IP45: Architecture and Design

http://www.ip45.org/

Tomáš Podermański
Brno University of Technology
Center of Computer and Information Services
Antonínská 1, 601 90 Brno, Czech republic
Email: tpoder@cis.vutbr.cz

Matěj Grégr, Miroslav Švéda
Brno University of Technology
Faculty of Information Technology
Božetěchova 2, 612 66 Brno, Czech republic
Email: igregr,sveda@fit.vutbr.cz

***Abstract*—The IP45 architecture and protocol design presented in this paper is an extension of IPv4 protocol with ability to regain the end-to-end connectivity in the networks placed behind Network Address Translators (NATs). The proposed solution can be implemented within existing networks with minimal effort, at very low cost. The paper discusses the benefits and limitations of the proposed solution and compares it with others. The implementation available for all important platforms is also briefly presented.**

***Keywords*—Internet Protocol; End-to-End Communication**

I. Introduction

Soon it will be more than twenty years since the Internet community started to realize that the address space provided by the IP protocol was not sufficient enough for future deployment. Being aware of the problem, in 1993, IETF made a request for a proper solution and in a few years, the first set of standards was released and the protocol got the name under which it is known today - *IPv6 Protocol*. The initial idea of IPv6 deployment was quite simple. The new protocol would have been adopted by vendors gradually and implemented into existing networks along with existing IP protocol [1]. The key motivation to deploy IPv6 might have been new features such as security, mobility, autoconfiguration, etc., which were added as the integral part of the design. The whole process of the transition to IPv6 should have been finished before the depletion of the IPv4 address pool. To support that, many projects were funded and political forces in almost every country, including great powers like U.S., EU and China, created their own strategies and deployment plans to support IPv6 adoption.

The statistics from the beginning of 2014 provided by Geoff Huston [2], Google [3] or 6lab.cz [4], show that more than 16% of *Autonomous Systems* (AS) announced IPv6 prefix in the global BGP table. More than 3.5% of clients are able to use native IPv6 connectivity and more than 6% of the web content is available over IPv6. Some statistics such as 6lab.cisco.com [5], which take into consideration only the most visited websites, indicate that in some countries this number can reach up to 30%; however, if we look at the numbers from a different perspective, the situation might not look so optimistic. We could claim that 94% of web content is available only over IPv4, 96% of users are not able to have IPv6 connectivity and 74% of AS have never delivered a single IPv6 packet.

It is quite difficult to underline the true reason why IPv6 deployment is not going well. The most operating systems support IPv6 for more than a decade and major applications like web browsers, mail, torrent, ssh, rdp clients do not fall behind as well. It seems that the ultimate problem of IPv6 deployment is the lacking support within the network infrastructure. Some organizations and ISPs prefer playing the waiting game with IPv6 deployment to avoid early adoption disadvantages and to postpone the cost related to running dualstack environment. In any case, IPv6 transition is a very complex process involving technical, social and economical factors. Today, even the most optimistic predictions expect that the transition will not take less than decade.

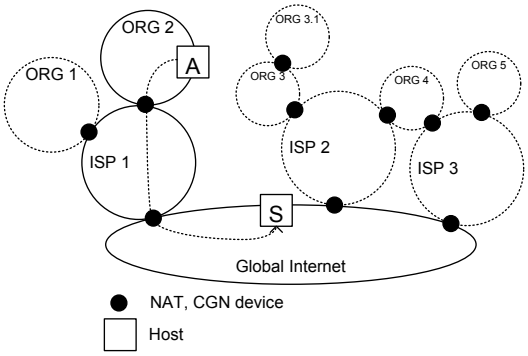

Fig. 1: The Network Base on NAT Architecture

However, the IANA IPv4 address pool is already depleted and ISPs, who want to start a new business or extend the existing one, have a problem. Deployment of IPv6 only services do not help much as most services are available only via IPv4.

One of the options is to obtain an IPv4 address block on the market. Although, the rules of Regional Internet Registries (RIRs) tend to forbid trading with addresses, some of them have already issued rules to legalize IP address transfers between organizations. According to the information released by one of the official address brokers [6], a single IP address has a price range from $9.85 to $35. In the remaining regions, address trading has become a part of gray market which obviously brings other problems as pointed out in Eric Osterweil's work [7].

Another solution to mitigate the problem with the depleted address space is to deploy NAT or CGN into the network. Big providers like Verizon [8] or British Telecom [9] have started testing or deploying CGN in their network for some customers, even if they have already started deploying IPv6.

However, insufficient address space is not the only reason why NATs are used. One of the key advantages provided by NAT is the *address independency* and *reusing single IP address* by many devices. In practice, it does not matter how many NATs are connected in the chain and the IP network topology can be arranged into hierarchical and recursive structure as shown in figure 1. The remaining benefits provided by NAT are, generally, the result of the two main attributes mentioned before. In a very easy and cheap way, the *address independency* allows us to implement features like small site multihoming without running a BGP router [10], or to change the upstream ISP without the readdressing of all devices. Another frequent reason for using NAT is the consequence of *reusing a single IP address*. Firstly, it hides the internal structure of the network and secondly, it forbids a connection with a device in the NATed network from the outside. Both of them are considered as security features.

Nevertheless, the use of NATs brings some disadvantages [11]. The common (and in many cases the only) reason to condemn NAT is breaking the end-to-end connectivity. The devices placed behind NAT, or several NATs, are not globally addressable. Another disadvantage is the limited number of sessions passing through NAT at the same time. This number cannot cross the combination of possible ports in TCP or UDP sessions with a single destination IP address. NAT also needs to keep a state for every running session. Finally, it works on the transport layer, thus it has to know every transport protocol that can pass through it.

Despite many issues, NAT is a massively used and popular technology which brings some interesting and often overlooked benefits. According to Paul Francis [12], NAT has become the integral part of current Internet architecture and we agree with that since it can be found in almost every home or enterprise network.

This work discusses and proposes an experimental network architecture that tries to eliminate the key disadvantages of existing network architectures and, at the same time, keeps its benefits. The following section provides a short overview of identified challenges, the addressing approaches and networking architectures. Section IV discusses the model of IP45 architecture which is extended with design specification in section V. Section VI discusses benefits and limitations of the proposed architecture and compares it with others. Finally, section VII deals with implementation matters which are concluded in section VIII where the future work on IP45 architecture is also discussed.

