servers by sending a `SOLICIT` message to the well-known link-scope multicast address of all servers and relay agents. The servers (if need be through a relay agent) answer with an unicasted `ADVERTISE` message.

The client generates a transaction ID and includes it together with its DUID and an *Option Request* option in the `SOLICIT` message. The *Option Request* option contains option codes of those in which the client is interested. The client also includes options for IAs, as described later.

The server copies the transaction ID and client DUID into the `ADVERTISE` message. It also adds its DUID and values for IAs and the options which the client requested. If configured to do so, the server adds a *Preference* option and a *Reconfigure Accept* option. The latter indicates support for the reconfiguration mechanism, which is explained in section 2.8.8.

The following selection process is illustrated in figure 7.

The client waits for more than a second[9] to receive `ADVERTISE` messages. If the client receives an `ADVERTISE` message with a preference value of 255, it aborts waiting and selects the server of that `ADVERTISE`. If the client receives a single `ADVERTISE` after the waited interval, it answers to that. If it did not receive any, it retransmits the `SOLICIT` message and waits about a second[10]. The retransmission is repeated indefinitely until first receipt of an `ADVERTISE` message, which will be used by the client. If the client received multiple `ADVERTISE` messages after the first `SOLICIT` message, it chooses one.

If the client can choose, it should select the server with the highest preference value. It may disregard this rule to select a server with a better set of advertised parameters. The best set of advertised parameters also wins among multiple `ADVERTISE` messages with the same preference value.

Rapid Commit The algorithm of selecting a server is slightly different when the client wants to use the *rapid commit* mechanism. This mechanism implements the configuration process with

[9] The client waits for SOL_TIMEOUT + SOL_TIMEOUT · RAND, where SOL_TIMEOUT is one second and RAND is a random real number between zero (exclusive) and 0.1 (inclusive).

[10] The time to wait is changed after the first `SOLICIT` message by changing the lower bound of RAND to −0.1.

name	description	T1 and T2 values	IAID	other content
IA_TA	temporary addresses		×	list of addresses
IA_NA	non-temporary addresses	×	×	list of addresses
IA_PD	prefix delegation	×	×	list of prefixes

Tab. 7: These *Identity Associations* are available in DHCPv6. The IA_TA does not contain T1 and T2 values. All contain an identifier, the IAID. Each address and prefix has preferred and valid lifetimes associated with it. See section 2.8.7.

only two messages, as a performance optimization. The client can request this by including a *Rapid Commit* option in the `SOLICIT` message. Servers that offer support for rapid commit reply with a `REPLY` message containing a *Rapid Commit* option, instead of an `ADVERTISE` message. Simultaneously, other servers may send `ADVERTISE` messages to the client.

The client uses the first appropriate REPLY message it receives and aborts waiting for more messages, even when only having sent the initial `SOLICIT` message.

Until the next retransmission of the `SOLICIT` message, the client may have received `ADVERTISE` messages but no appropriate `REPLY` messages. In this case, the client starts the configuration without the rapid commit mechanism. While this is in progress, the client could still receive `REPLY` messages with Rapid Commit option. In this situation the client may decide to cancel the normal configuration and use the `REPLY` message with Rapid Commit option.

Identity Associations and Lifetimes An *Identity Association* (IA) is a group of stateful information of a single type. RFC 8415 [9] specifies the three types listed in table 7. Addresses in IAs are meant to be assigned to interfaces of the requesting node. Prefixes in IAs can be used by the DCHPv6 client to in turn assign contained addresses or prefixes to other nodes (so the DHCPv6 client sends Router Advertisements and/or becomes a DHCPv6 server).

A DHCPv6 client can request multiple addresses or prefixes by including multiple IAs in a `SOLICIT` message. A client has to use at least one IA per interface. It chooses an identifier per IA that is unique among itself and the IA type.

Each address and prefix has a preferred and valid lifetime, as described in figure 3. Those values may be set to infinity, but usually they expire eventually. DHCPv6 has a mechanism to extend lifetimes, which can be applied to all lifetimes, except to the preferred lifetime of temporary addresses. This exception is what distinguishes temporary addresses from non-temporary addresses. After the preferred lifetime of a temporary address expires, the addresses is in deprecated state, meaning it is only used for established, but not for new upper layer connections (e.g. TCP sessions). The deprecated state may be prolonged by extending the valid lifetime.

The extension of lifetimes is controlled by two values in an IA named T1 and T2. T1 is the time interval after which the client should start to *renew* the IA. T2 is the time interval after which the client should start to *rebind* the IA. The difference is, that when *renewing*, the client only contacts the server from which it obtained the IA. When *rebinding*, the client contacts all servers. Both behaviors are explained in the following sections. IA_TA do not come with T1 and T2 time intervals, because their preferred lifetime can not be extended. The values should

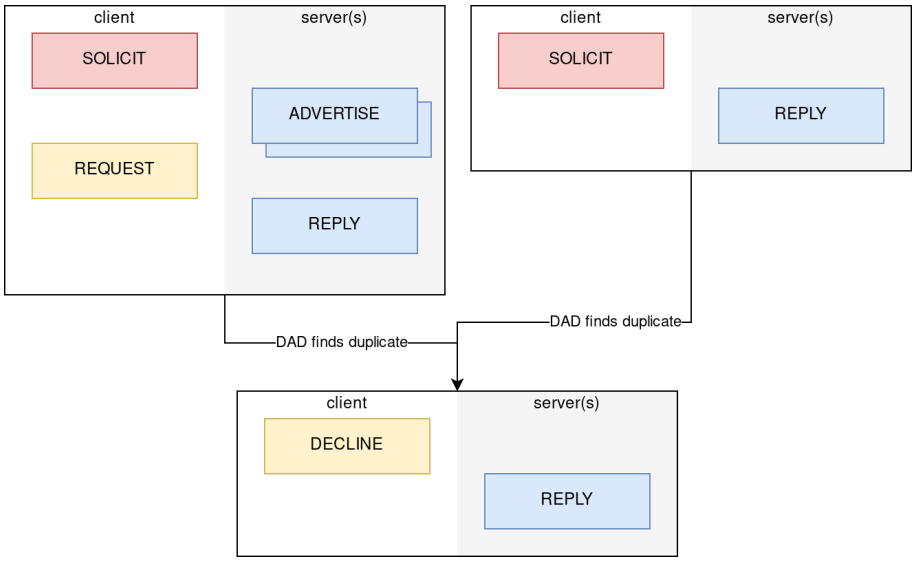


Fig. 8: These message exchanges can be used by DHCPv6 clients to obtain Identity Associations. There can be Relay Agents involved between client and servers, which are not shown here. This is described in more detail in 2.8.7.

satisfy the following equation[11]:

$$\text{T1} < \text{T2} < \text{smallest preferred lifetime} \tag{2}$$

The server may set T1 and T2 to zero to indicate the client may choose appropriate values.

On-Link Determination A client is not allowed to assume a certain prefix is *on-link* because it received a certain IA_NA or IA_TA. Instead, it should rely on the information propagated by Router Advertisement messages (DHCPv6 has no mechanism to convey that information). If an IA_NA or IA_TA is assigned to a client, but the prefix was not specified as *on-link* in a Router Advertisement message, the client will presume all addresses of that prefix (except its own) are *off-link* and send traffic to them via a router. The router may use an ICMPv6 Redirect message to inform the client that a certain address is in fact on-link. If a client learns that an address or prefix is on-link, it will send traffic to that directly over the link.

Obtaining Identity Associations for Address Assignment A DHCPv6 client can obtain one or more IA_NA or IA_TA using the message exchanges described hereafter and illustrated in figure 8. IA_PD are not considered here because none of the clients in the RMC-DHCP or PXE-DHCP group is expected to delegate prefixes to other nodes.

First, a client selects a server using the *Server Discovery* algorithm described in 2.8.7.

During the Server Discovery, the clients can include IA_NA or IA_TA options in the `SOLICIT` message to obtain IAs. The client chooses IAIDs. Each IA_NA and IA_TA needs to be associated with exactly one client interface. The client can request multiple IAs per interface by either including multiple IA_NA or IA_TA options in the messages to the server or by repeating

[11] T1 and T2 may equal if they and the lifetimes are set to infinity. If the server is not willing to extend the lifetimes, it may set T1 and T2 to the valid lifetime.

the message exchange with different IAIDs. The client does not nest any IA Address options in the IA_NA or IA_TA options in the `SOLICIT` message, because it does not know the addresses it may get yet. The client can set the T1 and T2 fields of IA_NA options in `SOLICIT` messages to values it would like to get.

The server can assign multiple addresses (IA Address options) to each IA_NA or IA_TA option. It fills in all values it is willing to offer and sends those in the `ADVERTISE` and `REPLY` messages.

When sending a `REQEST` message, the client copies the IA_NA and IA_TA options from the `ADVERTISE` message of the server. The client may adjust the lifetimes, T1 and T2 values to request specific values. The server may honor the requested values.

Before assigning them to the interfaces, the client performs DAD (see 2.4.1) for each address. If DAD reports that the address is already present on the link, the clients notifies the server using a `DECLINE` message. The server confirms the rejection using a `REPLY` message. By protocol default, the client attempts four times to send a `DECLINE` message.[12]

Maintaining Identity Associations Some time after a DHCPv6 client obtained IAs, it might wish to extend the preferred and valid lifetimes of the addresses or prefixes. The preferred lifetime of temporary addresses can not be prolonged by design, hence the following only applies to non-temporary addresses and prefixes. A server can indicate that it will not extend lifetimes of non-temporary addresses or prefixes by setting the T1 and T2 values to the valid lifetime.

The following process is shown in figure 9.

After the interval T1 passed, a client should start attempting to *renew* its addresses or prefixes. This is done by sending a `RENEW` message to the server that assigned the IAs to the client. The server may extend some lifetimes. It can be configured to assign, if necessary, new addresses or prefixes to an IA. The server sends a `REPLY` message to the client in which it includes the updated IAs.

If the DHCPv6 server receives a `RENEW` message with an IA it does not know about, it has two choices:

- The server may create a new binding with new addresses or prefixes. If it is configured to do so, but has no addresses or prefixes available, it sends the IA with the status code `NoAddrsAvail` or `NoPrefixAvail`.
- The server may not create a new binding. In this case, it sends an IA with the status code `NoBinding`. The client responds to this with a `REQUEST` message.

If the client failed to renew, and T2 expires, it starts to *rebind* its addresses or prefixes. To do this, the client multicasts a `REBIND` message to all servers and relay agents. The server that assigned the IAs to the client may receive the `REBIND` message and answer with a `REPLY` message like to a `RENEW` message. Other servers may receive the `REBIND` message too. If they do not want to update any IAs, they refrain from sending a `REPLY` message. If a server finds an IA in a `REBIND` message for which it has not assigned any addresses or prefixes, it may choose to assign

[12] See the variable `DEC_MAX_RC` in RFC 8415 [9].

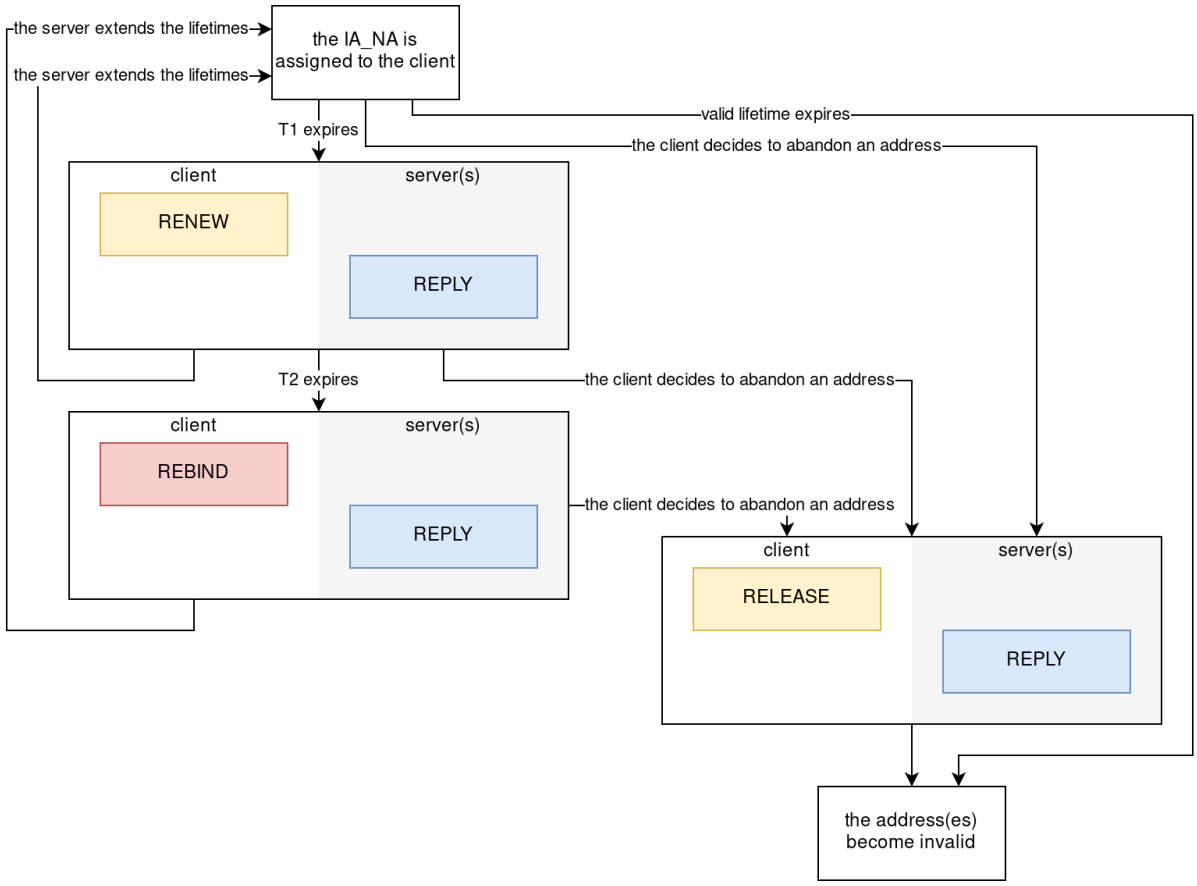


Fig. 9: These message exchanges can be used by DHCPv6 clients to extend lifetimes of or relinquish addresses or IA_NA. This also applies to prefixes of IA_PDs. Addresses of IA_TA may follow this too, but they can not extend their preferred lifetimes and do not have T1 or T2 values. There can be Relay Agents involved between client and servers, which are not shown here. This is described in more detail in 2.8.7.

new addresses or prefixes. This should only be done if the server supports the *rapid commit* message exchange.

Upon receiving a `RENEW` or `REBIND` message, a server may inform a client that some of its addresses are not appropriate for the link on which they are used by updating the corresponding lifetimes to 0 using the `REPLY` message.

Once a client knows it will no longer use an address or prefix (e.g. because it is about to shut down), it can inform the server by sending a `RELEASE` message. The client includes all and only the addresses or prefixes it wants to relinquish in the `RELEASE` message. The server confirms that using a `REPLY` message.

If a client wants to know whether its addresses are still appropriate for the links, it can ask a server by multicasting a `CONFIRM` message containing the addresses in question. The servers respond with a `REPLY` message that approves or rejects the addresses with a single Status Code option.

2.8.8 Server-Triggered Client Reconfiguration

A DHCPv6 server can initiate a message exchange by unicasting a `RECONFIGURE` message to a client. The server learns the address of the client from previous message exchanges. Using the Reconfigure Message option, it can specify whether the client should answer with a `RENEW`, `REBIND` or `INFORMATION-REQUEST` message. Thereby, the server can trigger an update of the Identity Associations and other configuration information at the client.

The client will ignore all `RECONFIGURE` messages after it has received one until it has completed the requested message exchange.

To protect clients against malicious `RECONFIGURE` messages, the server is required to authenticate its `RECONFIGURE` message using the Reconfiguration Key Authentication Protocol specified in RFC 8415 [9].

The client uses the Reconfigure Accept option in `SOLICIT` and `REQUEST` messages to let the server know whether it will accept `RECONFIGURE` messages. Likewise, the server should include the Reconfigure Accept option in answers to that messages, to inform the client of whether it intends to send `RECONFIGURE` messages.

2.9 Redundant Service Instances

As the desired configuration mechanism for IPv6 nodes is planned to serve many clients across multiple data centers, avoiding service interruptions and providing the service fast become requirements. Therefore, the following sections clarify terms concerning high availability and loadbalancing.

2.9.1 High Availability

The term *high availability* refers to the goal to provide a service without any interruption. The duration of interruption is called *downtime*. (See [48].)