II. Identified Challenges

There are some criteria that can be considered while discussing network architectures. In our work we focused on the following main first-principles and features:

Sattzer's End-to-End argument represents one of the basic principles when a new network architecture is designed [**?**]. According to our observation, following the principle determines whether a network architecture will be succesfull or not.

Thanks to **End-to-End addressing** every device in the network is able to send data to any other node connected to the network directly. In current networks this principle is broken by NAT and the transition to IPv6 should have solved the problem.

Address space hierarchisation allows to keep a number of routing entries, especially at the top in the hierarchy of the global network, at the necessary minimum. The good example of such architecture is IPv6 in which every subnetwork (in IP world known as Autonomous System) can be determined by a single routing entry. In practice, the benefit of hierarchised address space in IPv6 is impaired by *Provider Independent* addresses which are used for multihoming or to avoid demanding readdressing within the site.

Address space independency allows the network to reconnect a part of subtree to another node without any need for readdressing within a child subtree. Address space independency can solve some crucial problems of hierarchical address space mentioned in the previous paragraph. Protocol IPv6 is a nice example which shows that the absence of *address space independency* implies some solutions where one problem can lead to another. In IPv6 every host (or interface) needs to know a *network prefix* to which is connected. It can be set either manually or advertised by a router. For the distribution of prefixes across routers, IPv6 must have an additional mechanism called *Prefix Delegation* [13]. If network prefix is changed, new prefix information has to be distributed to every IPv6 host. As the readdressing of hosts is in many cases very complicated, IPv6 offers some solutions to mitigate the problem. Firstly, every IPv6 interface can configure an additional, IPv6 address[1]. But the existence of multiple addresses brings another complication. To choose a proper IPv6 address as the source address in the outgoing packets, every IPv6 host must keep the *Priority Table*. Another, relatively controversial, solution is known as *IPv6-to-IPv6 Network Prefix Translation* [14]. It allows to map the global IPv6 prefix assigned from the ISP to the *Unique Local IPv6 Unicast Address* used inside of the site. But it brings many disadvantages and limitations to IPv6 architecture caused by NAT. The third approach to deal with address space independency in IPv6 is to obtain *Provider Independent (PI)* address space and to announce it within its own *Autonomous System (AS)* to the global BGP. However, if every organization takes its own PI prefix, the number of routing entries and updates in BGP will exceed the tolerable limit. In IPv4 the absence of address space independency is not as noticeable as in IPv6. The problem is mitigated by NATs which add address space independency into IPv4 address schema. On the contrary, architectures such as RINA [15] or IPNL [12] use address space independency as a key feature of protocol design.

Session independency together with address space independency is an important prerequisite to multihoming and mobility. Thanks to session independency the address of end nodes can be changed during running session. Today, neither IPv4 nor IPv6 support session independency and the feature must be solved on top of those protocols.

Scalability is a feature which allows to expand the nework without creating some kind of bottleneck. In current network architectures there are many potential bottlonecks which could restrict future network growth. For example, current global

[1]It could be Link-Local, ULA or another global address.

routing table grows every day and all routers must keep all routes connected to the global routing system. Another example with poor scalability is NAT, where a NAT device has to keep the state for every running session in memory.

Implementation and operational complexity are not directly related to the architectural principles, however, they represent a very important factor considering implementation and operational cost. It is obvious that a more complex design makes implementation, testing debugging more difficult and more expensive.

Adoption difficulty is a factor similar to the previous one. The protocol design must take into account existing protocols, technologies and application interfaces. The transition process of IPv6 points out how important it is.

It may seem that every principle or feature previously mentioned has been already solved in some way, but adding them together into one architecture is a real challenge. For example, an architecture which supports address space hierarchy (e.g. LNAT [16] or IPNL [12]), usually contradicts the design requirements of multihoming and mobility which again disagree with scalability and Saltzers's end-to-end principle. Similar situation is with NATs. NAT perfectly implements address space hierarchisation and independency and can be adopted very well, but it does not support End-to-End addressing and has poor scalability. Concerning mobility, both IPv4 and IPv6 designs do not support session independency and the mobility solution is built on top of protocols such as MIPv6, MPTCP, PMIPv6 or SHIM6 [17]). As a result, none of those solutions is practically used and mobility in IPv4 and IPv6 is therefore rather a theoretical concept than a feature used on daily basis.

III. Related Work

One of the first works related to the topic of address space exhaustion can be found in Van Jacobsons's draft of the LNAT [16]. A couple of years before IETF began its search for the future Internet solution, Jacobson had pointed out two basic concepts of address space expansion. Firstly, the conversion to a new protocol with larger address space, secondly, reusing the existing protocol by dividing it into more domains. Being aware of the difficulties with the transition, Jacobson proposed a solution in which the networks are organized in *Addressing Domains*. The communication between the domains is based on a *mapping table* which is built according to modified DNS queries and responses. After IETF announced the call for a new Internet architecture, most of the proposals like SIPP (PIP) [18], CATNIP [19] and TUBA [20], preferred to use an incompatible protocol with an incompatible addressing scheme. The key focus was put on features such as autoconfiguration, security and mobility and the process of transition was not expected to be so demanding. Some early attempts to bring IPv4 compatible architecture can be found in EIP [21] where the packets use an additional IPv4 option header. The extra option header extends addressing schemes with an extra 32 bits. There were doubts about the performance of the packet processing because IPv4 options are, typically, not processed by specialized hardware. Today, EIP is marked as deprecated [22].

A similar approach was taken by IPv4+4 [23]. The compatibility with IPv4 protocol is secured by adding a shim header that carries extended address information. For devices which do not support IPv4+4, the traffic acts as regular traffic that bears IPv4+4 protocol on the transport layer. IPv4+4 proposes an addressing scheme in which the Internet is separated into address areas called *realms*. Every *realm* has its own addressing scheme independent of the others. When a packet is sent to a different realm, it is passed through *realm gateways* which are responsible for *swap*, an operation that allows IPv4+4 packet delivery within a targeted *realm*. Unfortunately, even if IPv4+4 architecture was implemented [24], and later the architecture was modified to use LSSR route option [25], the work on the original architecture would not have continued, as it was confirmed by the author.

Mike O'Dell proposes different address semantics and separation. He divides the address into two parts; *Locator* and *Identifier* of the communicating device. His scheme [26] was not implemented, but a very similar approach was used in many proposals [17], [27] as summarized in [28]. Some of that ideas were standardized later in LISP architecture [29].