In order to reduce downtime, a service needs to tolerate faults. There are various types of fault tolerance (see [49]).

One option is to connect clients to other, functioning servers in the event that the current server fails. Such a maneuver is called a *failover*. If the new server has not served clients until then, it is called a *standby* server. If the failover can be done without considerable downtime, it is said to be a *hot standby* server. Otherwise, if the standby server requires preparation before serving clients, it is called a *cold standby* server. If the servers have hot standby servers in different geographic locations, they are said to be *georedundant*. (See [48], [50].)

There are many other aspects to be considered when aiming for high availability (see [50] for example). However, this work does not intend to cover all of them, but rather focus on what can be achieved using redundant service instances.

2.9.2 Loadbalancing

The term loadbalancing refers to the distribution of workload across multiple distributed instances of a service (see [51], [52]). The workload can be measured in various ways, for example

in CPU load or network bandwidth usage (see [49]).

Loadbalancing can be used to pursuit different goals (see [52]), some of which are listed below.

- The *service latency* is the delay at the client between sending a message and receiving a matching answer. It can be improved by reducing the network latency which in turn can be improved by moving server and client closer together (in terms of geography and network topology). Loadbalancing can help to reduce network latency by routing traffic to the closest service instance.

- Loadbalancing is crucial to make a service *scalable*. Duboc et al. [53] found that the term *scalability* is poorly defined and attempted to define it as "a quality of software systems characterized by the causal impact that scaling aspects of the system environment and design have on certain measured system qualities as these aspects are varied over expected operational ranges". This general definition is not practical for the much more limited scope of this work. In this context, scalability is considered to be *the ability to increase the number of requests the service can handle by adding new service instances*. By distributing the workload, loadbalancing enables a service to scale beyond the capabilities of a single service instance.

- If provided with the capability to detect the failure of a service instance, loadbalancing can also provide failovers.

2.9.3 Anycast Using BGP

The *Border Gateway Protocol* (BGP, see RFC 4271 [54]) is a path-vector routing protocol that is used to exchange IPv4 and IPv6 routing information between routers. BGP is the predominant routing protocol of the Internet and is heavily used in the networks of the *United Internet AG*.

A BGP router learns BGP routes from its configuration and other BGP routers. A BGP route consists of a target IP prefix, the address of the next router (next-hop) and more information (e.g. a "cost"). The BGP router collects all that information in its *Routing Information Base* (RIB). A RIB often contains multiple BGP routes for a single target IP prefix. Based on that, the BGP router decides which BGP route to write in the *Forwarding Information Base* (FIB). The FIB contains for each target IP prefix usually exactly one next-hop address, because is used to actually forward IP packets.

BGP can be leveraged to create anycast traffic (see figure 2) across many links by configuring appropriate route information. This can be used as a failover or loadbalancing mechanism.

In a failover configuration, multiple BGP routes are configured in the RIB such that one at a time is preferred (e.g. using different *costs*). When a failure happens (e.g. because the next-hop becomes unavailable), the route is removed from the RIB and the optimal route is recalculated and updated in the FIB.

In a loadbalancing configuration, multiple BGP routes are configured in the RIB *and in the FIB*. An algorithm (e.g. round-robin) must be configured to select one next-hop address for each packet that needs to be routed. This way of selecting a next-hop address is called *Equal Cost Multi Path* (ECMP) routing.

As the destination host may change any time per IP packet, anycast based on BGP should only be used for connections that consist of a single request and reply (e.g. DNS queries) or such that do not last for long.

2.9.4 IPv6 Link-Scope Anycast

Neighbor Advertisement messages have a flag called *Override*. This toggles, whether the recipient of the message should remove other link-layer address from its cache. If the Override flag is not set, the recipient will continue to use the link-layer address of the first Neighbor Advertisement it received. To avoid congestion, such Neighbor Advertisements should be delayed by a random amount of time, with a fixed upper bound.

This makes not-deterministic link-scope anycast addresses possible. It can be regarded as a sort of hot-standby redundancy mechanism where the active side is randomly chosen per client whenever Neighbor Advertisements are sent.

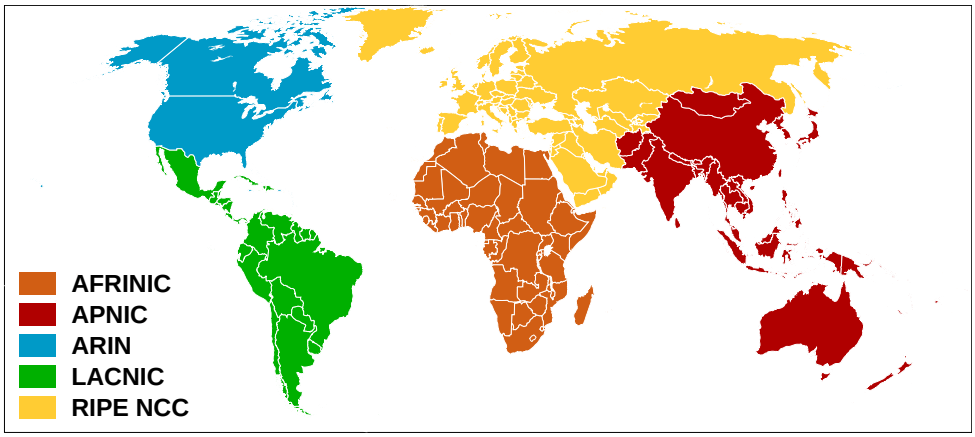


Fig. 10: The *IANA* allocates the IPv4 and IPv6 address space to these five *Regional Internet Registries*. (see [55], image source: [56])

3 Evaluation of a Migration to IPv6-only

This section highlights problems of IPv4 in general and discusses a migration from networks that only use IPv4 to networks that only use IPv6 ("IPv6-only"). This general consideration is applied to the usecases of the *United Internet AG* which are specified in section 1.1.

3.1 The Problem: IPv4 Address Exhaustion

The allocation of IP addresses is made hierarchically. The *Internet Assigned Numbers Authority* (IANA) is the organization which has the ultimate authority on the IP address allocation. The IANA distributes the entire address space in big blocks among the five *Regional Internet Registries* (RIRs, see figure 10), which in turn allocate slices of their parts to *Local Internet Registries* (LIRs). Eventually, an *Internet Service Provider* (ISP) can get globally unique IP addresses and configure them on its computers (users). [55]

The IANA has allocated 3 707 764 736 addresses (86 %) of the IPv4 address space to RIRs [57]. All remaining addresses are reserved for special purposes. All of the five RIRs have in turn allocated (almost) all of their addresses to LIRs. The RIR *RIPE NNC* has completely run out of IPv4 addresses [58], as did *ARIN* [59]. *AFRINIC* is allocating from a final /11 IPv4 address pool [60]. *LACNIC* allocates at most a /22 IPv4 address block per member from a final address pool which is expected to drain on 2020-09-27 [61]. *APNIC* allocates at most a /23 IPv4 address block per member [62].

Hence, the IPv4 address space should be considered exhausted.

3.1.1 Mitigation: Network Address Translation (NAT)

Since there are not enough addresses to give each computer a unique or even multiple unique addresses, the uniqueness cannot be fulfilled. Special address ranges have been defined where the uniqueness may be violated [7]. Networks within that address ranges still need to have unique addresses, but other networks with the same addresses may exist (overlapping address spaces).

In the style of the IPv6 addressing architecture (see section 2.3.2), such networks will here be called *zones of non-global scope*. They have gained extensive usage.

To route IPv4 packets between zones of non-global scope, a mechanism called *Network Address Translation* (NAT) was standardized. RFC 2663 [63] specifies terminology around variations of NAT. The variations have in common to change addresses in IPv4 headers so that they are valid within the zone to which they are routed. This is done by routers that connects zones. For the endpoints of the communication the NAT is "transparent".

RFC 2663 differentiates between *packet flow* and *session flow*, which both indicate the direction of IPv4 traffic with respect to an interface. The former is the direction of a single packet, the latter the direction of a *session*, which is defined to be a "set of traffic that is managed as a unit for translation".

Mapping a Single Address to Many A "Network Address Port Translation" (NAPT) is a variation of NAT that is like a power strip: It translates a single address into multiple. The side of the NAPT with the single address is named *external*, while the other side is named *private*. Figure 11 shows an example for a network with NAPT.

To identify to which of the multiple addresses traffic needs to be translated, NAPT alienates address space from transport layer protocols (see table 2). TCP [15] and UDP [14] port numbers, as well as ICMPv4 [30] query identifiers[13] are used for target identification. The number of possible addresses in the transport layer protocol becomes an upper bound for the number of sessions that is shared among the addresses on the private side of the NAPT.

Session flow is only allowed in the direction from the private to the external side, meaning the external side cannot initiate sessions. Therefore, the NAPT router needs to track the state of sessions.

Patching Address Collisions The NAT variation "Twice NAT" translates source *and* destination addresses of IPv4 headers. It allows session flow in both directions. When the same addresses occur in both networks between which traffic is desired, Twice NAT can help. Each network needs a special DNS server that maps all domains of targets in the other network to the address of the Twice NAT router.

3.1.2 Service Differentiation on Upper Layer Protocols

Instead of requiring more IPv4 addresses, services can be made available over a single address and be differentiated at a higher level protocol (e.g. using TCP ports or HTTP URLs). But this requires to run all services on the same node, which might not be reasonably possible in practice due to the required resources.

[13] ICMPv4 is defined to be at the network layer. But because it is transmitted in the payload of IPv4, it needs to be treated like a transport protocol in a NAPT environment.

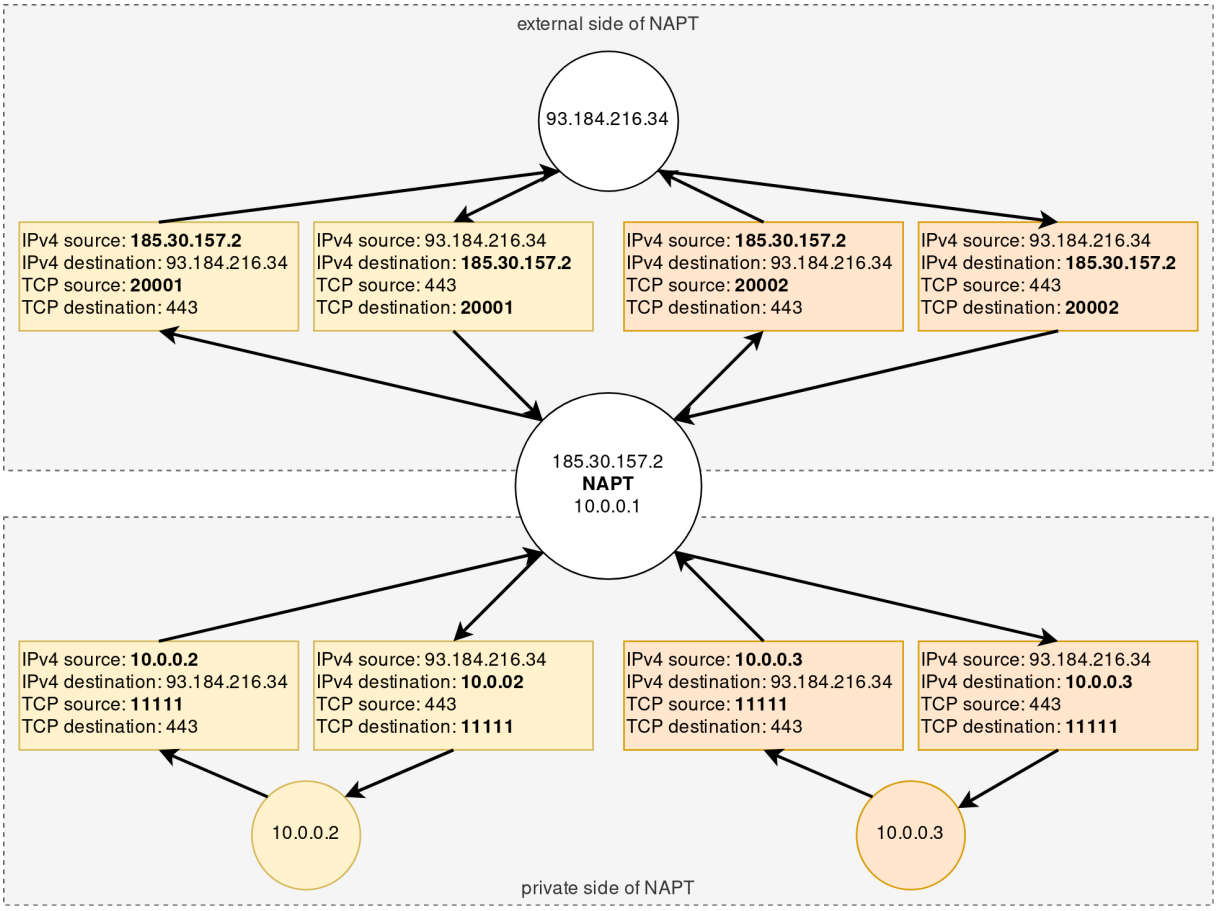


Fig. 11: Fictional example for communication via NAPT: The router doing the NAPT alters addresses in the IPv4 headers. Thereby the two nodes (the circles) on the private side are seen as a single IPv4 address. The router distinguishes the connections by altering the TCP ports too.

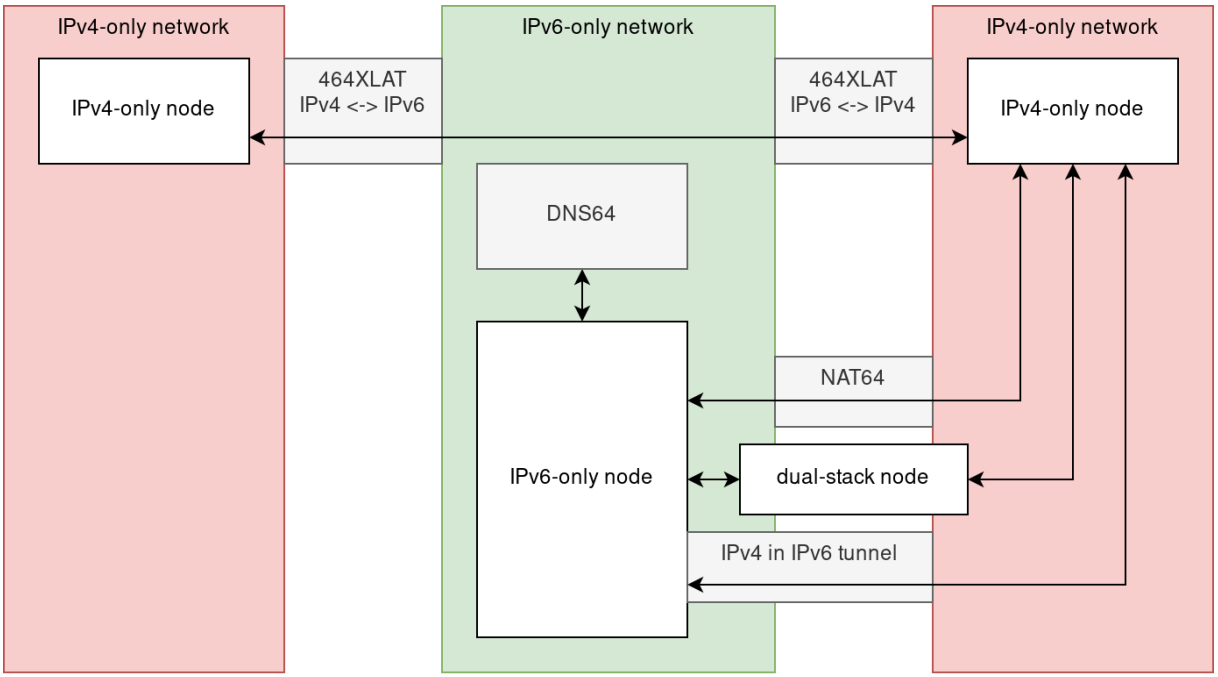


Fig. 12: These mechanisms for exchanging data between IPv6 and IPv4 nodes are explained in section 3.2.

3.2 Connecting IPv6 Nodes With IPv4 Nodes

There are many mechanisms that ease the migration to IPv6. A migration can be done network per network. IPv6 can be enabled while IPv4 is still operational. However, before moving to an IPv6-only network, the used upper-layer software needs to be examined for compatibility with IPv6. The support of IPv6 is common nowadays but should not be taken for granted.

Connectivity at the network layer between IPv6 and IPv4 nodes can be established using various mechanisms, which are described hereafter and shown in figure 12. Such connectivity is desired in most cases, as not all nodes support IPv6.

- Dual-Stack means to use IPv6 and IPv4 by the same node at the same time (see [64]). In that case they need to decide which protocol to use when. Dual-Stack nodes can be configured to use *Happy Eyeballs*, which is an algorithm for choosing IPv6 or IPv4 if both are available (see [65]). The algorithm attempts to establish a connection using both IPv6 and IPv4, gives IPv6 a head start and then only uses the first successfully established connection. This gives IPv6 preference, but provides a quick failover to IPv4 in case IPv6 does not work.
  A Dual-Stack node can be used as *proxy* for certain upper layer protocols (e.g. HTTP, see [66]). A proxy uses an upper layer protocol on behalf of other nodes, which makes it possible to translate the upper layer protocol between IPv6 and IPv4.

- Configuring nodes with IPv6 and disabling IPv4 ("IPv6-only") is an alternative to Dual-Stack. In this setup, IPv6-only nodes would not be able to communicate with nodes that use only IPv4.

- *NAT64* is a mechanism to translate traffic between IPv4-only and IPv6-only hosts (see [67]). A router running NAT64 must be connected to IPv6 and IPv4 (dual-stack). NAT64 is intended to be used in conjunction with DNS64 (see [68]), which extends an DNS server to generate `AAAA` records from `A` records such that IPv6-only nodes will send traffic to IPv4-only nodes to those generated addresses. They are routed to a node performing NAT64, which translates the traffic to IPv4 (and back, for replies). DNS64 works fine until the authoritative nameserver cryptographically secures its DNS records using DNSSEC (see [69]). Then, a resolver that validates DNSSEC will reject the `AAAA` records generated by the DNS64 server (or the DNS64 server will not attempt to generate `AAAA` records), because the DNS64 server lacks the authority to add records. Linkova described this DNSSEC issue and found that, depending on the DNS64 implementation and the resolver, up to 94 % of the Alexa 1M websites would not be available through NAT64 with DNS64 (see [70]).
- Another way to connect node in an IPv6-only network to an IPv4 node is to *tunnel* (e.g. an IPsec VPN) a IPv4 connection through the IPv6-only network. The IPv4 target must be either within the tunnel or a dual-stack node needs to route the IPv4-traffic from the tunnel to an IPv4 network.
- A connection between two IPv4 nodes over an IPv6-only network is also possible using *464XLAT* (see [71]). This translates all IPv4 packets to IPv6 packets and back, to transport the IPv4 packet over an IPv6 network.

3.3 What IPv6 Does Differently Than IPv4

This section discusses how IPv6 differs from IPv4 and whether it should be preferred over IPv4.

3.3.1 Solving the Address Exhaustion

IPv4 has the big problem of having not enough unique addresses. This led to parsimonious address allocation, which hinders the scalability of networks. Further, public IPv4 addresses became financially expensive. For example, there are IP-capable computers available that are cheaper than a public IPv4 address (see [72]). Often, merging multiple separate networks requires NAT or renumbering the nodes, because the address ranges overlap. With the vast address space of IPv6, those problems will virtually never occur.

IPv6 [5] offers a quite trivial solution to the IPv4 address exhaustion: It extends the address length. Instead of 32 bit, IPv6 addresses are 128 bit long.

IPv6 addresses are so long, there will be enough addresses forever (assuming they are used as expected by the current standards). This can be illustrated in various ways (see e.g. [73]). For example, assuming the entire IPv4 address space were consumed at a constant rate from the year of the definition of IPv4 until the year when the first RIR ran out of addresses (see [4], [59]):

$$\frac{2^{32}\text{IPv4 addresses}}{\text{minutes between 1981 and 2015}} = \frac{2^{32}\text{IPv4 addresses}}{1.788 \cdot 10^{7}\text{minutes}} \approx 240.2\frac{\text{IPv4 addresses}}{\text{minute}} \tag{3}$$

At 10 times that rate, allocating IPv6 addresses since the big bang[14] could have been satisfied with only half of the IPv6 address size:

$$\log_2\left(10 \cdot 240.2 \frac{\text{IPv6 addresses}}{\text{minute}} \cdot 7.2 \cdot 10^{15}\text{minutes}\right) \approx 63.91 \tag{4}$$

This calculation makes extreme simplifications. But it should give an idea of how vast the IPv6 address space is.

3.3.2 Abolishing NAT

NAT is useful, because it fixes the problem that every computer can get an address. But it can not maintain address uniqueness. Thereby NAT is rather treating the symptoms than the cause. IPv6 abolishes the need for NAT. Simply by having enough addresses available, they do not need to be reused, which was the main motivation for using NAT. There are several more reasons to avoid NAT (also see [75]):

- NAT increases the complexity of a network, which makes the operation costlier and more prone to human errors.
- Especially NAPT operates stateful, because it needs to track the state of sessions. If the router doing the NAPT looses its state (e.g. due to a failover to a new NAPT router), the sessions will loose connection.
- NAPT alienates address space from transport layer protocols. This blurs the border between the OSI layers and makes the transport layer address space shared among the nodes on the private side of the NAPT.
- A UDP connection does not have a state on its own. To determine what is a UDP session, timers or information from even higher level protocols have to be used.

3.3.3 Other Improvements

In contrast to IPv4, IPv6 does not reserve the last address of every network for broadcasting traffic to it. As the usual network size allocates 64 bits to identify hosts, reserving one addresses would not reduce the number of available addresses noticeably. But it makes using the network easier. IPv6 replaces broadcast with multicast. Every node can join multicast groups, and traffic to a multicast address is routed to all members of that group. The multicast addresses are allocated from a dedicated range of IPv6 addresses. Every node joins a multicast group together with its neighbors, which effectively provides a link-scope broadcast.

The IPv6 header has a fixed length of 40 bytes. (The IPv4 header has a variable length.) Optional information is implemented using *extension headers*, which are additional headers appended after the IPv6 header. This enables more efficient processing.

IPv6 has no checksum. Because the lower and higher layers already (can) validate the integrity, there is no need to repeat this in IPv6. In the UDP, the originally optional checksum was made mandatory.

[14] Assuming the big bang was $7.2 \cdot 10^{15}$minutes $\approx 13.8 \cdot 10^{9}$years ago, see [74].

IPv6 routers no longer need to handle fragmentation, this is now the duty of the endpoints of the communication.

3.3.4 Conclusions

IPv6 introduces many changes, most prominently the extension of the address length. The changes affect the way IPv6 is used (e.g. no NAT, multiple addresses per interface, SLAAC). This in turn requires network administrators to learn and implement the new behavior.

There are network administrators of large networks that chose to migrate to IPv6-only. For example, Microsoft is migrating its campus networks to IPv6-only (see [76]). Their motivations include having enough addresses available, avoiding overlapping address spaces (such from RFC 1918) and avoiding the operational complexity of Dual-Stack. The problems they encountered when attempting to move to IPv6-only include lack of supported features by individual vendors and people not understanding IPv6 or not prioritizing the migration to it.

The IT infrastructure of the federal administration of Germany aims to migrate to IPv6-only networks (see [77]). The IPv4 address exhaustion and the worldwide migration towards IPv6 are named as justification.

Considering the usecases PXE-DHCP and RMC-DHCP (see section 1.1), the requirements can be narrowed down. The nodes in these groups are predictable, so vendor support for IPv6 only needs to be assured for a limited amount of device types. The nodes in those groups can reach everything they need directly via company-internal networks or via a Dual-Stack proxy. Thus, they could be migrated to IPv6-only without needing NAT64 and DNS64, an IPv4-in-IPv6 tunnel or Dual-Stack.

The *United Internet AG* is currently running into issues because merging IPv4 networks of different businesses requires renumbering nodes because of duplicate IPv4 addresses. Also some IPv4 networks outgrow their size, meaning some nodes are left without address, until more addresses are allocated to the network. Enlarging a network sometimes requires renumbering nodes, because the neighboring addresses are already allocated.

In sum, a migration to IPv6-only network is advisable, especially in the long run.

4 Stateful DHCPv6 and Redundancy Issues

This section studies issues that may occur when operating multiple stateful DHCPv6 servers for the same RMC-DHCP clients (see section 1.1.2). The focus lies on leasing Identity Associations for non-temporary addresses (IA_NA, see section 2.8.7) to the clients. The DHCPv6 servers have to create and delete DNS records according to their knowledge of the address state and hostname of the client. Consistency of other stateful or stateless configuration information is not analyzed to reduce the scope of work. The standard policy for handling inconsistent stateless configuration information is mentioned in section 2.6.

Section 4.1 explains how knowing about the assignment of an address to a node is difficult. Section 4.2 goes into detail about possible issues when maintaining DNS records for RMC-DHCP clients. A caching policy for those records is proposed in section 4.3. Several methods to operate multiple redundant DHCPv6 servers are investigated in section 4.4. Mechanisms to synchronize state between multiple DHCPv6 servers are not specified here because they are implementation-dependent.

4.1 Knowing That an Address Is Assigned

The Internet Protocol version 6 requires its addresses to be unique across their scope. ICMPv6 includes the algorithm Duplicate Address Detection (DAD, see 2.4.1) to ensure the uniqueness of an addresses among the nodes on the link. DAD is not completely reliable, but the chances of not detecting an address that is already in use are low.[15] This algorithm must be performed before an address becomes valid, regardless of the used configuration mechanism. Hence, having the same address in the same zone twice (or more) is unlikely.[16]

It should be noted that DAD can only be used by nodes at the questioned link and for addresses configured on interfaces. DHCPv6 servers and authoritative nameservers maintain lists of addresses that may differ from the actual addresses present at a link. A DHCPv6 client will inform the server if DAD reports a duplicate address, so the server can update its information. The DHCPv6 server could then update the authoritative nameserver appropriately.

Because an entire DHCPv6 server could fail, or the message transmission between DHCPv6 server and client, the information about the addresses at the server may become (partially) invalid. The same could be said about the synchronization from the DHCPv6 server to the authoritative nameservers, but that falls out of scope for this work. Therefore, the following sections analyze how the information at the DHCPv6 servers can be kept up to date when operating multiple servers.

4.2 Difficulties When Altering DNS Records

This sections investigates some possible problems concerning DNS records for RMC-DHCP clients. Figure 13 provides an overview. It is assumed that multiple redundant DHCPv6 servers

[15] DAD might incorrectly indicate uniqueness if the interfaces connected to the link change while DAD is running.

[16] Algorithms for choosing addresses (e.g. for SLAAC or DHCPv6) should generate different addresses for different interfaces. Otherwise, they could produce duplicate addresses. Those could be detected with DAD, but could be avoided by the algorithm.

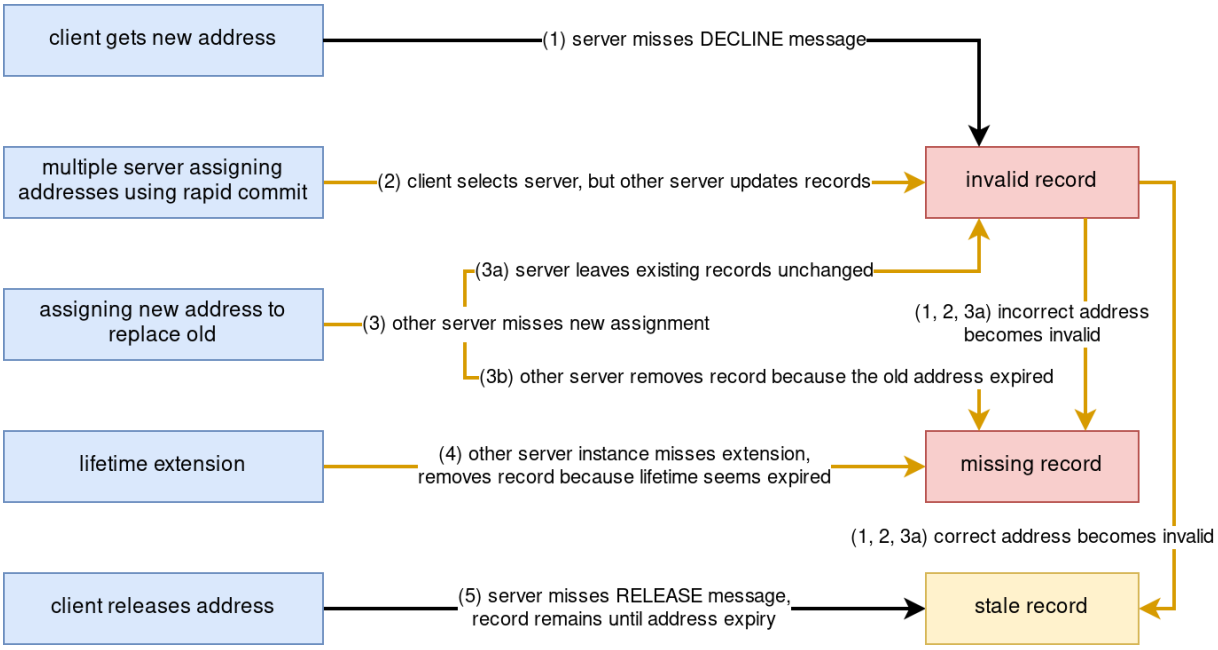


Fig. 13: This shows some problems concerning DNS records of RMC-DHCP clients and in what circumstances they might occur. Some client-initiated events (blue boxes) will lead with certain constraints (arrows) to problems (red/yellow boxes). Orange arrows indicate that multiple servers are involved. The red boxes represent persistent problems, the yellow box stands for a temporary problem. This diagram is explained in section 4.2. The numbers in the diagram match the numbers in that section. The events "incorrect address becomes invalid" and "correct address becomes invalid" can happen in multiple situations.

assign temporary or non-temporary addresses to clients and update DNS records (`AAAA` and `PTR`) accordingly.[17] The records for a client may get into these problematic states:

invalid The records point to a wrong target. This should be avoided entirely as it undermines trust in the correctness of the DNS records.

missing The records should be configured, but are not. A record is not supposed to be in this state for long because it impacts the availability of the client.

stale The records point to a target that no longer exists. Such records should be removed eventually to reduce obsolete information.

The following situations, numbered like in figure 13, will bring DNS records into one of the named problematic states:

1. Every time the server assigns a new address to the client, the client has to perform Duplicate Address Detection. If a duplicate address is detected, the client sends a `DECLINE` message to the server. It attempts by protocol default four times to transmit the message.[18] The server would confirm that with a `REPLY` message. If the server does not get the `DECLINE` message, it will falsely assume that the address is now assigned to the client and create or update the DNS records accordingly. Because the client can not be reached under the address, the DNS records are *invalid*. A server could miss `DECLINE` messages if a redundancy mechanism transmitted the messages to a different server that ignored them because the Server Identifier option did not contain its DUID. If the server receives a `DECLINE` message after it created or updated the DNS records, those would be temporarily *invalid* until the server reverts the change. After declining an address, a client will probably request a new address which would lead to situation number 3.

2. If a client requests a new address using the rapid commit mechanism, multiple servers could assign different addresses to the client without awaiting confirmation. Because they do not know which address the client chose, they are unable to determine the address to which the DNS records need to point. Without special implementation-dependent behavior of client and server, an *invalid* record might end up in the nameservers. If the valid lifetime of the address assigned to the client outlasts the valid lifetime of the address for which DNS records exist (perhaps because the client extended the lifetime), the DNS records will get deleted and thereby become *missing*. If vice-versa the address of the client becomes invalid first, the DNS records become *stale*.

3. If a client requests an address from server A, and then a second address from server B during the valid lifetime of the first address to replace it, server A will not know about the second address. This information could be synchronized between the servers, but this scenario assumes it is not. The client might request a new address because it moved to

[17] The anticipated infrastructure also requires `CNAME` records that are added and removed together with the `AAAA` records. Because `CNAME`, `AAAA` and `PTR` records are created, updated or deleted together, they are simply referred to as *DNS records* in the remainder of this section.

[18] See variable `DEC_MAX_RC` in RFC 8415 [9].

a new link with a different prefix. When updating the DNS records, server B has these options:

(a) If server B assumes the existing DNS records belong to a *different client* because of a duplicate address or FQDN, it should refrain from updating the DNS records, which would leave them valid for the existing client but invalid for the new client of server B. Indeed encountering a different client is unlikely, because duplicate addresses are prevented using DAD and FQDNs are formed in the intended usecase using serial numbers. Solving a duplicate address or FQDN could require manual intervention.
Because in this scenario the existing and new clients are identical, the DNS records simply become invalid. Like in issue 2, depending on whether the old or new address becomes invalid first, the DNS records end up in the *missing* or *stale* state.