Some authors tried to take a completely different approach. In works mentioned above, the address was usually taken as a number with a fixed length which could be, in some cases, multiplied. In the architecture proposed by Nimrod [30] and later by IPNL [12] and TRIAD [31], the numerical address is not a valid identifier anymore. The *named address* is represented by a character string which is, usually, identical to a fully qualified domain name (FQDN). This way the architecture can, theoretically, operate with almost infinite address space and create a hierarchical network with unlimited levels. Especially IPNL divides the network into different *realms* where each *realm* is of the same manner as it is used in DNS. These ideas are later extended in NUTSS [32] and STUNT architecture. One of the biggest advantages is that all proposals expect to build a topology on the top of the existing IPv4 infrastructure which accepts the existence of NATs.

With regards to networking architectures and addressing, the substantial RINA project [33] cannot be omitted. RINA sees the networking communication as a common interprocess communication (IPC) where the communication passes through repeating the same IPC layers which are named DIFs. In this way RINA allows the building of a recursive structure of the network.

The remaining group of solutions is not directly designed to extend the address space. There are two protocols, UpNP/IGD [34] and NAT-PMP [35], which use almost the same principle. A client placed behind NAT can send a packet with a map request to the NAT device. After that, the NAT device creates a binding between the port on the public address and the private address of the device that requested it. Those protocols are supported by many vendors producing CPEs and network devices. Nevertheless, there are several critical disadvantages regarding the use of such techniques. Firstly, the protocols expect only a simple L2 topology. Secondly, the protocols work only on one level of NATs. If the device is placed behind two levels of NATs (for example CGN and home NAT), the protocols do not work. Thirdly, NAT devices have to keep the information about mapping state. Another NAT traversal protocol, STUN [36], faces similar problems. It works only for UDP, requires to cooperate with STUN servers on public address and might not work properly with all forms of NAT.

IV. The Model of IP45 Networks

The IP45 network model is represented by the collection of *Administrative Domains (AD)* which are organised into a tree structure. Every AD represents a node of the tree and the depth is defined as a *Level*. At the top of the hierarchy there is the *root AD* called *Administrative Domain Level 0 (AD 0)*. An *AD* can be connected to the parent or child *AD* through *Border Gateway (BGW)*. *BGW* configures an *Upstream Address (UA)* and a *Downstream Prefix (DP)*. The *UA* is an address of the address space which belongs to the parent *AD*. The *DP* defines relevant address space for devices connected within *AD*. When a packet is supposed to be delivered to or through the parent *AD*, *BGW* performs the operation, as shown in equation 1. While a packet with destination (*D*) and the source (*S*) address passes *BGW*, the *BGW* extends (*E:*) a relevant part of the source address (*S*) with configured *UA* (α). In the opposite direction, there is a symmetric operation. When a packet is delivered to or through child *AD* the destination address ($D\beta$) is reduced (*R:*) by *DP* (β) to get the destination address (*D*) within the child *AD*.

$$\frac{S}{D} \xrightarrow{E:\alpha} \frac{S}{D\alpha} \quad , \quad \frac{S\beta}{D} \xrightarrow{R:\beta} \frac{S}{D} \tag{1}$$

When we used these two simple operations within the tree structure of *ADs*, we come across two basic equations. The first one (2) describes a situation when a packet from the host *S* placed within AD α_n goes to the root AD (AD 0), and the second one (3) refers to a situation when the packet is delivered to its destination - the host D placed in the AD β_n. To make it simple, we can say that the source address of a packet extends when it passes through *BGWs* to the root AD, and the destination shortens when it goes to the designated AD. Figure 2 illustrates such a situation.

When the host *D* gets a packet, the source and destination addresses are exchanged (4) and packets with a response can be sent back to the host *S* the same way.

$$\frac{S\alpha_n\alpha_{n-1}\ldots\alpha_0}{D} \rightarrow \frac{D}{S\alpha_n\alpha_{n-1}\ldots\alpha_0} \tag{4}$$

The architecture, however, does not strictly relay on the incremental sequence of the level of *ADs*. Any *AD* can be omitted and the extension or reduction operation is avoided. Formally said, the *DP* (α) and *UA* (β) has an empty value. Practically, it means that *AD 4* can be directly connected to *AD 0*. There is also a possibility to make some shortcuts in the tree structure. Such a situation is illustrated in figure 2 with a dotted line. Two *ADs*, which are not a part of the same subtree, are connected via direct link. In that case the *BGW* performs several extension and reduction operations at once and delivers data directly to the neighbouring *AD*.

A. Session Identification

The communication between applications in the IP45 network expects the use of the traditional client server model. The application can use connection-oriented or connectionless abstract layers. When a client needs to connect to the server, it uses the server's address (to identify the interface of the host) and the port (to identify the application). In the traditional TCP/IP approach, every connection *C* is defined as a tuple *C(CA, CP, SA, SP)* where *CA* is a client IP address, *CP* stands for TCP port and *SA* and *SP* represents server address and server port. The connectionless communication is organized in a similar way. In the IP45 model of the network, the connection is defined as a triplet *C(SID, CP, SP)*, where *CP* and *SP* have the same meaning as before and *SID* refers to the *Session Identifier* of the connection. The *SID* is generated as a unique number when a client sends the request to establish connection or when the first packet is sent in connectionless protocols.

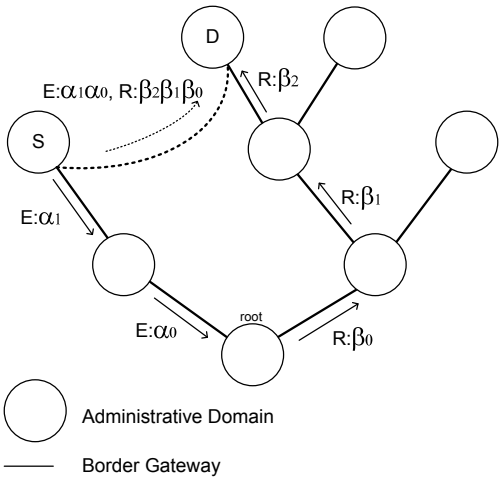

Fig. 2: The Model of IP45 Network

In fact, the address in IP45 network represents only a topological locator of the host (or its interface) where the first initial packet to establish connection is sent. *SID*, and practically the whole session, is independent of the source or destination addresses which means that the address can change on both the client's and the server's side, without breaking the session.