(b) If server B assumes the existing DNS records point to the *same client*, it updates them to the new address. If the existing DNS records would actually belong to a different client (which is unlikely, as stated before), they would become invalid for that. When the valid lifetime of the address of either the existing or new client expires first, the DNS records will be deleted and become missing for the other. The DNS records will also become missing when the clients are the same and the valid lifetime of the old address expires, because server A will delete the DNS records, unbeknownst the still valid new address. Therefore, when a valid lifetime expires, a server should assert the DNS records match the address known to it, before deleting them.

4. This issue is similar to the one before, but a valid lifetime extension is missed instead of a new address assignment. With multiple redundant servers, server A could become temporarily unavailable. During that downtime, server B could receive a `RENEW` or `REBIND` message through which the client extends the valid lifetime of an address. If server A becomes available again after that, it should be informed about the updated lifetime. If that does not happen, server A will eventually assume that the valid lifetime expired, whilst the client and server B know that it did not expire yet. Server A will delete the DNS records, putting them in the *missing* state.
This issue exists likewise with DHCPv4 and has been observed in the current DHCPv4 architecture of the United Internet AG.

5. When a server does not receive a `RELEASE` message, it will refrain from deleting DNS records although it should, which creates *stale* records. A server could miss `RELEASE` messages like `DECLINE` messages if a redundancy mechanism transmitted the messages to a different server that ignored them because the Server Identifier option did not contain its DUID. This issue is resolved when the valid lifetime expires, because that triggers the server to delete the DNS records.

4.2.1 DHCID RR

A goal of the DHCPv6 architecture for RMC-DHCP clients is to maintain a one-to-one relationship between client and FQDN. The actual relationship is more complex: An address of an

interface of a client is mapped to (using a `PTR` record) and from (using an `AAAA` record) an FQDN.

The RFCs 4701 [78], 4702 [79], 4703 [80] and 4704 [81] specify a mechanism to avoid some issues with maintaining DNS records for DHCP clients. They define a new DNS resource record type called `DHCID` that creates a direct mapping between a client and an FQDN.

The DNS resource record `DHCID` stores a cryptographic hash (SHA-256) of an DHCP client identifier and the matching FQDN. The RFC specifies three possible types of client identifiers: The DHCPv6 client DUID, the DHCPv4 Client Identifier option value (see [82]) and the concatenated `htype` and `chaddr` fields of a DHCPv4 `DHCPREQUEST` message. As per RFC 4361[43] nodes should use the same DUID for the DHCPv4 Client Identifier option as for the DHCPv6 Client Identifier option, which enables to use a single `DHCID` record for DHCPv4 and DHCPv6.

Regarding the issues discussed earlier, the `DHCID` record does only affect issue number 3. The `DHCID` record helps to preserve any preexisting DNS records of other clients when attempting to create new DNS records.

4.2.2 Conclusions

The analyzed issues show the necessity to synchronize the information about the assigned addresses and their valid lifetimes when operating multiple redundant stateful DHCPv6 servers. The redundancy mechanism should take into account that the servers will only accept messages having their DUID, if they include a Server Identifier option.

Only one server at a time should offer the rapid commit mechanism.

When the valid lifetime of an address expires, the server should assure the address is the target of the DNS records before deleting them.

The `DHCID` record type could be used to detect duplicate addresses or duplicate FQDNs and to help investigate further issues.

4.3 Difficulties When Querying DNS Records (Caching)

In the RMC-DHCP setup the DNS records should be reported by DNS resolvers from when the associated leased address becomes valid up until it becomes invalid.

The previous section explains scenarios in which such changes may result in invalid, missing or stale DNS records at the authoritative nameserver(s). Even if those problems were solved, a DNS resolver still may experience invalid, missing or stale DNS records due to cached information. This section analyzes for how long caching of DNS records should be permitted.

4.3.1 Record-Changing Events

A DNS resolver should update its cache whenever the information could have changed. The time until the next change to a DNS record may happen is governed by the following events:

- When the valid lifetime of an address expires, its DNS records get deleted. The valid lifetime may be prolonged, but not shortened. DNS records that are cached beyond that event are stale or even invalid, when the FQDN gets a new address assigned.

commit	345841
expiry	70001
release	23

Tab. 8: The number of ISC-DHCPD events logged for RMC-DHCP clients by servers of the current DHCPv4 architecture within a 72 hour interval. See section 4.3.2 and section 6.1.

- The `RELEASE` message can be sent by a DHCPv6 client to relinquish an address. After the address was released, a server will remove the DNS records. Because this is initiated by the client, the server can not predict, when it will happen. A cached DNS record would be stale then. After the release, a server may reassign the address. The cached DNS record would then become invalid, which is more problematic.
- When a new address gets assigned to a client (perhaps because it moved to a different link), the DNS records get updated. The time between the assignment and the change, as well as the likelihood of such a change depends on the individual usage and can not be predicted in general. A cached DNS record would be invalid after the update.

4.3.2 Practical Relevance of `RELEASE` Messages

To estimate how often a DHCPv6 client will release an address before expiration, the current DHCPv4 architecture for RMC-DHCP clients is investigated. The events logged by the servers within a 72 hour time interval between 2020-06-06 and 2020-06-09 are analyzed using the *management service database* (described in section 6.1). Only a fraction of the events are `RELEASE` messages, as shown in table 8. Of those, the delay between *commit* and *release* event varies across the common lifetime of 24 hours. Two leases were released after over 22 hours, four after almost 12 hours and twelve after less than two minutes. The delay could not be calculated for five more *release* events, because the corresponding *commit* events were not in the dataset.

During the 72 hours of the considered data, at least three IPv4 addresses were reused for different clients.

The data shows that release events are rather seldom. Nonetheless, they should be expected any time during the valid lifetime. The considered data does not contain enough release events to suggest a most frequent time after which release messages can be expected. It is questionable whether the IPv4-specific behavior of the RMC-DHCP clients will be the same for IPv6.

4.3.3 DNS TTL recommendation

DNS relies heavily on caching records. The Time To Live (TTL) is attached to every DNS record and specifies the number of seconds that record may be cached (see [11]). It should be maximized to reduce the traffic on the nameservers while it should be minimized to let the clients pick up changes (and deletions, with lesser priority) as soon as possible.

A deletion may occur after the valid lifetime expires or upon a `RELEASE` message. If the client does not request a new address, the DNS records can only change after a deletion. The likelihood of a DHCPv6 server directly reassigning an address to a new client depends on the address allocation policy of the server implementation.

The DHCPv6 standard permits a server keeping track of previously assigned addresses to reassign them to the same client in the future (compare RFC 8145 section 18.3.7 [9]). Thus, a server could be programmed to avoid reassigning addresses for which a `RELEASE` message was received until the valid lifetime would have expired. Considering the vast address space in the intended /64 networks, it may not be necessary to reuse addresses *ever*. Then, the only limit to the TTL would be the permitted age of stale DNS records. However, this approach is presumably limited in practice by storage complexity.[19]

The current DHCPv4 architecture creates DNS records for RMC-DHCP clients with a TTL of 10 minutes.

As a recommendation, the TTL should be set to the minimum of the following three values:

- the valid lifetime
- the time after which an unassigned address may be reallocated to a different client
- the time after which the client may have been assigned a new address

The first two values can be configurable in a DHCPv6 server implementation.

To assess this recommendation, the number of DNS requests at the authoritative nameservers, the number of stale records and the storage and performance impact on the DHCPv6 servers should be monitored while trying different TTL values. This assessment is left for future work.

4.4 Issues With Failovers and Synchronization

This section analyzes what issues might arise when a DHCPv6 server goes down and a second server takes over.

The described event of switching to another server is analyzed for the following synchronization scenarios:

no-synchronization The servers synchronize nothing. They have identical configuration, but know nothing about each other. Their DUIDs and lease information differ.

lease-synchronization The serves synchronize their leases but have different DUIDs.

comprehensive-synchronization The servers synchronize everything. The configuration, lease information and server DUID is ensured to always be the same on all servers using an out-of-scope protocol between the servers.

The messages sent by clients can be distributed in various ways across the available servers. The following three mechanisms are analyzed:

multiple-servers The clients see both servers. Multicast traffic goes to both servers.

dynamic-routing The servers share the same IPv6 address. Traffic to that address is dynamically routed using BGP. Multicast traffic goes only to that address.

[19] Storing for each address of a /64 network whether it has been assigned using an uncompressed bit array would require two exbibytes of storage.

<table>
<tr><th></th><th>comprehensive-synchronization</th><th>lease-synchronization</th><th>no-synchronization</th></tr>
<tr><td>multiple-servers</td><td>multicasted messages are processed multiple times by the same logical server</td><td colspan="2" rowspan="3">messages with Server Identifier option will not be processed after a failover</td></tr>
<tr><td>dynamic-routing</td><td rowspan="2">no transmission issues</td></tr>
<tr><td>DUID-mapping</td></tr>
</table>

Tab. 9: Depending on how bindings are synchronized between DHCPv6 servers and how messages sent by clients are distributed to them, the messages sent by clients might not get processed (appropriately). The issues with comprehensive-synchronization and multiple-servers are explained in section 4.4.1, the issues with lease-synchronization or no-synchronization in section 4.4.2.

DUID-mapping A Relay Agent performs loadbalancing across the servers. Each client DUID is mapped to a fixed server. If that server fails, the mapping changes to a different server. Multicast traffic goes only to the selected server.

Those synchronization scenarios and client message distribution mechanisms could be combined arbitrarily, but some combinations can disturb the message exchanges. Table 9 provides an overview over the combinations and the following sections explain the issues with them.

4.4.1 Too Much Redundancy

The messages `SOLICIT`, `CONFIRM` and `REBIND` are always multicasted by clients. If they are received by multiple servers that use comprehensive-synchronization, they might run into internal conflicts or each sends the same message to the client. The behavior of the clients and servers in such a configuration is implementation-dependent.

4.4.2 Refusing Client Messages

The messages `DECLINE`, `RELEASE, RENEW` and `REQUEST` always need to include a Server Identifier option. `INFORMATION-REQUEST` messages *can* include a Server Identifier option. A server will ignore all messages that contain a Server Identifier option, but not its DUID. This can happen when multiple servers use different DUIDs but are seen by the client as a single server.

When a server ignores a client's message, the client will retransmit the message. When and how often a client retransmits a message depends on the message type (see section 2.8.4). Until the retransmission terminates and the client proceeds without Server Identifier option (e.g. because it is restarting the configuration process using a `SOLICIT` message), it can not complete the configuration process. The client behavior after vainly transmitting messages with Server Identifier option is explained hereafter.

`DECLINE` A client should proceed to use other configuration information and any addresses that were not detected as duplicate. The client can restart the configuration process by multicasting a `SOLICIT` message. This is independent of the success of the `DECLINE-REPLY`

message exchange. The consequences of a missed DECLINE message are explained from a server perspective in section 4.2.

INFORMATION-REQUEST When responding to a RECONFIGURE message, the client copies the server DUID from that to the INFORMATION-REQUEST message. Thus, the client will not get new configuration, when the server does not accept this DUID. Regardless of the attempted reconfiguration message exchange, the client may send an INFORMATION-REQUEST message without Server Identifier option some time later. A server can indicate the duration after which the client should do so using the Information Refresh Time option in REPLY messages in response to INFORMATION-REQUEST messages.

RELEASE The configuration of a client does not depend on the success of a RELEASE-REPLY message exchange, so no further actions are required by the client. The impact of a missed RELEASE message is explained from a server perspective in section 4.2.

RENEW When T2 expires, the client will switch from sending RENEW messages with Server Identifier option to multicasting REBIND messages to all servers, to achieve the same. If the other servers are not able or willing to extend the lifetimes of the leases, the leases expire eventually and the client has to restart the configuration process using a SOLICIT message.

REQUEST After the client gives up retransmitting REQUEST messages, its actions are implementation-dependent. It may attempt sending REQUEST messages to other servers if known, restart the configuration procedure by sending SOLICIT messages or simply remain not configured.

4.4.3 Conclusions

A DUID stands for a single server (or client). Multiple servers should only use the same DUID, if multicasted traffic gets routed to only one instance.

If client messages get dynamically routed (e.g. using BGP) or dynamically relayed (e.g. using DUID-mapping) to a single server instance, the server instances should share a common DUID. Otherwise, they will not get processed (until the client addresses all available servers). This delays the configuration of the client or notification of the server that an address is not used by the client.

5 DHCPv6 Server Implementations Running on Linux

This section presents some DHCPv6 server implementations. The requirements for selecting an implementation to be used at the *United Internet AG* are discussed in section 5.1. A list of possible implementations is presented in section 5.2 and explained in section 5.2.

5.1 Criteria for Selecting Implementations

To reduce the scope of work and not require proprietary operating systems or hardware, only implementations that can be installed on Linux are considered. To be able to understand and modify the implementation, it should be distributed under an open source license (see [83]).

The implementation must conform to the newest DHCPv6 standard (RFC 8415) for the features it implements. The implementation must be "stable enough" for a production environment, meaning the software quality and feature completeness.

The implementation should be...

- under active support (recent changes of the source code, commercial support, popularity).
- implementing the required options for the other configuration information specified in table 12.
- offering some interface to create, update and delete DNS records according to the leases.
- able to share or replicate its state (assigned Identity Associations including lifetimes) with other instances.
- offering interfaces to monitor the properties specified in section 6.3.5.
- able to update its configuration without downtime.
- capable of handling a sufficient number of client messages per second.
- able to reserve an address for a configurable time before assigning it to a different client.
- able to vary the lifetimes of addresses and T1 and T2 variables of Identity Associations to distribute the renewal attempts of clients over time.
- supporting the DHCPv6 *reconfigure* mechanism (see section 2.8.8).

5.2 Open Source DHCPv6 Servers Running on Linux

The implementations of DHCPv6 servers listed in table 10 are discussed in this section.

The development of *Dibbler* has been officially discontinued. The software project *wide-dhcpv6* has shown no activity in the last five years. Therefore, both are not suitable for a production environment.

The implementations *ISC DHCPDv6* and *Kea* are developed by the *Internet Systems Consortium, Inc.* (ISC). The former is popular and integrated in various other software, but it does

name	support / development activity	license	reference
coredhcp	last commit 2020-08-09, documentation says "This is still a work-in-progress"	MIT (dhcp libary: BSD-3-Clause)	[84], [85]
dhcp6	last change 2019-03-11	MIT	[86]
dhcplb	last commit 2020-07-18, in use by Facebook	MIT (dhcp libary: BSD-3-clause)	[85], [87]
dhcpy6d	last commit 2020-07-24, last release on the same date	GPL 2.0	[88]
Dibbler	not maintained	GPL 2.0	[89]
dnsmasq	last commit 2020-07-19, last release on the same date	choosable: GPL 2 or GPL 3	[90]
ISC DHCPDv6	only security fixes, no new features, developers recommend to use Kea	MPL 2.0	[91]
jagornet dhcp	last change 2020-05-12, last release 2016-07-07	GPL 3	[92]
Kea	commercial support, purchasable features	MPL 2.0	[93]–[95]
ndhs	last change 2018-02-09	BSD-2-clause	[96]
wide-dhcpv6	last change 2015-07-13	BSD	[97]

Tab. 10: Those DHCPv6 server implementations are open source and can be run on Linux. The times of the most recent changes are based on the public version control repositories and were researched on 2020-08-28.

name	language	implemented RFCs	partially implemented RFCs	DHCPv4 server	DHCPv6 relay agent
coredhcp	Go	not documented		yes	no
dhcp6	Go	3315		no	no
dhcplb	Go	not documented		yes	yes
dhcpy6d	Python		3315, 5908, 4291, 3646	no	no
Dibbler	C++	3315, 2136, 2845		no	
dnsmasq	C	not documented		yes	yes
ISC DHCPDv6	C	3315, 3319, 3646, 3898, 4075, 4242, 4280, 4580, 4649, 4704, 4994, 2136, 4941, 3633	4701, 4702, 4703, 5942	yes	yes
Kea	C++	3319, 3646, 4242, 6422, 7550, 7598	3315, 3633, 3736, 4649, 4703, 4704, 6334, 6603, 6939, 8357, 8415	yes	no
jagornet dhcp	Java	3315, 3633, 3646, 3736, 4703, 4704		yes	no
ndhs	C++	not documented		yes	no
wide-dhcpv6	C	3315, 3319, 3633, 3646, 4075, 4242		no	yes

Tab. 11: Feature comparison of the DHCPv6 server implementations listed in table 10. The differentiation between “implemented RFCs” and “partially implemented RFCs” is based on statements of the respective documentation. The column ”DHCPv4 server” indicates whether the implementation includes or is accompanied by a DHCPv4 server. The column ”DHCPv6 relay agent” indicates whether the implementation includes or is accompanied by a DHCPv6 relay agent.

not support configuration changes without downtime and the ISC currently only provides security updates, no new features. The ISC recommends to use their other implementation instead. (See [98])

Kea appears to be the most comprehensive implementation of DHCPv6 (RFC 8415 and related standards). The Kea DHCPv6 server implements a failover and a loadbalancing mechanism, which is described in section 6.2.2. The Kea software includes a command line interface named *perfdhcp*, which can generate DHCPv6 messages to benchmark the performance of a DHCPv6 server (see [99]).

The servers *coredhcp* and *dhcplb* are build on the same DHCPv6 library written in *Go*.

6 Configuration Service Architecture Evaluation

This section explains the DHCPv4 architecture currently operated by the *United Internet AG* (section 6.1), presents related work about DHCPv6 architectures (section 6.2) and proposes an DHCPv6 architecture to provide configuration for PXE-DHCP and RMC-DHCP clients at the United Internet AG.

6.1 Current DHCPv4 Architecture

Currently, the *United Internet AG* operates an architecture of services that provides PXE-DHCP and RMC-DHCP for IPv4 clients (DHCPv4). This architecture is illustrated in figure 14 and will be explained hereafter.

The clients communicate with DHCPv4 servers to obtain their configuration. The servers share an anycast address and appear as a single server to the clients.

The architecture can allocate addresses as needed or provide clients, identified by their MAC address, with unchanging addresses ("fixed-address").

The architecture is split up into a management service, a DHCP service and other services that are out of scope for this work. An instance of the DHCP service can be either one of the flavors PXE-DHCP or RMC-DHCP. Only in the latter case, DNS records are updated based on DHCP leases. Apart from that, the flavors do not differ.

6.1.1 Management Service

The management service consists of multiple services and the *management service database* (msdb). This database contains a history of the events reported by the DHCP servers and all necessary information to configure the DHCP servers.

Administrators can query the msdb using a web-frontend or HTTP API and thereby see for example the active leases.

The DHCP service instances read their configuration information from the msdb via a dedicated HTTP API ("DHCP server configuration API"). The configuration is refreshed periodically in the msdb by the *msdb update service* based on metadata of the asset management software of the company and *DIM*. DIM ("DNS and IP Management" [100]) is an open-source software developed by the *United Internet AG* that is used to manage IPv4 addresses, IPv6 addresses and DNS records. For the scope of this work, the asset management software and DIM will be regarded as black-box HTTP APIs.

6.1.2 DHCP Server Configuration

Each DHCP service instance runs the DHCPv4 server implementation of the *Internet Systems Consortium, Inc.* (ISC-DHCPD [91]), a database, a *configuration service* and an *event propagation service*.

The configuration of the ISC-DHCPD, including address pools and fixed assignments of IPv4 addresses to MAC addresses, is updated periodically every half hour by the configuration service. This service fetches information from the *DHCP server configuration API*, generates

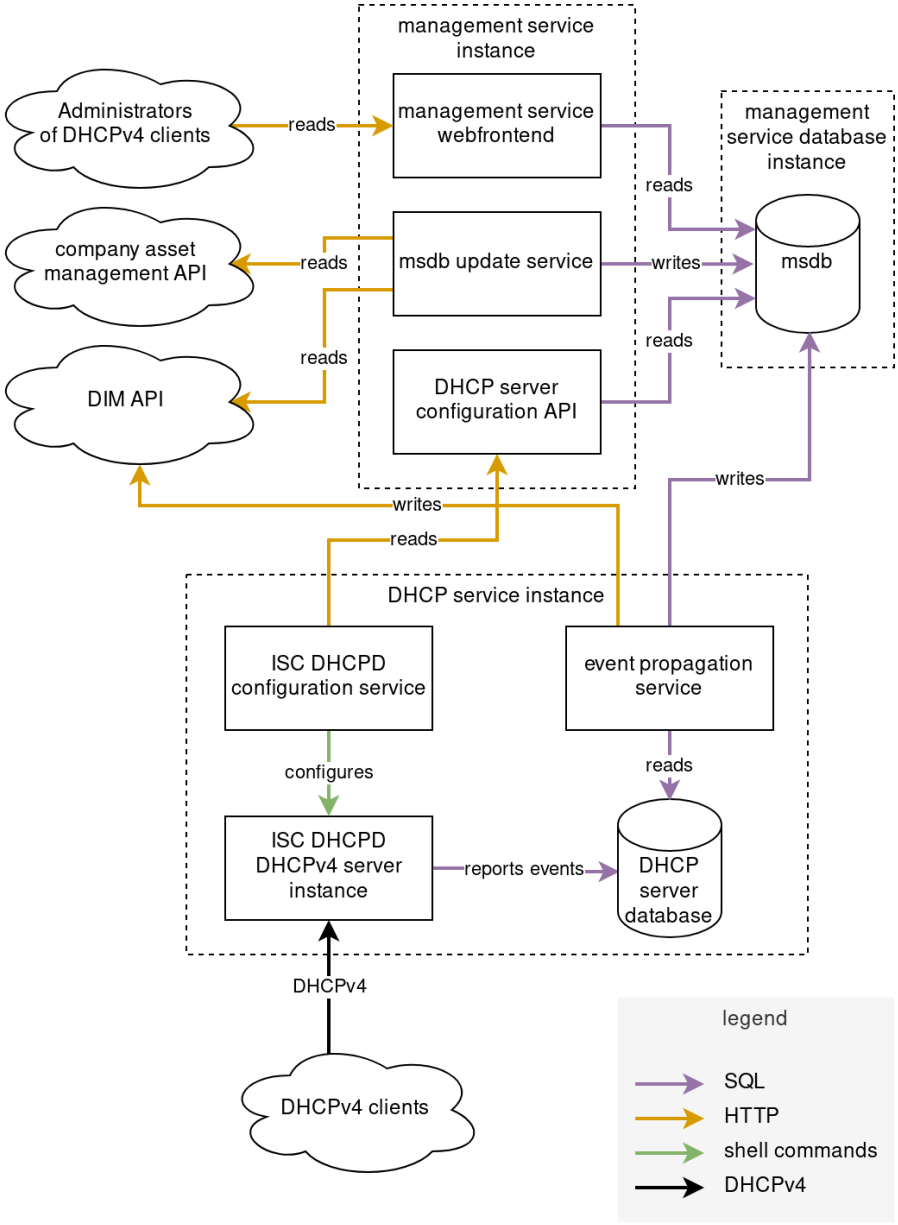


Fig. 14: This is the software architecture currently used by the *United Internet AG* to supply RMC-DHCP for IPv4 clients. Each box with dashed border represents one of multiple redundant hosts. A host runs programs (boxes with solid borders) and databases (cylinders). The redundant hosts are accessed via anycast addresses (using BGP). The arrows are oriented to point *from* the one that initiates the communication. This architecture is explained in section 6.1.

DHCPv4 option
range (lower and upper IP address)
default lease time (seconds)
maximum lease time (seconds)
DNS search list
list of recursive DNS servers
netbios nameservers
dynamic DNS domain
boot image server ("next-server")
boot image filename (CPU architecture dependent)

Tab. 12: These options can be configured in the current service architecture (described in section 6.1). They are managed in DIM and can be set per *pool*. A pool consists of IP subnets, these options and other options. Only the *range* is a required attribute. The *dynamic DNS domain* is appended to a hostname to form an FQDN for assigned addresses (in case of RMC-DHCP). Different boot image filenames can be specified for different DHCP client CPU architectures.

configuration files and restarts the ISC-DHCPD to apply the changes. The information that can be configured using the current architecture is listed in table 12.

An ISC-DHCPD reports the events *commit*, *expiry* and *release* into its DHCP server database. This is a *PostgreSQL* database [101], which is run on the same host as the ISC-DHCPD for latency and availability reasons.

The *commit* event is triggered every time the ISC-DHCPD assigns a client an address or extends the time that address may be used. The *expiry* event is triggered whenever the lifetime of a leased address expires without prolongation by the client. When the server marks an allocated address as available again because it received a `DHCPRELEASE` message by a client, a *release* event is triggered.

Those events are written into the local database as they occur. At most five seconds later, the *event propagation service* attempts to copy the events to the msdb. If configured to do so (RMC-DHCP), the event propagation service creates, updates or deletes DNS records appropriately by using the DIM HTTP API when the assignment state of an address changes.

6.2 Related Work

This section presents related work to DHCPv6 in data centers. No scientific work directly addressed to this topic was found during the research for this work. However, *Facebook* has published details about how they use DHCPv6 in their data centers (see section 6.2.1) and the *Internet Systems Consortium* has published work on how their DHCPv6 server implementation *Kea* implements failover and loadbalancing mechanisms (see section 6.2.2).

6.2.1 DHCPv6 Deployment at Facebook

Facebook, Inc. is using DHCPv6 (and DHCPv4) as configuration mechanism for IP nodes in its production data centers (see [102]–[104]). They claim to use it for network booting and out-of-band management interfaces, which precisely matches the client groups PXE-DHPC and

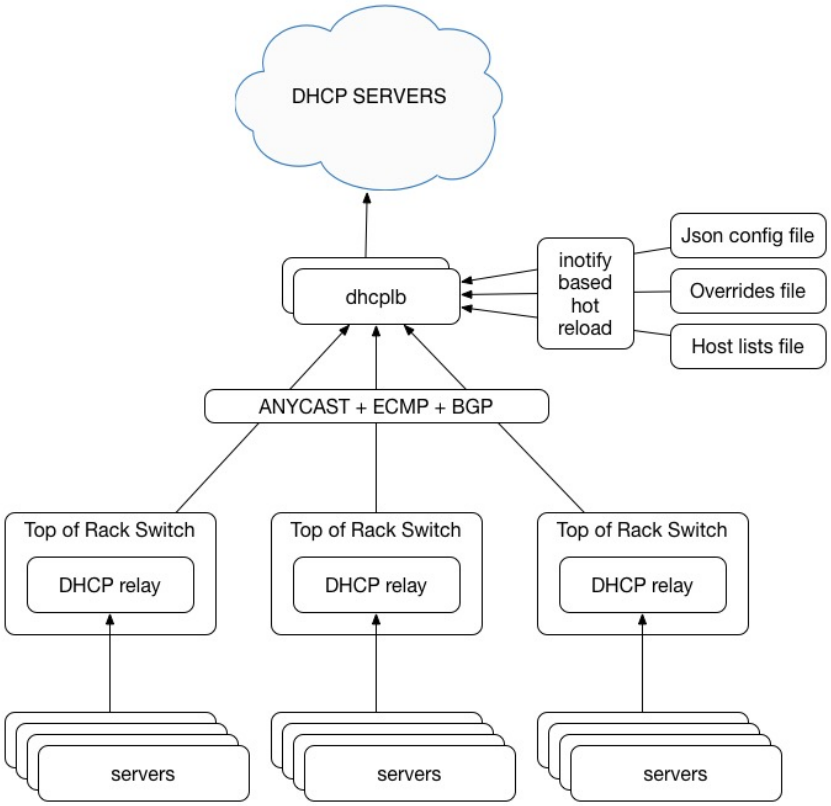


Fig. 15: The deployment of the DHCP relay *dhcplb* used by *Facebook, Inc.*, see section 6.2.1. (image source: [87])

RMC-DHCP that are targeted in this work.

In a blogpost from 2019 [102], Facebook explains how their DHCP architecture evolved. They started with the DHCP server *ISC-DHCPD* (see [91]), but moved away from that because altering its configuration requires a restart that reduces availability. Configuration changes became more frequent and eventually the ISC-DHCPDs were spending more time restarting than serving traffic.

After ISC-DHCPD they switched to Kea, because it enables to centralize the configuration data and execute custom code as specific events occur ("hooks"). Facebook ran multiple instances of Kea and advertised them as a single anycast address using BGP. (see [102], [104])

However, that Anycast setup was not satisfying for Facebook, because the BGP anycast setup did not distribute the traffic evenly. To solve that, they developed a DHCP relay and named it *dhcplb* (see [87]). This relay is capable of load-balancing the traffic between multiple servers. The server selection algorithms *Round Robin* and *Modulo* are implemented. The latter calculates a hash of the DHCP client DUID and maps that to a server by taking the modulo of the hash. Additionally, dhcplb allows A/B testing of servers by categorizing the servers into two groups and selecting a group based on configuration. Facebook runs dhcplb itself in multiple instances that are accessed by an BGP anycast address. The DHCP clients reach the dhcplb instances through another relay, as shown in figure 15.

The next problem Facebook ran into was the single-threaded design of Kea. When Kea was

hit with more traffic than it could keep up with (that boundary is especially limited by calls to other services), it started to drop packets. The developers of Kea have recently released a stable version of Kea with multi-threading support (see [105]). Instead of waiting for that, Facebook extended dhcplb to make it a DHCP server and replaced Kea with that. Obviously, dhcplb was programmed with multi-threading support and could handle measurably more traffic. Also, the configuration of dhcplb can be updated without downtime like ISC-DHCPD. (see [102])

Discussion The configuration mechanism architecture of Facebook presumably serves a much larger number of clients than the anticipated usecases at the United Internet AG. Copying their infrastructure might provide their performance and scaling optimizations, but it might be overly complex.

6.2.2 ISC High Availability Considerations

The *Internet Systems Consortium, Inc.* (ISC) develops the two DHCP server implementations *ISC-DHCPD* [91] and *Kea* [93]–[95], both for DHCPv4 and DHCPv6.

The DHCPv4 implementation ISC-DHCPD does support a failover mechanism[20], but the DHCPv6 implementation of ISC-DHCPD does not. As the ISC shifted its resources from developing ISC-DHCPD to developing Kea, it is unlikely that this DHCPv6 implementation will support a failover or loadbalancing mechanism in the future. In contrast, the Kea DHCPv6 (and DHCPv4) implementation does support Kea-specific failover and loadbalancing mechanisms. (see [107])

Therefore, the following sections only consider the Kea DHCPv6 server implementation.

Redundancy Features of Kea Kea (as of version 1.8.0) supports running servers without redundancy mechanism, in loadbalancing mode, in hot-standby mode or in cold-standby ("backup") mode. Those redundancy mechanisms are explained in the Kea wiki by the developers for Kea version 1.4.0 (see [108]) and are summarized hereafter.

Loadbalancing and failover is implemented for exactly two Kea servers. More servers may be added in *backup* mode, which makes them synchronize the state of servers in hot-standby or loadbalancing mode. The backup servers are not answering to DHCP client messages and need manual configuration to do so. The two servers not in backup mode automatically detect and react to a failure of their partner server.

Kea servers may share a database in which leases are stored. To avoid just shifting the single point of failure to the database, the database should be operated with multiple redundant instances.

If the servers do not share a database for leases, they exchange updates about the leases over a Kea-specific API. This is done synchronously, meaning the answer to the client is sent after the update at the other server was successful. If the update failed, no answer is sent to the client.

The documentation of Kea 1.8.0 explains the decision against implementing the DHCPv6 failover standard RFC 8156 [109] in Kea:

[20] ISC-DHCPD implements the DHCP Failover Protocol draft [106] as a DHCPv4 failover mechanism.

"The DHCPv6 failover standard (RFC 8156) was published, but it is complex, difficult to use, has significant operational constraints, and is different than its v4 counterpart. Although it may be useful for some users to use a 'standard' failover protocol, it seems that most Kea users are simply interested in a working solution which guarantees high availability of the DHCP service. Therefore, the Kea HA hook library derives major concepts from the DHCP Failover protocol but uses its own solutions for communication and configuration." (see [110])

The redundant Kea servers appear as individual DHCPv6 servers to the clients. Only if a client uses multicast (e.g. for `SOLICIT` messages), all servers receive the message. Each server determines, whether it is meant to process a message. If not, the message is dropped. This matches the redundancy mechanism described in section 4.4 as *lease-synchronization*.

Servers in cold-standby mode never respond to DHCP client messages. In hot-standby mode, only one of the two servers is configured to process messages. In loadbalancing mode, the decision to process a message is based on a client identifier, such that each server is assigned half of the possible client identifiers. The mapping from client identifier to server is implemented according to the *DHC Loadbalancing algorithm* as specified by RFC 3074 [111]. This matches the DUID-mapping method described in section 4.4. If one server detects the failure of the other, the remaining server begins processing all messages.

Failure Detection Two Kea servers can detect their partners failure using a Kea-specific protocol. They exchange heartbeat messages over that. If one partner vainly waits for a configurable time for a heartbeat message, it starts analyzing the client messages that would be handled by the other server. If enough (a configurable amount) client messages exceed a configurable value in the "elapsed time" DHCPv6 option[21], the other server is considered to have failed.

Recovery from Failure Once a server becomes available again, it can take up its work without manual configuration.