V. The Design of IP45 Protocol

The following sections discuss the details of all relevant components of IP45 design. The essential part of IP45 is *IP45 Host*. *IP45 Host* is a device able to address another *IP45 Host* and maintain the mutual exchange of IP45 packets between them. *IP45 Hosts* are organized into *Administrative Domains (ADs)*. As was discussed before, the *ADs* represent a tree structure in which the depth of the tree defines the *Level* of administrative domain. An example of such structure is depicted in figure 3. There are four different *ADs* - Global Internet, home network (HOME), organization network (ORG) and service provider (ISP). *AD 0* represents current public Internet. The home network *AD 4* is connected through ISP's *AD 2*. The organization network *AD 4* is directly connected to *AD 0*. In this case, *ADs of level 1-3* were omitted.

Packets inside of *AD* are delivered as regular IPv4 packets. Practically, the network, within a single *AD*, can be implemented from a simple L2 network (e.g. a small home network) up to a network with complex topology, with many routers (e.g. ISP's or corporate network). When a packet hits the

$$\frac{S}{D\beta_n\beta_{n-1}\dots\beta_0} \xrightarrow{E:\alpha_n} \frac{S\alpha_n}{D\beta_n\beta_{n-1}\dots\beta_0} \xrightarrow{E:\alpha_{n-1}} \frac{S\alpha_n\alpha_{n-1}}{D\beta_n\beta_{n-1}\dots\beta_0} \cdots \xrightarrow{E:\alpha_0} \frac{S\alpha_n\alpha_{n-1}\dots\alpha_0}{D\beta_n\beta_{n-1}\dots\beta_0} \tag{2}$$

$$\frac{S\alpha_n\alpha_{n-1}\dots\alpha_0}{D\beta_n\beta_{n-1}\dots\beta_0} \xrightarrow{R:\beta_0} \cdots \frac{S\alpha_n\alpha_{n-1}\dots\alpha_0}{D\beta_n\beta_{n-1}} \xrightarrow{R:\beta_{n-1}} \frac{S\alpha_n\alpha_{n-1}\dots\alpha_0}{D\beta_n} \xrightarrow{R:\beta_n} \frac{S\alpha_n\alpha_{n-1}\dots\alpha_0}{D} \tag{3}$$

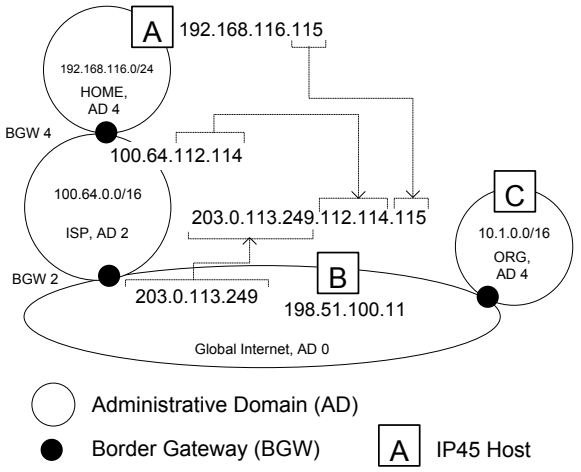

Fig. 3: Structure of IP45 Networks

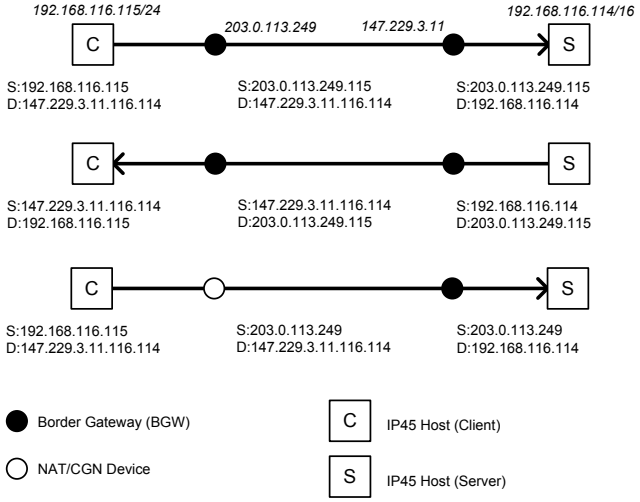

Fig. 4: Example of IP45 Packet Flow

border of *AD*, it is processed by a *Border Gateway* (BGW) that performs the reduction or expansion operation on the source or destination IP45 address and sends out the IP45 packet to the neighbouring AD.

From the user's perspective, the IP45 address of *IP45 Host* placed inside of AD seems to be a composition of two components. Firstly, a relevant part of the address which reflects the hierarchical structure of *ADs*, and secondly, the address assigned to the host within *AD*. With help of figure 3, we can illustrate the situation on an example in which host *A* wants to exchange some data with host *B*. The host *A* is connected through *AD 4* and *AD 2* to *AD 0* (root AD). The host *B* is placed directly in *AD 0*. When the host *A* sends a packet to host *B* with a destination address *198.51.100.11*, the source address of host *A* visible to host *B* will be *203.0.113.249.112.114.115*. If the host *B* (or any other host of course) wants to send a packet to the host *A*, it will just use *203.0.113.249.112.114.115* as the destination address.

The figure 4 shows another example of packet flow in IP45 network. There are three differed cases. In the first case the client *C* sends packet to the server *S*. Both the client and the server are placed behind BGWs. The figure shows how the source and destination address is changed while the packet is delivered to the server *S*. The second case describes the situation when the packet with response is delivered back to the client *C*. The third case depicts same situation as before with one difference. The client is placed behind ordinary NAT/CGN device.

A. IP45 Address, IP45 Stack and Session ID

The IP45 architecture introduces three new formats of address: *IP45 Address*, *IP45 Stack* and *Session ID*. From the user's and application perspective, the most important is the *IP45 Address*. The address is used by applications which connect to the remote host.

The maximum length of *IP45 Address* is 128 bits. The full address is represented by 16 octets. A single octet can be written as a single number with value 0-255 (eg. *0.0.0.0.0.0.0.0.0.0.203.0.113.249.115*). However, such notation is quite confusing. To make the address well-organized, the leading octets with zero value should be avoided (eg. *203.0.113.249.115*). In the case that the first twelve octets are set to zero value, the IP45 address would not be distinguishable from the IPv4 address. Therefore, IP45 address must be noted with at least one leading zero (eg. *0.203.0.113.249*).

In practice, IP45 address is not usually used in direct format but in the form of symbolic names. On the Internet, the conversion between these symbolic names and IP address is provided by the DNS system. There are two kinds of records in the DNS system. The *A record*, which returns a 4 byte number that represents the IPv4 address, and *AAAA record*, which returns a 16 byte number that represents the IPv6 address.