If the communication between the servers fails, the continue to operate but do not synchronize their leases. To recover from that, they need manual configuration.

Discussion Having only two servers in a redundancy relationship limits the way configuration service architectures can be built for larger services (e.g. when providing DHCPv6 for more than two data centers).

When a client gets mapped to a different server, its messages will be ignored if they contain a Server Identifier option (whose value will differ from the DUID of the new server). This problem is described in section 4.4.2.

6.3 Recommended Configuration Mechanism Architecture

This section proposes an architecture of services to configure IPv6 nodes at the *United Internet AG*. It is intended to replace the existing DHCPv4 architecture because a migration to IPv6-only is desirable (see section 3.3.4).

[21] When using DHCPv4, Kea examines the "secs" field.

The PXE-DHCP and RMC-DHCP node groups are treated differently because of different requirements. The architecture for stateful DHCPv6 (for RMC-DHCP) is illustrated in figure 16, and the architecture for stateless DHCPv6 (for PXE-DHCP) in figure 17.

Both architectures include redundancy and loadbalancing mechanisms to meet the high availability requirements of the anticipated usecases.

The implementation of the proposed architecture and the interaction of the individual parts with each other and with the existing infrastructure is not specified here, because the available implementations are not studied thoroughly enough.

6.3.1 Configuration Mechanism for PXE-DHCP Nodes

PXE-DHCP nodes require addresses and the *other configuration information* listed in table 12. Because they require more than just DNS configuration, SLAAC alone is not sufficient. Either SLAAC in combination with stateless DHCPv6 or stateful DHCPv6 could be used. The former could be achieved with a less complex architecture, because knowledge about the valid assignment of an address does not need to be synchronized from a node to a central database. Using SLAAC for PXE-DHCP nodes would come at the cost of not having a central database of their addresses. Since the intended usecase is to boot an operating system for a temporary maintenance, having a central database may not outweigh the benefits of a simpler architecture. In the following, it is assumed that SLAAC in combination with stateless DHCPv6 servers is desired for the PXE-DHCP nodes. Otherwise, the architecture for RMC-DHCP (perhaps without maintaining DNS records) could be used for PXE-DHCP nodes as well.

6.3.2 Configuration Mechanism for RMC-DHCP Nodes

The RMC-DHCP nodes require DNS records pointing to their addresses, meaning the addresses must be stored at the authoritative DNS nameservers. Thus, a "central database" is needed for the RMC-DHCP clients, and consequently, *stateful* DHCPv6 servers. The stateful DHCPv6 servers should assign addresses to the clients and create DNS records accordingly. They should also create `DHCID` records (see section 4.2.1). The stateful DHCPv6 servers should not offer the *rapid commit* mechanism to avoid consistency issues.

6.3.3 Routers and Relay Agents per Link

The clients in the considered usecases are distributed across many links, such that operating multiple DHCPv6 servers per link (redundancy requirement) is not feasible. At least one relay agent per link with clients is necessary. Those are called "client" relay agents here. If multiple relay agents are installed at a single link, only one of them should relay DHCPv6 messages at a time. They listen to the multicast address `ff02::1:2` and would thereby create redundant copies of the messages sent by clients. Because *the clients* send their messages to the multicast address, IPv6 anycast is not possible. The duplicated messages could be filtered out by following relay agents (if any) and the servers based on the transaction-id and the Elapsed Time option. Maybe a link-layer protocol could be used to repurpose the IPv6 multicast address into an anycast address.

The "client" relay agents are configured in a per-usecase (e.g. RMC-DHCP) uniform way. They implement RFC 6939 to add the link-layer address of the client to the message (see section 2.8.2 and [42]). Servers should not accept unicast connections from clients (meaning not to send a Server Unicast option) to force them through the relay agents which add the Client Link-Layer Address option. They relay the messages to an IPv6 anycast address (which can be implemented using BGP). In case of PXE-DHCP, that address is advertised by DHCPv6 servers, but in case of RMC-DHCP, the address is advertised by "loadbalancing" relay agents (see section 6.3.4).

In contrast to DHCPv4, routers need special configuration for DHCPv6. They need to send ICMPv6 Router Advertisement messages with the flags listed in table 13. Those ensure that PXE-DHCP clients use SLAAC to obtain their addresses and stateless DHCPv6 to obtain other configuration information, and RMC-DHCPv6 clients use DHCPv6 for both.

In case direct communication (not via a router) between clients at the same link is desired, the IPv6 prefix of the network needs to be advertised as *on-link* in the Router Advertisement messages. Considering the usecases, direct communication seems to be unnecessary. Thus, the routers could conceal the *on-link* property and block traffic between neighbored clients.

6.3.4 State Synchronization and Message Distribution

The DHCPv6 servers in the PXE-DHCP architecture operate only in *stateless* mode, which is why no special synchronization is needed and the architecture can be build simpler. The servers of the RMC-DHCP architecture operate in *stateful* mode and need to implement comprehensive-synchronization (synchronization of lease information and DUIDs, see section 4.4).

The "loadbalancing" relay agents receive client messages from "client" relay agents and send them to DHCPv6 servers. They distribute them based on DUID-mapping (see section 4.4). They keep a connection with each DHCPv6 server they relay to, to track whether they are operational. That way, a "loadbalancing" relay agent maintains a list of operational servers, and updates the mapping of clients to servers when that list changes.

6.3.5 Monitoring

In order to successfully operate the proposed service architecture, detailed knowledge about its state is required. At least the following properties should be monitored:

- The values of Elapsed Time options sent by clients, per server instance.
- The number of messages sent and received by servers and relay agents, grouped by the message type and Status Code option value, if applicable.
- The number of DNS requests at the authoritative nameservers that query the zones managed by the DHCPv6 servers.

flag	PXE-DHCP	RMC-DHCP
managed address configuration	0	1
other configuration	1	1
router	1	1
prefix-specific: *on-link*	0	0
prefix-specific: *autonomous address-configuration*	1	0

Tab. 13: Proposed values for flags of Router Advertisement messages.

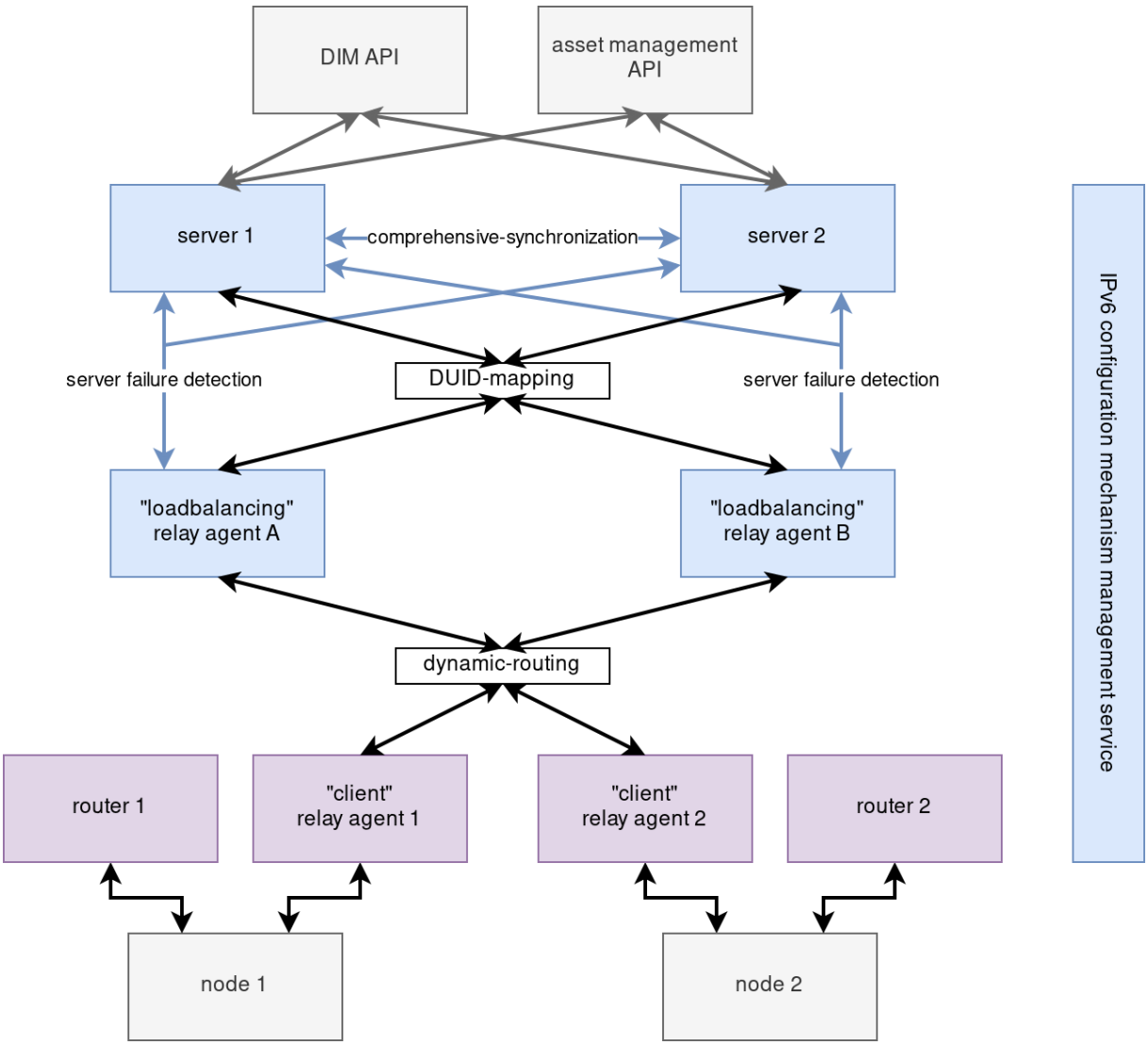


Fig. 16: This is the software architecture proposed for stateful DHCPv6, see section 6.3. The purple relay agent and router are deployed per link with clients. The blue services are deployed in other quantities. The gray services are out of scope. The "IPv6 configuration mechanism management service" configures and monitors the other colored components.

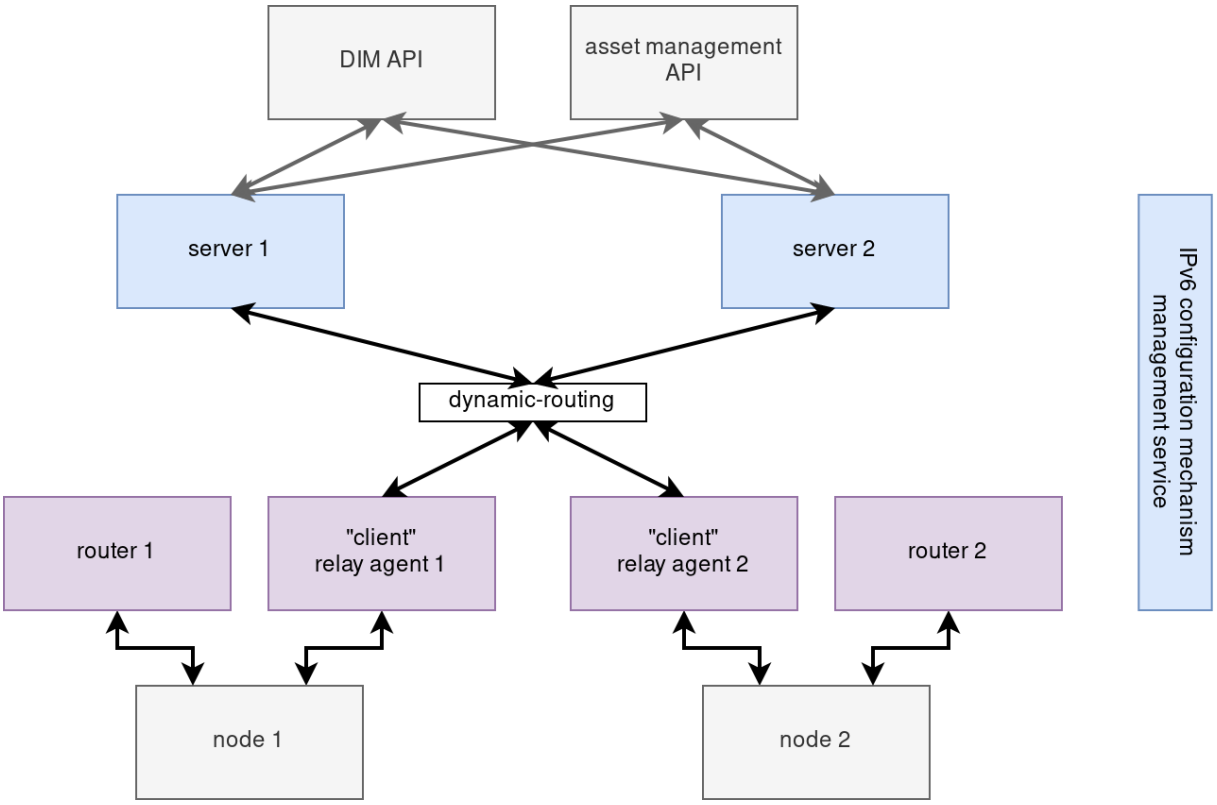


Fig. 17: This is the software architecture proposed for stateless DHCPv6, see section 6.3. For a description of this figure also see figure 16.

7 Results, Discussion and Future Work

The following sections summarize this work and propose future work.

7.1 Scope and Requirements

This work analyzes configuration mechanisms for IP addresses and other configuration information for two groups of nodes: PXE-DHCP and RMC-DHCP. The former describes nodes that need IP addresses and certain other configuration information to retrieve operating system images over the network and boot them (using the PXE standard). The latter group comprises remote management controllers of servers which need in addition to the requirements of PXE-DHCP nodes DNS records that point the IP addresses assigned to the nodes.

This work investigates what configuration mechanisms for IPv4 and IPv6 are available and how they work, namely DHCPv4, DHCPv6 and SLAAC (see section 2).[22]

7.2 Benefits of Migrating to IPv6

The necessity of a migration from IPv4 to IPv6 is evaluated in section 3. Solving the IPv4 address exhaustion, simplifying the network architecture by avoiding NAT and efficiency improvements are the main technical arguments in favor of IPv6. The increasing worldwide popularity of IPv6

[22] There are other configuration mechanisms which are left out here to focus on those relevant for the considered usecases.

and the issues with IPv4 in practice (caused by the limited address space) show that a migration of the considered node groups to IPv6-only is desirable.

Apart from four times lager addresses, IPv6 differs from IPv4 significantly. The protocol ICMPv6 is a mandatory part of IPv6 and defines much of its behavior. It not only implements functionality equivalent to ICMPv4, but also the mapping between link-layer addresses and IPv6 addresses, the assignment of IPv6 addresses to interfaces and much more.

7.3 Configuration Mechanisms for IPv6 Nodes

Multiple configuration mechanisms can be used simultaneously for assigning IPv6 addresses and other configuration information. Those mechanisms do not need to be aware of each other, because each node is required to assure the uniqueness of an address using ICMPv6 messages before assigning it to an interface. The mechanisms can be either *stateless* or *stateful*. A stateful mechanism requires in contrast to a stateless mechanism to synchronize node-specific configuration information to another node. For example, if a server maintains a database of IPv6 addresses for assigning them to nodes, it would be a *stateful* configuration mechanism.

The Stateless Address Auto Configuration (SLAAC) enables IPv6 nodes to assign themselves addresses without any dependency on other nodes.[23] Implementing SLAAC is compulsory for IPv6 nodes.

The DHCPv6 protocol is quite similar to DHCPv4 at first glance, but differs in various details, as described in section 2.8. It can be used without assigning addresses, which makes it a *stateless* configuration mechanism. It can be used to assign addresses and delegate entire address prefixes, in which case it becomes *stateful*. Clients are no longer identified using their link-layer address, but instead by a dedicated number generated per client. A message exchange can optionally be triggered by a server.

Information like router addresses and whether a specific prefix is *on-link* is configured using ICMPv6 messages sent by routers.

7.4 Resilient Stateless Configuration

IPv6 addresses and other configuration information can be configured in a stateless manner using Router Advertisement messages sent by routers (SLAAC) and stateless DHCPv6 servers. Because no client-specific information needs to be synchronized between them and they are queried using well-known multicast addresses, multiple instances can be run without further configuration.

7.5 Redundant Stateful DHCPv6 Servers

Maintaining consistency is a complex task when operating multiple redundant stateful DHCPv6 servers, especially when DNS records that point to leased addresses are wanted. Multiple scenarios that lead to inconsistency are analyzed in section 4. Issues arrive when a client changes its state (e.g. it gets a new address) but this does not get synchronized to all servers.

[23] To be able to assign addresses of a scope greater than link link-local, an IPv6 node needs to learn an appropriate address prefix from an ICMPv6 message sent by a router.

7.6 Results of the DHCPv6 Server Implementation Evaluation

Eleven open source implementations of DHCPv6 servers that run on Linux are found (see section 5). Some requirements for using those at the United Internet AG are specified, but not explored in detail. Future work could provide a more in-depth study of those.

7.7 The Proposed Service Architecture

For comparison, the current DHCPv4 architecture and related work concerning configuration of IPv6 nodes is presented (see section 6.1 and section 6.2).

Two architectures that provide IPv6 configuration for PXE-DHCP and RMC-DHCP nodes are proposed, one for each client group (see section 6.3). Both are designed to be entirely redundant. The architecture for PXE-DHCP nodes uses SLAAC, Router Advertisement messages and stateless DHCPv6 servers. The architecture for RMC-DHCP nodes uses Router Advertisement messages and stateful DHCPv6 servers. Both architectures use DHCPv6 relay agents, the latter also such that anycast messages from clients based on their DUID to different servers.

7.8 Discussion of Methodology And Future Work

7.8.1 Theory and Practice

The topic of this work is approached from a theoretical perspective. The protocols are studied and discussed in detail. Less work has been done on the evaluation of available implementations for deploying the proposed service architecture. Also the behavior of the existing nodes that need to be configured was not investigated. Those could deviate from the standards in form of missing features, not compliant implementations or additional implementations that offer other opportunities.

The proposed service architecture does neither specify what implementations to use, nor has it been tested.

Future work could test a range of values for the preferred lifetime and valid lifetime of addresses, T1 and T2 variables of Identity Associations and the TTL of DNS records. For those values should be tested how fast nodes receive their configuration, how fast the whole architecture propagates the state and how much network traffic and server resources that requires.

7.8.2 Performance

Due to the large number of nodes that need configuration, the traffic they might cause in a worst-case scenario could be overwhelming the infrastructure or at least require unnecessarily many resources. The many variables of the protocols (e.g. address lifetimes) could be tweaked and tested to find an optimum where the configuration information is refreshed often enough but also not congesting the infrastructure. If necessary, the performance of DHCPv6 server implementations (in terms of processed messages per time) could be benchmarked to evaluate implementations and find bottlenecks.

7.8.3 Security

The study of the protocols and the proposed service architecture do not contain any security considerations. Because the IPv6 addresses (and link-layer addresses) authenticate a node, special attention should be paid to ensure their correct assignment and no rogue assignments. Also, rogue DHCPv6 messages or Router Advertisement messages could spread malicious information. The *Secure Neighbor Discovery* protocol could be useful for this purpose (see [112], [113]).

7.8.4 More Literature

There is currently no standard that defines the synchronization of redundant DHCPv6 servers. However, RFC 6853 [114] discusses the deployment of redundant DHCPv6 servers and RFC 7031 [115] specifies requirements for designing a failover protocol between them. These documents are not considered in this work due to a lack of time.

Likewise, RFC 3449 [116] ("IPv6 Host Configuration of DNS Server Information Approaches") and RFC 4472 [117] ("Operational Considerations and Issues with IPv6 DNS") can be consulted for a discussion of configuration mechanisms for DNS configuration information for IPv6 nodes.

7.8.5 Economics

This work does not make an economic evaluation of the proposed configuration mechanism architecture.

7.8.6 Consistency

Methods to evaluate the consistency of given stateful configuration mechanism architectures could be addressed in future research.

References

[1] R. Callon, *The Twelve Networking Truths*, RFC 1925, Apr. 1996. DOI: 10.17487/RFC1925. [Online]. Available: https://rfc-editor.org/rfc/rfc1925.txt.

[2] International Telecommunication Union, *Measuring digital development Facts and figures 2019*. ITUPublications, 2019, ISBN: 978-92-61-29521-9. [Online]. Available: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/FactsFigures2019.pdf.

[3] M. Winther, *Tier 1 ISPs: What They Are and Why They Are Important*, External Publication of IDC Information and Data, white paper, May 2006. [Online]. Available: https://www.gin.ntt.net/wp-content/uploads/2020/01/IDC_Tier1_ISPs.pdf.

[4] *Internet Protocol*, RFC 791, Sep. 1981. DOI: 10.17487/RFC0791. [Online]. Available: https://rfc-editor.org/rfc/rfc791.txt.

[5] D. S. E. Deering and B. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 8200, Jul. 2017. DOI: 10.17487/RFC8200. [Online]. Available: https://rfc-editor.org/rfc/rfc8200.txt.

[6] S. Coty, *Where is IPv1, 2, 3,and 5?* Feb. 11, 2011. [Online]. Available: https://blog.alertlogic.com/blog/where-is-ipv1,-2,-3,and-5/ (visited on 08/27/2020).

[7] M. Cotton, L. Vegoda, R. Bonica, and B. Haberman, *Special-Purpose IP Address Registries*, RFC 6890, Apr. 2013. DOI: 10.17487/RFC6890. [Online]. Available: https://rfc-editor.org/rfc/rfc6890.txt.

[8] R. Droms, *Dynamic Host Configuration Protocol*, RFC 2131, Mar. 1997. DOI: 10.17487/RFC2131. [Online]. Available: https://rfc-editor.org/rfc/rfc2131.txt.

[9] T. Mrugalski, M. Siodelski, B. Volz, A. Yourtchenko, M. Richardson, S. Jiang, T. Lemon, and T. Winters, *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, RFC 8415, Nov. 2018. DOI: 10.17487/RFC8415. [Online]. Available: https://rfc-editor.org/rfc/rfc8415.txt.

[10] Intel Corporation, *Preboot Execution Environment (PXE) Specification*, Version 2.1, Sep. 1999. [Online]. Available: https://web.archive.org/web/20110524083740/http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf.

[11] *Domain names - concepts and facilities*, RFC 1034, Nov. 1987. DOI: 10.17487/RFC1034. [Online]. Available: https://rfc-editor.org/rfc/rfc1034.txt.

[12] D. T. Narten, T. Jinmei, and D. S. Thomson, *IPv6 Stateless Address Autoconfiguration*, RFC 4862, Sep. 2007. DOI: 10.17487/RFC4862. [Online]. Available: https://rfc-editor.org/rfc/rfc4862.txt.

[13] *Information technology - open systems interconnection - basic reference model: The basic model*, ITU-T Recommendation X.200, Telecommunication Standardization Sector of International Telecommunication Union, Jul. 1994. [Online]. Available: https://www.itu.int/rec/T-REC-X.200-199407-I.

[14] *User Datagram Protocol*, RFC 768, Aug. 1980. DOI: 10.17487/RFC0768. [Online]. Available: https://rfc-editor.org/rfc/rfc768.txt.

[15] *Transmission Control Protocol*, RFC 793, Sep. 1981. DOI: 10.17487/RFC0793. [Online]. Available: https://rfc-editor.org/rfc/rfc793.txt.

[16] LAN/MAN Standards Committee, *IEEE Standard for Ethernet*, IEEE-SA Standards Board, Ed., IEEE Std 802.3-2018. Institute of Electrical and Electronics Engineers, Inc, Jun. 2018, ISBN: 978-1-5044-5090-4. DOI: 10.1109/ieeestd.2018.8457469.

[17] D. M. Crawford, *Transmission of IPv6 Packets over Ethernet Networks*, RFC 2464, Dec. 1998. DOI: 10.17487/RFC2464. [Online]. Available: https://rfc-editor.org/rfc/rfc2464.txt.

[18] *DoD Internet host table specification*, RFC 952, Oct. 1985. DOI: 10.17487/RFC0952. [Online]. Available: https://rfc-editor.org/rfc/rfc952.txt.

[19] V. Fuller and T. Li, *Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*, RFC 4632, Aug. 2006. DOI: 10.17487/RFC4632. [Online]. Available: https://rfc-editor.org/rfc/rfc4632.txt.

[20] *An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*, RFC 826, Nov. 1982. DOI: 10.17487/RFC0826. [Online]. Available: https://rfc-editor.org/rfc/rfc826.txt.

[21] *A Reverse Address Resolution Protocol*, RFC 903, Jun. 1984. DOI: 10.17487/RFC0903. [Online]. Available: https://rfc-editor.org/rfc/rfc903.txt.

[22] *Bootstrap Protocol*, RFC 951, Sep. 1985. DOI: 10.17487/RFC0951. [Online]. Available: https://rfc-editor.org/rfc/rfc951.txt.

[23] W. Wimer, *Clarifications and Extensions for the Bootstrap Protocol*, RFC 1542, Oct. 1993. DOI: 10.17487/RFC1542. [Online]. Available: https://rfc-editor.org/rfc/rfc1542.txt.

[24] J. K. Reynolds, *BOOTP Vendor Information Extensions*, RFC 1497, Aug. 1993. DOI: 10.17487/RFC1497. [Online]. Available: https://rfc-editor.org/rfc/rfc1497.txt.

[25] B. Hinden and D. S. E. Deering, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 2460, Dec. 1998. DOI: 10.17487/RFC2460. [Online]. Available: https://rfc-editor.org/rfc/rfc2460.txt.

[26] D. S. E. Deering and B. Hinden, *IP Version 6 Addressing Architecture*, RFC 4291, Feb. 2006. DOI: 10.17487/RFC4291. [Online]. Available: https://rfc-editor.org/rfc/rfc4291.txt.

[27] B. Haberman, B. Zill, E. Nordmark, T. Jinmei, and D. S. E. Deering, *IPv6 Scoped Address Architecture*, RFC 4007, Mar. 2005. DOI: 10.17487/RFC4007. [Online]. Available: https://rfc-editor.org/rfc/rfc4007.txt.

[28] *Broadcasting Internet datagrams in the presence of subnets*, RFC 922, Oct. 1984. DOI: 10.17487/RFC0922. [Online]. Available: https://rfc-editor.org/rfc/rfc922.txt.

[29] M. Gupta and A. Conta, *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, RFC 4443, Mar. 2006. DOI: 10.17487/RFC4443. [Online]. Available: https://rfc-editor.org/rfc/rfc4443.txt.

[30] *Internet Control Message Protocol*, RFC 792, Sep. 1981. DOI: 10.17487/RFC0792. [Online]. Available: https://rfc-editor.org/rfc/rfc792.txt.

[31] B. Cain, D. S. E. Deering, B. Fenner, I. Kouvelas, and A. Thyagarajan, *Internet Group Management Protocol, Version 3*, RFC 3376, Oct. 2002. DOI: 10.17487/RFC3376. [Online]. Available: https://rfc-editor.org/rfc/rfc3376.txt.

[32] W. A. Simpson, D. T. Narten, E. Nordmark, and H. Soliman, *Neighbor Discovery for IP version 6 (IPv6)*, RFC 4861, Sep. 2007. DOI: 10.17487/RFC4861. [Online]. Available: https://rfc-editor.org/rfc/rfc4861.txt.

[33] G. Kuijpers, J. Wiljakka, J. Arkko, H. Soliman, and J. A. Loughney, *Internet Protocol Version 6 (IPv6) for Some Second and Third Generation Cellular Hosts*, RFC 3316, May 2003. DOI: 10.17487/RFC3316. [Online]. Available: https://rfc-editor.org/rfc/rfc3316.txt.

[34] D. S. E. Deering, B. Fenner, and B. Haberman, *Multicast Listener Discovery (MLD) for IPv6*, RFC 2710, Oct. 1999. DOI: 10.17487/RFC2710. [Online]. Available: https://rfc-editor.org/rfc/rfc2710.txt.

[35] L. Costa and R. Vida, *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*, RFC 3810, Jun. 2004. DOI: 10.17487/RFC3810. [Online]. Available: https://rfc-editor.org/rfc/rfc3810.txt.

[36] K. van den Hout, A. Koopal, and R. van Mook, *Management of IP numbers by peg-dhcp*, RFC 2322, Apr. 1998. DOI: 10.17487/RFC2322. [Online]. Available: https://rfc-editor.org/rfc/rfc2322.txt.

[37] R. Droms, *Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6*, RFC 3736, Apr. 2004. DOI: 10.17487/RFC3736. [Online]. Available: https://rfc-editor.org/rfc/rfc3736.txt.

[38] J. P. Jeong, S. D. Park, L. Beloeil, and S. Madanapalli, *IPv6 Router Advertisement Options for DNS Configuration*, RFC 8106, Mar. 2017. DOI: 10.17487/RFC8106. [Online]. Available: https://rfc-editor.org/rfc/rfc8106.txt.

[39] A. Atlasis and E. Rey, "IPv6 Router Advertisement Flags, RDNSS and DHCP Conflicting Configurations – Operational & Security Implications," ERNW Enno Rey Netzwerke GmbH, research rep. 1.0, Mar. 2015. [Online]. Available: https://www.ernw.de/download/ERNW_Whitepaper_IPv6_RAs_RDNSS_DHCPv6_Conflicting_Parameters.pdf.

[40] S. Venaas and T. Chown, *Rogue IPv6 Router Advertisement Problem Statement*, RFC 6104, Feb. 2011. DOI: 10.17487/RFC6104. [Online]. Available: https://rfc-editor.org/rfc/rfc6104.txt.

[41] C. E. Perkins, B. Volz, T. Lemon, M. Carney, and J. Bound, *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, RFC 3315, Jul. 2003. DOI: 10.17487/RFC3315. [Online]. Available: https://rfc-editor.org/rfc/rfc3315.txt.

[42] G. Halwasia, S. Bhandari, and W. Dec, *Client Link-Layer Address Option in DHCPv6*, RFC 6939, May 2013. DOI: 10.17487/RFC6939. [Online]. Available: https://rfc-editor.org/rfc/rfc6939.txt.

[43] T. Lemon and B. E. Sommerfeld, *Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)*, RFC 4361, Feb. 2006. DOI: 10.17487/RFC4361. [Online]. Available: https://rfc-editor.org/rfc/rfc4361.txt.

[44] J. Neville. (Nov. 13, 2018). "Android does not support dhcpv6 and google 'won't fix' that," [Online]. Available: https://www.nullzero.co.uk/android-does-not-support-dhcpv6-and-google-wont-fix-that/ (visited on 07/31/2020).

[45] (Dec. 9, 2014). "Issuetracker: Support for dhcpv6 (rfc 3315)," [Online]. Available: https://issuetracker.google.com/issues/36949085#comment66 (visited on 08/03/2020).

[46] J. Sanders, *Android's lack of DHCPv6 support poses security and IPv6 deployment issues, Android doesn't support dhcpv6, the most common management method for enterprise wi-fi deployments. find out why, and how to work around this problem.* CBS Interactive Inc., Sep. 1, 2015. [Online]. Available: https://www.techrepublic.com/article/androids-lack-of-dhcpv6-support-poses-security-and-ipv6-deployment-issues/ (visited on 08/27/2020).

[47] B. Volz, *Re: [dhcwg] [Technical Errata Reported] RFC8415 (6269)*, IETF mail archive, Aug. 31, 2020. [Online]. Available: https://mailarchive.ietf.org/arch/msg/dhcwg/IS955z1mhCjGuuWC4HCMkS75N2k/ (visited on 08/31/2020).

[48] Fujitsu Technology Solutions GmbH, *Frequently asked questions on high availability*, Jul. 2012. [Online]. Available: https://sp.ts.fujitsu.com/dmsp/Publications/public/wp_faq-h-availability.pdf.

[49] G. P. Sarmila, N. Gnanambigai, and P. Dinadayalan, "Survey on fault tolerant — load balancing algorithmsin cloud computing," in *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, 2015, pp. 1715–1720.

[50] Bundesamt für Sicherheit in der Informationstechnik, *Hochverfügbarkeitskompendium*, Version 1.6, Kapitel 5: Server. 2013, vol. Band B. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Hochverfuegbarkeit/BandB/B5_Server.pdf?__blob=publicationFile&v=1.

[51] A. M. Alakeel, "A guide to dynamic load balancing in distributed computer systems," *International Journal of Computer Science and Information Security*, vol. 10, no. 6, pp. 153–160, 2010.

[52] N. J. Kansal and I. Chana, "Cloud load balancing techniques: A step towards green computing," *IJCSI International Journal of Computer Science Issues*, vol. 9, no. 1, pp. 238–246, 2012.

[53] L. Duboc, D. Rosenblum, and T. Wicks, "A framework for characterization and analysis of software system scalability," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2007, pp. 375–384.

[54] Y. Rekhter, S. Hares, and T. Li, *A Border Gateway Protocol 4 (BGP-4)*, RFC 4271, Jan. 2006. DOI: 10.17487/RFC4271. [Online]. Available: https://rfc-editor.org/rfc/rfc4271.txt.