The IP45 architecture uses the DNS in the same way as it is used in IPv4 and IPv6. To keep the compatibility with applications (as will be discussed later in section V-D), IP45

reuses *AAAA records* to get information about the IP45 address. To distinguish IP45 address from IPv6 address, the architecture reuses a block of IPv6 address *0::/8* which was originally reserved by IETF for *IPv4-Compatible IPv6 Address*. Nevertheless, the block was never used and, in January 2006, was deprecated [37]. This means that such addresses can be easily reused for addressing in IP45 without any conflicts with existing IPv6 addressing schemes[2].

For converting IP45 address into its symbolic name, the situation is even easier. IP45 uses *PTR* records in the *in-addr.arpa.* zone. Thanks to the notation of IP45 address, there is no need to do any additional changes into the DNS. PTR can be used in IP45 in the same way as it is in IPv4, only the levels of delegation are extended according to the length of IP45 address. Thus, the address from the previous example (*203.0.113.249.115*) will be noted as *115.249.113.0.203.in-addr.arpa.*.

Require: $A45$ is IP45 Address
Require: $A4$ is IPv4 Address
Require: M is Mark Value
Require: S is IP45 Stack
procedure IN45TOSTCK45($IP45$)
  $M \leftarrow 0$
  while $A45[M] = 0$ **do**
    $M \leftarrow M + 1$
  end while
  $A4[0..3] \leftarrow IP45[M..(M+3)]$
  $M \leftarrow M + 4$
  if $M < 12$ **then**
    $S[(M-4)..11] \leftarrow A45[M..15]$
  end if
  $M \leftarrow 16 - M$
  return $A4, M, S$
end procedure

Algorithm 1: Converting IP45 Address into IPv4 Address, Mark and IP45 Stack

For packets which are delivered via the network, the *IP45 Address* is divided into *IP45 Stack* and *IPv4 Address*. The *IP45 Address* can be algorithmically (1) converted into *IPv4 address*, *IP45 Stack* and *Mark*, and vice versa. The relationship between those fields is depicted in 5.

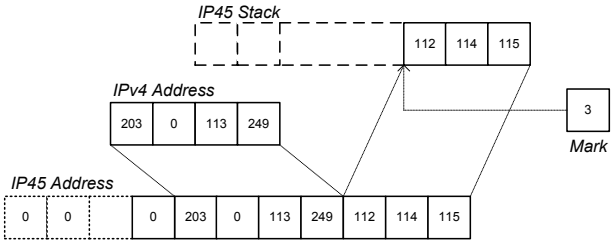

Fig. 5: Relationship Between IP45 Stack, IPv4 address, Mark and IP45 Address

The third new type of address introduced by IP45 is *Session ID (SID)*. *SID* is a 128 bit number that identifies all packets which belong to one session. The session could be a TCP/SCTP session, a UDP stream, a GRE tunnel, etc. Usually, SID is invisible to applications or users. To display *SID* makes sense only for debugging purposes. The basic format for displaying *SID* is the hexadecimal representation that can be shortened to the first and last 7 characters of its full notation divided by two dot symbols (eg. *ca0903..418d803*). The use of *SID* will be discussed in the V-D section.

[2]Except for the case of IPv4-Mapped IPv6 Address, which must be avoided by the configuration of the network.

B. IP45 Header Format

IP45 protocol uses a special header format of packets. The basic content of IP45 packets is not different from other protocols like IPv4 or IPv6. The packet starts with IP45 header area which contains fields needed by the protocol itself. After the IP45 header, the packet bears the header of transport protocol (TCP, UDP, ICMP, GRE, etc.) and data related to upper layers. The structure of the IP45 packet header is depicted in figure 6.

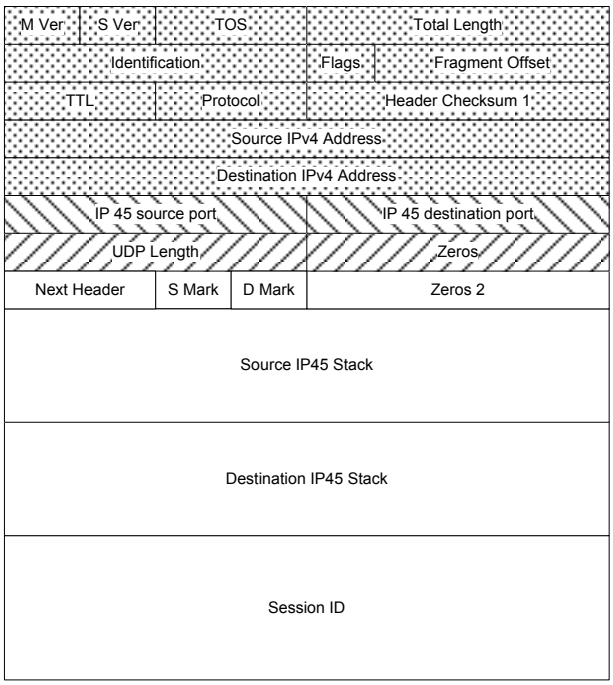

Fig. 6: The IP45 Header Format

The IP45 packet header consists of three parts. Firstly, there is a part of the header compatible with IPv4 header (dotted area). Any device supporting IPv4 protocol can treat IP45 packets as regular IPv4 traffic. Secondly, there is a part that allows IP45 packets to pass through all forms of NAT devices which treat them as if it was regular UDP traffic (hatching area). The last part of the packet header is IP45 specific. There are several fields which help to deliver IP45 packets to *IP45 Hosts* through *Border Gateways*.

Concerning the first part of the header, the meaning of the fields is mostly equivalent to the definition of IPv4 packet header format, as it is defined in RFC791 [38]. Similarly, the fields of the second part of the header are equivalent to the definition of UDP header format described in RFC768 [39]. In the following section we will focus only on the fields which are relevant to IP45 or which modify the meaning of the original fields.

M Ver, S Ver (4 bits): The fields indicate *Major* and *Sub* Version of the protocol. The value must be set to 4 for the *Major*, 5 for the *Sub*. The *M Ver* and *S Ver* fields together indicate the version of protocol, IP version 4.5 (IP45). The semantics of both fields is overloaded in comparison to the original specification in which the *S Ver* field replaces the original IHL field defined in RFC791 [38]. Originally, the value of the field determines the total size of IP header including the optional header. As the IP45 header does not support IP options, this value is always set to 5.

Protocol (8 bits): This field is always set to 17 which is the value reserved for UDP protocol on the upper layer. The field identifies only the transport protocol for devices which do not support IP45 protocol natively.

IP 45 Source/Destination Port (16 bits): Two fields which are always set by the sending host to the value 4. For devices that do not support IP45 natively, the fields identify the UDP source and the destination port[3]. Although, the sending host must set the value of those fields to 4, the receiving host can receive the packets for which the value of *IP 45 Source port* can be modified. It indicates that the *IP 45 Host*, who has originally sent the packets, is placed behind NAT. The use of such fields will be discussed in the section V-D.

Zeros (16 bits): the field filled with zeros. Originally, the field is reserved for UDP checksum in UDP header. Zero value indicates that no checksum is performed on UDP packets.

The fields described until now were more or less inherited fields which make IP45 packets compatible with IPv4 and NATs. The following fields extend the header and allow the IP45 protocol to use new features. For most of them, we provide only a brief description. Sections V-C and V-D will clear up the meaning and use of the fields in detail.

Next Header (8 bits): This field identifies the transport protocol. The meaning is the same as the field *Protocol* used in IPv4 or the field *Next Header* used in IPv6. Similarly to IPv4 or IPv6, the concept of extension headers can be used here as well. Also the use of shim headers like ESP or AH may be possible.

Src/Dst Mark (4 bits): The fields determine the number of valid bytes of *IP45 Source/Destination Stack*. In other words, it indicates the current size of the stack. The fields are primarily used by *Border Gateways*.

Zeros 2 (16 bits): The zero values to align the header.

Source/Destination IP45 Stack (96 bits): The field, where a relevant part of the *Source* or *Destination IPv4* address is stored as packets, travels through *Administrative Domains*. *IP45 Stack*, together with *IPv4 address* and *Mark*, represent the *IP45 Address*. The algorithm that shows how the *Border Gateways* manipulate with the *IP45 Source* and *IP45 Destination Stack* will be discussed in section V-C.

SID (128 bits): A unique randomly generated ID to identify the session between two *IP45 Hosts*. SID will be discussed in detail in section V-D.

[3]According to IANA, UDP port 4 is not assigned, so it can be used without causing any conflicts.

C. IP45 Border Gateway (BGW)

The role of the *Border Gateway* (BGW) is to allow packets to traverse between *Administrative Domains*. In general, BGW works as a regular IPv4 router with an extra code performing operations on the interface where the *Upstream Address* (UA) is configured. Algorithm 2 shows the operation performed by *BGW* according to the model described in chapter IV.

Require: UA is Upstream Address
Require: DP is Downstream Prefix
Require: DL is Length of Downstream Prefix in Octets
Require: SA is Source IPv4 Address
Require: DA is Destination IPv4 Address
Require: SM is Value of Source Mark
Require: DM is Value of Destination Mark
Require: SS is Source IP45 Stack
Require: DS is Destination IP45 Stack
Require: CS is Checksum

1: **procedure** BGWPASSPKT
2: **if** $not\ Check IP54 Fields$ **then**
3: $ProcessAsIPv4Packet()$
4: **end if**
5: **if** $SA\ matches\ DP/DL$ **then**
6: $SM \leftarrow SM + DL$
7: **if** $SM > 12$ **then**
8: DropPacket()
9: **end if**
10: $SS[(11 - SM)..(DL - 1)] \leftarrow SA[0..(DL - 1)]$
11: $SA \leftarrow UA$
12: **end if**
13: **if** $DA = UA\ \ \&\ \ DM > 0$ **then**
14: $DM \leftarrow DM - DL$
15: $DA[0..(DL - 1)] \leftarrow DP[0..(DL - 1)]$
16: $DA[DL..3] \leftarrow DS[(11 - DM)..(11 - DM + DL)]$
17: **end if**
18: UpdateCksum(CS)
19: **end procedure**

Algorithm 2: Operations Performed on IP45 Border Gateway

At the beginning, the algorithm checks whether the processed packet is a valid IP45 packet. This is proceeded by checking several items for proper values. The *Protocol* field must be set to value 17 (protocol UDP) and either *IP 45 Source Port* or *IP 45 Destination Port* must be set to 4. To be sure that the packet is a valid IP45 packet, the *Zeros 2* field is verified. If any of those conditions are not satisfied, the packet is processed as a regular non IP45 packet.

The core algorithm is divided into two parts. Firstly, the part which deals with the situation when IP45 packets traverse from the child to the parent *AD*. As the entry condition, the algorithm checks whether the first octets defined by *Downstream Prefix Length* are equivalent in the *Source Address* and in configured *Downstream Prefix*. If the condition is satisfied, the value of the *Source Mark* is increased by value of *Downstream Prefix Length*. If the value of *Source Mark* is higher than 12, it indicates that there is no more space in the *IP45 Source Stack* and the packet is dropped. Depending on the implementation, the original host can be informed by an ICMP message that the packet could not be delivered. Nevertheless, in major cases the packet is passed to the next phase where the *Source IP45 Stack* is extended on the left side with the top octets from the *Source IPv4 Address*. The number of octets to extend is the

value which is expressed as 4 - *Downstream Prefix Length*. Finally, *Source Address* is replaced by *Upstream Address*.

To clarify further, figure 7 shows an example of the operation on the BGW where the *Downstream Prefix* is configured as 100.64.0.0/16. This implies that the value of the *Downstream Prefix Length* is 2 and the *Upstream Address* is configured as 203.0.113.249. The BGW receives a packet with the *Source IPv4 address* 100.64.112.114 and the *Source IP45 Stack* set to 115. The BGW performs the operation described above and the packet leaves the router's interface with the *Source IPv4 address* set to 203.0.113.249 and the *Source IP45 Stack Address* extended to 112.114.115.