[55] Internet Assigned Numbers Authority. (). "Number resources," [Online]. Available: https://www.iana.org/numbers (visited on 06/09/2020).

[56] User:Sémhur, *Regional internet registries world map*, Jan. 2009. [Online]. Available: https://commons.wikimedia.org/wiki/File:Regional_Internet_Registries_world_map.svg.

[57] Internet Assigned Numbers Authority, *Iana ipv4 address space registry*, The allocation of Internet Protocol version 4 (IPv4) address space to various registries is listed here. Originally, all the IPv4 address spaces was managed directly by the IANA. Later parts of the address space were allocated to various other registries to manage for particular purposes or regional areas of the world. RFC 1466 documents most of these allocations., Dec. 2019. [Online]. Available: https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml.

[58] Réseaux IP Européens Network Coordination Centre. (Nov. 25, 2019). "The ripe ncc has run out of ipv4 addresses," [Online]. Available: https://www.ripe.net/publications/news/about-ripe-ncc-and-ripe/the-ripe-ncc-has-run-out-of-ipv4-addresses (visited on 06/09/2020).

[59] J. Curran. (Sep. 24, 2015). "Arin ipv4 free pool reaches zero," American Registry for Internet Numbers (ARIN), [Online]. Available: https://www.arin.net/vault/announcements/2015/20150924.html (visited on 07/08/2020).

[60] African Network Information Centre (AFRINIC). (Apr. 14, 2020). "Afrinic enters ipv4 exhaustion phase 2," [Online]. Available: https://afrinic.net/20200113-afrinic-enters-ipv4-exhaustion-phase-2 (visited on 06/09/2020).

[61] Latin America and Caribbean Network Information Centre. (Jun. 10, 2020). "Phases of ipv4 exhaustion," [Online]. Available: https://www.lacnic.net/1039/2/lacnic/phases-of-ipv4-exhaustion.

[62] Asia-Pacific Network Information Centre. (2020). "Ipv4 exhaustion," [Online]. Available: https://www.apnic.net/manage-ip/ipv4-exhaustion/ (visited on 07/08/2020).

[63] M. Holdrege and P. Srisuresh, *IP Network Address Translator (NAT) Terminology and Considerations*, RFC 2663, Aug. 1999. DOI: 10.17487/RFC2663. [Online]. Available: https://rfc-editor.org/rfc/rfc2663.txt.

[64] R. E. Gilligan and E. Nordmark, *Basic Transition Mechanisms for IPv6 Hosts and Routers*, RFC 4213, Oct. 2005. DOI: 10.17487/RFC4213. [Online]. Available: https://rfc-editor.org/rfc/rfc4213.txt.

[65] D. Schinazi and T. Pauly, *Happy Eyeballs Version 2: Better Connectivity Using Concurrency*, RFC 8305, Dec. 2017. DOI: 10.17487/RFC8305. [Online]. Available: https://rfc-editor.org/rfc/rfc8305.txt.

[66] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, Jun. 1999. DOI: 10.17487/RFC2616. [Online]. Available: https://rfc-editor.org/rfc/rfc2616.txt.

[67] P. Matthews, I. van Beijnum, and M. Bagnulo, *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers*, RFC 6146, Apr. 2011. DOI: 10.17487/RFC6146. [Online]. Available: https://rfc-editor.org/rfc/rfc6146.txt.

[68] P. Matthews, A. Sullivan, I. van Beijnum, and M. Bagnulo, *DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers*, RFC 6147, Apr. 2011. DOI: 10.17487/RFC6147. [Online]. Available: https://rfc-editor.org/rfc/rfc6147.txt.

[69] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, *DNS Security Introduction and Requirements*, RFC 4033, Mar. 2005. DOI: 10.17487/RFC4033. [Online]. Available: https://rfc-editor.org/rfc/rfc4033.txt.

[70] J. Linkova, *Let's talk about IPv6 DNS64 & DNSSEC*, Jun. 9, 2016. [Online]. Available: https://blog.apnic.net/2016/06/09/lets-talk-ipv6-dns64-dnssec/ (visited on 08/27/2020).

[71] M. Mawatari, M. Kawashima, and C. Byrne, *464XLAT: Combination of Stateful and Stateless Translation*, RFC 6877, Apr. 2013. DOI: 10.17487/RFC6877. [Online]. Available: https://rfc-editor.org/rfc/rfc6877.txt.

[72] P. Stevens, *Ipv6-only*, RIPE NCC::Educa, Jun. 2020. [Online]. Available: https://www.ripe.net/support/training/ripe-ncc-educa/presentations/pete-stevens-ipv6-only.pdf.

[73] T. Coffeen, *IPv6 Address Planning: A Case-Study*, 2017 North American IPv6 Summit, Apr. 1, 2017. [Online]. Available: https://www.rmv6tf.org/wp-content/uploads/2017/04/03-2017-NAv6TF-Summit_4-21-2017-min.pdf (visited on 08/27/2020).

[74] P. A. R. Ade et al., "Planck 2015 results - xiii. cosmological parameters," *Astronomy and Astrophysics*, vol. 594, pp. 1–63, Sep. 2016. DOI: 10.1051/0004-6361/201525830.

[75] Asia Pacific Network Information Centre, *Network address translation*, 2020. [Online]. Available: https://www.apnic.net/community/ipv6/network-address-translation/ (visited on 08/27/2020).

[76] V. McKillop, *Microsoft IT: Journey to IPv6*, 2017 North American IPv6 Summit, Microsoft IT, Apr. 1, 2017. [Online]. Available: `https://www.rmv6tf.org/wp-content/uploads/2017/04/02-MicrosoftIT_IPv6-NA-IPv6-Summit-2017_VMcKillop-min.pdf` (visited on 08/27/2020).

[77] Der Beauftragte der Bundesregierung für Informationstechnik, *Architekturrichtlinie für die IT des Bundes*, Version 1.0 vom 31.07.2020, Jul. 2020. [Online]. Available: `https://www.cio.bund.de/SharedDocs/Publikationen/DE/Architekturen-und-Standards/architekturrichtlinie_it_bund_2020.pdf?__blob=publicationFile`.

[78] A. Gustafsson, T. Lemon, and M. Stapp, *A DNS Resource Record (RR) for Encoding Dynamic Host Configuration Protocol (DHCP) Information (DHCID RR)*, RFC 4701, Oct. 2006. DOI: `10.17487/RFC4701`. [Online]. Available: `https://rfc-editor.org/rfc/rfc4701.txt`.

[79] Y. Rekhter, B. Volz, and M. Stapp, *The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option*, RFC 4702, Oct. 2006. DOI: `10.17487/RFC4702`. [Online]. Available: `https://rfc-editor.org/rfc/rfc4702.txt`.

[80] B. Volz and M. Stapp, *Resolution of Fully Qualified Domain Name (FQDN) Conflicts among Dynamic Host Configuration Protocol (DHCP) Clients*, RFC 4703, Oct. 2006. DOI: `10.17487/RFC4703`. [Online]. Available: `https://rfc-editor.org/rfc/rfc4703.txt`.

[81] B. Volz, *The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option*, RFC 4704, Oct. 2006. DOI: `10.17487/RFC4704`. [Online]. Available: `https://rfc-editor.org/rfc/rfc4704.txt`.

[82] R. Droms and S. Alexander, *DHCP Options and BOOTP Vendor Extensions*, RFC 2132, Mar. 1997. DOI: `10.17487/RFC2132`. [Online]. Available: `https://rfc-editor.org/rfc/rfc2132.txt`.

[83] Opensource.org, *The open source definition*, Mar. 22, 2007. [Online]. Available: `https://opensource.org/docs/osd` (visited on 08/30/2020).

[84] A. Barberio, A. Denis, and P. Mazzini, *Coredhcp, Fast, multithreaded, modular and extensible dhcp server written in go*, commit a9aa31766d13f1169b98b8ce23ed632f344570c8, Aug. 9, 2020. [Online]. Available: `https://github.com/coredhcp/coredhcp` (visited on 08/22/2020).

[85] A. Barberio, *Dhcp, Dhcpv6 and dhcpv4 packet library, client and server written in go*, commit 2e1bf785d039de3fd451b63cfe937456b32e739c, Aug. 14, 2020. [Online]. Available: `https://github.com/insomniacslk/dhcp` (visited on 08/22/2020).

[86] M. Layher, *Dhcp6, Package dhcp6 implements a dhcpv6 server, as described in rfc 3315. mit licensed.* commit 2a67805d7d0b0bad6c1103058981afdea583b459, Mar. 11, 2019. [Online]. Available: `https://github.com/mdlayher/dhcp6` (visited on 08/23/2020).

[87] Facebook, Inc., *Dhcplb, Dhcplb is facebook's implementation of a load balancer for dhcp*, dhcplb is Facebook's implementation of a load balancer for DHCP. commit 3b70e1c, Jul. 18, 2020. [Online]. Available: https://github.com/facebookincubator/dhcplb (visited on 08/23/2020).

[88] H. Wahl, *Dhcpy6d, dhcpv6 made simple*, 2020. [Online]. Available: https://dhcpy6d.ifw-dresden.de/ (visited on 08/22/2020).

[89] T. Mrugalski, *Dibbler – a portable dhcpv6 user's guide*, 1.0.2RC1, Jul. 3, 2017. [Online]. Available: https://klub.com.pl/dhcpv6/doc/dibbler-user.pdf (visited on 08/23/2020).

[90] S. Kelley, *Dnsmasq - network services for small networks*. [Online]. Available: http://www.thekelleys.org.uk/dnsmasq/doc.html (visited on 08/22/2020).

[91] Internet Systems Consortium, Inc., *ISC DHCP*, ISC DHCP is enterprise grade, open source solution for DHCP servers, relay agents, and clients, supports both IPv4 and IPv6, and is suitable for use in high-volume and high-reliability applications. commit 26243975f0c0df85391d05a0c6b81cba56a656cd, Aug. 7, 2020. [Online]. Available: https://gitlab.isc.org/isc-projects/dhcp (visited on 08/22/2020).

[92] Jagornet Technologies LLC, *Jagornet dhcp server, Open source dynamic host configuration protocol server for ipv4 (dhcpv4) and ipv6 (dhcpv6)*. 2017. [Online]. Available: http://www.jagornet.com/products/dhcp-server (visited on 08/23/2020).

[93] Internet Systems Consortium, Inc., *Kea, Modern, open source DHCPv4 & DHCPv6 serve*, 2020. [Online]. Available: https://www.isc.org/kea/ (visited on 08/22/2020).

[94] ——, *Kea, Modern DHCP*, Kea is an open source software system including DHCPv4, DHCPv6 servers, Dynamic DNS daemon, REST API interface, MySQL, PostgreSQL and Cassandra databases, RADIUS and NETCONF interfaces and related utilities., Nov. 26, 2018. [Online]. Available: https://kea.isc.org/ (visited on 08/13/2020).

[95] T. Mrugalski, *Kea – modern dhcp server*, May 11, 2017. [Online]. Available: https://ripe74.ripe.net/wp-content/uploads/presentations/140-kea-ripe74-final.pdf (visited on 08/22/2020).

[96] N. J. Kain, *Ndhs, Dhcp4, dhcp6, and ipv6 router advertisement server*. commit 0119bd4, Feb. 9, 2018. [Online]. Available: https://github.com/niklata/ndhs (visited on 08/23/2020).

[97] Slashdot Media, *Wide-dhcpv6*, Jul. 23, 2015. [Online]. Available: https://sourceforge.net/projects/wide-dhcpv6/ (visited on 08/23/2020).

[98] Internet Systems Consortium, Inc., *Kea dhcp technical support, Key features of a kea support subscription*, version rev0720r2. [Online]. Available: https://www.isc.org/docs/Kea-Support-Subscription.pdf (visited on 08/31/2020).

[99] Internet Systems Consortium, *perfdhcp - DHCP benchmarking tool*, revision fbcbc870, 2020. [Online]. Available: https://kea.readthedocs.io/en/latest/man/perfdhcp.8.html (visited on 08/28/2020).

[100] 1&1 Group, *DIM (DNS and IP Management)*, commit 2599a1f, Apr. 20, 2020. [Online]. Available: `https://github.com/1and1/dim` (visited on 08/10/2020).

[101] The PostgreSQL Global Development Group, *PostgreSQL: The World's Most Advanced Open Source Relational Database*, 2020. [Online]. Available: `https://www.postgresql.org/` (visited on 08/11/2020).

[102] P. Mazzini, *Extending DHCPLB: The path from load balancer to server*, May 28, 2019. [Online]. Available: `https://engineering.fb.com/data-infrastructure/dhcplb-server/` (visited on 08/12/2020).

[103] A. Failla, *DHCPLB: An open source load balancer*, Sep. 13, 2016. [Online]. Available: `https://engineering.fb.com/data-infrastructure/dhcplb-an-open-source-load-balancer/` (visited on 08/12/2020).

[104] E. Green, *Using isc kea dhcp in our data centers*, Jul. 21, 2015. [Online]. Available: `https://engineering.fb.com/core-data/using-isc-kea-dhcp-in-our-data-centers/` (visited on 08/12/2020).

[105] T. Mrugalski, W. Wencel, and V. Risk, *Kea 1.8.0, aug 26th 2020, release notes*, Aug. 26, 2020. [Online]. Available: `https://gitlab.isc.org/isc-projects/kea/-/wikis/release%20notes/release-notes-1.8.0` (visited on 08/28/2020).

[106] R. Droms and K. E. K. Jr., "DHCP Failover Protocol," Internet Engineering Task Force, Internet-Draft draft-ietf-dhc-failover-12, Mar. 2003, Work in Progress, 133 pp. [Online]. Available: `https://datatracker.ietf.org/doc/html/draft-ietf-dhc-failover-12`.

[107] S. Goldlust and M. Siodelski, *Kea High Availability vs ISC DHCP Failover*, Aug. 6, 2019. [Online]. Available: `https://kb.isc.org/docs/aa-01617` (visited on 08/13/2020).

[108] T. Mrugalski, *High availability in kea 1.4.0 - design*, Nov. 11, 2019. [Online]. Available: `https://gitlab.isc.org/isc-projects/kea/-/wikis/designs/High-Availability-Design` (visited on 08/13/2020).

[109] T. Mrugalski and K. Kinnear, *DHCPv6 Failover Protocol*, RFC 8156, Jun. 2017. DOI: `10.17487/RFC8156`. [Online]. Available: `https://rfc-editor.org/rfc/rfc8156.txt`.

[110] Internet Systems Consortium, Inc., *Kea 1.8.0-git documentation, 16.15. ha: High availability*, version Revision c8699245, 2020. [Online]. Available: `https://kea.readthedocs.io/en/kea-1.8.0/arm/hooks.html#ha-high-availability` (visited on 08/30/2020).

[111] B. Volz, S. Gonczi, T. Lemon, and R. L. Stevens, *DHC Load Balancing Algorithm*, RFC 3074, Feb. 2001. DOI: `10.17487/RFC3074`. [Online]. Available: `https://rfc-editor.org/rfc/rfc3074.txt`.

[112] J. Kempf, J. Arkko, B. Zill, and P. Nikander, *SEcure Neighbor Discovery (SEND)*, RFC 3971, Mar. 2005. DOI: `10.17487/RFC3971`. [Online]. Available: `https://rfc-editor.org/rfc/rfc3971.txt`.

[113] A. AlSa'deh and C. Meinel, "Secure neighbor discovery: Review, challenges, perspectives, and recommendations," *ieee symposium on security and privacy*, vol. 10, no. 4, pp. 26–34, 2012. DOI: `10.1109/MSP.2012.27`.

[114] J. J. Brzozowski, J.-F. Tremblay, J. Chen, and T. Mrugalski, *DHCPv6 Redundancy Deployment Considerations*, RFC 6853, Feb. 2013. DOI: 10.17487/RFC6853. [Online]. Available: https://rfc-editor.org/rfc/rfc6853.txt.

[115] T. Mrugalski and K. Kinnear, *DHCPv6 Failover Requirements*, RFC 7031, Sep. 2013. DOI: 10.17487/RFC7031. [Online]. Available: https://rfc-editor.org/rfc/rfc7031.txt.

[116] J. Jeong, *IPv6 Host Configuration of DNS Server Information Approaches*, RFC 4339, Feb. 2006. DOI: 10.17487/RFC4339. [Online]. Available: https://rfc-editor.org/rfc/rfc4339.txt.

[117] A. Durand, J. Ihren, and P. Savola, *Operational Considerations and Issues with IPv6 DNS*, RFC 4472, Apr. 2006. DOI: 10.17487/RFC4472. [Online]. Available: https://rfc-editor.org/rfc/rfc4472.txt.