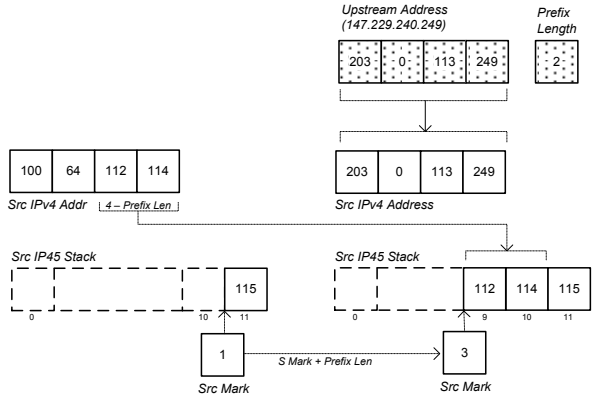

Fig. 7: Modification of the Fields in IP45 header *when a packet is received on* Downstream Prefix *interface*

When a IP45 packet goes in the opposite direction, from the parent to the child *AD*, a symmetric operation is performed. At the beginning the value of the *Destination Mark* is checked. The zero value indicates that the packet already reached the target host. For packets with a value greater than zero, the *BGW* performs a reduction of the *Destination IP45 Stack* by simply decreasing the value of the *Destination Mark* by the value of *Downstream Prefix Length*. The *Destination IPv4 Address* is composed from the reduced part of the *Destination IP45 Stack* which is placed to the top octets. The remaining (bottom) octets are filled with the bottom octets from the *Downstream Prefix*. Again, in an example with the configuration of BGW used previously, the BGW receives a packet with the *Destination IPv4 Address* 203.0.113.249 and *Destination IP45 Stack* set to 112.114.115. At the output the *Destination IP45 Stack* is reduced to 115, and the resulting *Destination IPv4 Address* is 100.64.112.113. The 100.64 is obtained from the *Downstream Prefix* and 112.113 is the reduced part of the *Destination IP45 Stack*. The operation is depicted in figure 8.

In general, the whole algorithm contains a few trivial comparisons and shifts of the memory blocks. As the consequence, the algorithm can be easily implemented as a specialized high speed hardware block or several parallel blocks implemented in an FPGA or ASIC chip.

D. IP45 Host

The IP45 host is responsible for sending and receiving IP45 packets, managing sessions and providing API for applications. The interface between an application and the network interface

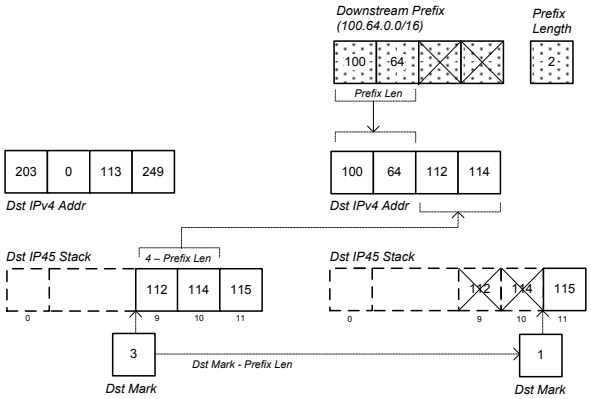

Fig. 8: Modification of the Fields in IP45 header *when a packet is received on* Upstream Address *interface*

provided by operating system is typically provided through universal sockets API. When a new protocol is introduced, the typical solution is to implement a new socket type that allows the application to use the protocol. However, this approach requires a redesign of all applications. For that reason IP45 reuses the existing IPv6 socket interface. Once the application is prepared to use IPv6 sockets, it can automatically use the IP45 protocol without any modification.

When an application needs to establish a connection to a remote host, it uses a system call in which the address of the remote host is handed over. In case of IP45, the application makes a standard call to open/bind the IPv6 socket; it uses overloaded semantics of the IPv6 address as discussed in chapter V-A. For example, if the application needs to connect to the IP45 host with the address *203.0.113.249.112.113.115*, the connection is made to the address *::00cb:0071:1870:7173*, which is the hexadecimal IPv6 notation of the same address. Through the address prefix *::/8* the system is able to recognise that the requested type of socket is IP45 and that the call is internally interpreted as the IP45 socket operation.

Require: DA is Destination Address
Require: S is Socket structure
1: **procedure** INITSOCK(S,DA)
2: **if** $DA\ matches\ ::0/8$ **then**
3: $S.Type \leftarrow IP45$
4: $S.SessionID \leftarrow Random()$
5: $(A4, M, S) \leftarrow In45ToStack45(DA)$
6: $S.RemoteAddr \leftarrow A4$
7: $S.RemoteMark \leftarrow M$
8: $S.RemoteStack \leftarrow S$
9: $S.RemotePort \leftarrow 4$
10: $InitSockIPv4(S)$
11: **else**
12: $S.type \leftarrow IPv6$
13: $InitSockIPv6(S)$
14: **end if**
15: **end procedure**

Algorithm 3: Initialization of IP45 Socket on IP45 Host

The algorithm 3 displays an operation performed during the IP45 socket initialization. After identifying that the operation is requested to IP45 socket, initial steps are performed. Firstly,

a randomly generated *SID* is assigned to the internal socket structures. Then the remote address is decomposed into *Remote IPv4 Address* (*S.RemoteAddr*), *Remote Mark* (*S.RemoteMark*) and *Remote IP45 Stack* (*S.RemoteStack*) as was described in the section V-A. Lastly, the *Remote IP45 Port* (*S.RemotePort*) is set to value 4. The rest of the initialization procedure is the same as for the common IPv4 socket.

After initialization, the IP45 socket is ready to send/receive packets to/from the network via the standard system calls.

Require: S is Socket Structure
Require: P is Packet Buffer
procedure SENDPKT
  if $S.type\ is\ ip45$ **then**
    $P.SessionID \leftarrow S.SessionID$
    $P.DstIPv4Addr \leftarrow S.RemoteAddr$
    $P.DstIP45Stack \leftarrow S.RemoteStack$
    $P.DstIP45Mark \leftarrow S.RemoteMark$
    $P.DstIP45Port \leftarrow S.RemotePort$
    $P.SrcIP45Port \leftarrow 4$
    $P.SrcMark \leftarrow 0$
    $P.Protocol \leftarrow UDP(16)$
    $P.Zeros2 \leftarrow 0$
    $SendIPv4Pkt(P, S)$
  else
    $SendIPv6Pkt(P, S)$
  end if
end procedure

Algorithm 4: Sending IP45 Packet

As it is shown in algorithm 4, several operations must be performed to send a packet. If the socket was initialized as IP45, the procedure begins to prepare the header of the IP45 packet. The items like *Destination IPv4 Address*, *Destination IP45 Socket*, *Destination Mark*, *Destination IP45 Port* are directly copied from their "remote" equivalents stored in the socket structure. The *Source Mark* and *Source IP45 Stack* are set to zero values. The remaining fields in the IP45 packet (Length, Checksum, Source IPv4 address, etc.) are set by standard IPv4 procedures as the packet is passed to the output interface.

The receiver must at first distinguish IP45 from non-IP45 IPv4 packets. The same check we used as the entry condition in the BGW algorithm. The whole procedure of receiving packets is shown in algorithm 5. After the entry check detects whether the packet is a valid IP45 packet, the lookup for the socket structure is performed. Firstly, on existing IPv6 sockets and, if it fails, the lookup is performed within structures representing IPv4 sockets. This way the IP45 protocol can be connected to the listening socket on both, IPv6 and IPv4, sockets. If the representation of the socket is not found in the system, the ICMP error message is returned back to the communicating peer. Otherwise, items in the internal socket structures are updated and the content of packet can be handed to the application.

The algorithms described above expect that IP45 is implemented as a part of socket functions which are, in most systems, implemented as a part of the *kernel code* as it is shown in figure 9.

The main advantage of this approach remains that the IP45

Require: P is Received Packet
Require: S is Socket Structure
Require: PS is Protocol Specific Structure
procedure RECVIP45PKT(P)
  if $not\ CheckIP54Fields$ **then**
    $RecvIPv4Packet(P)$
  end if
  $PS \leftarrow GetProtocolSpecific(P.NextHeader, P)$
  $S \leftarrow LookupIPv6(P.SessionID, PS)$
  if $not\ S$ **then**
    $S \leftarrow LookupIPv4(P.SessionID, PS)$
  end if
  if $S.type\ is\ ip45$ **then**
    $S.RemoteAddr \leftarrow P.SrcIPv4Addr$
    $S.RemoteStack \leftarrow P.SrcIP45Stack$
    $S.RemoteMark \leftarrow P.SrcIP45Mark$
    $S.RemotePort \leftarrow P.SrcIP45Port$
    $HandleToUpperLayer(P)$
  else
    $ICMPErrorMessage()$
  end if
end procedure

Algorithm 5: Receiving IP45 Packet

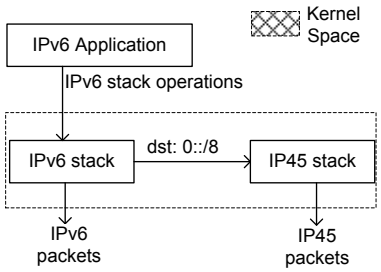

Fig. 9: IP45 Data Flow in the Kernel

traffic is processed directly by system calls without any extra overhead. Nevertheless, there are two barriers. Firstly, to make these changes in major operating systems is not possible due to enclosed development. Secondly, even if the kernel source is available, patching and recompiling is an option only for advanced users.

Another possibility is to add *the IP45 host's support* that is executed as *userspace process*. This can work almost on all operating systems including those whose development and access to the sources are limited. In such case, IP45 packets are directed to IPv6 socket as a regular IPv6 traffic. When the traffic passes through the system routing table, the traffic that matches the prefix 0::/8, is diverted to the user space process as shown in figure 10

The process replaces the original IPv6 header by IP45 header and sends it out as regular IPv4/UDP traffic. At the same time, the user process listens to the UDP socket for incoming IP45 packets. When the packets arrive, they are reshaped back to the IP45 packet, and injected to the system as a regular IPv6 packet. The disadvantage of this approach is that the packets have to be passed within the operating system twice. It could be reflected in the resource consumption which would be higher in comparison to IP45 implemented in the kernel. It would not be a problem on the client systems

where the traffic usually does not cross gigabit speed, but the performance could be an issue on servers where traffic can reach up to several Gb/s.

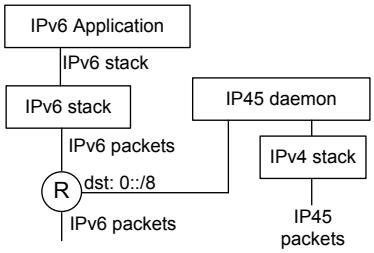

Fig. 10: IP45 Data Flow Using IP45 Translator

E. Addressing in Administrative Domains

As it might seem that *AD* can use any address space, there are some conditions to be kept. The address space used in any child *AD* must not collide with the one in *AD 0* and If two neighbouring *AD* uses the same address space it might cause problems on the *BGW* device.[4].

In order to avoid the problem of conflicting address space, the IP45 explicitly prescribes blocks of addresses to be used on a particular level of *AD*. The table I shows the division of existing address space in IP45.

TABLE I: Address Space Used in Different AD Levels

AD level	purpose	used address
0	global Internet	Public
1	reserved	-
2	ISP's network	Shared, RFC 6598 [40]
3	reserved	-
4	home, organisation	Private, RFC 1918 [41]
5-11	reserved	-

The division of address space matches the actual standards, recommendations and common practice of using address space within the network behind NAT or CGN. Currently, address space is explicitly defined only for levels 0, 2 and 4. The address space used on remaining levels might be specified in future. However, this in fact does not prevent the use of, for example, private addresses (RFC 1918 [41]) on levels 1,3 or 5, but the administrator must be aware of potential problems.

One of the things the administrator of *AD* has to take into consideration is the prefix size used for addressing within *AD*. The maximum level of the *ADs* is limited to 12. It is a special case when all *ADs* in the subtree use *Upstream Prefix (UP)* with a size of 24 bits. In such cases only 252 nodes can be addressed within every *AD*. On the other hand, if the size of *UP* in every *AD* was set to 8 bits, the *AD* would count more than 16 million devices; however, the maximum number of *AD* levels would be limited to 4 (including the root one, *AD 0*). In practice, the size of *DP* in every *AD* will be decided by the administrator and different *ADs* on a different level will use the size of *DP* according to their need. Returning back to figure 3 from section V, the administrator of *AD 4* decided to use a *DP* of size 24 bits that might be enough for example for a home network; but, in *AD 2* was decided to use a *DP* with the size of 16 bits (ISP's network) which allows to connect approximately 60 thousand devices.

[4]The same route entry on two different router interfaces.

VI. Features and Limitations

In this section we focus on some interesting features provided by IP45 design, and make the comparison to other network architectures. We mainly point out the similarities and differences with IPv4 and IPv6 because these two architectures have very mature implementations and provide a significant record of architectural and operational experience. In table II there is a raw overview of the key features provided by IP45 design and its comparison with others. Beside IPv6 and IPv4, IPv4+4 [23] and IPNL [12] are added as a good representative of alternative approaches. The following sections will discuss the features in detail.

A. Independent and Hierarchical Addressing

Every *Administrative Domain (AD)* in IP45 can use its own address space that can overlap the address space of another *AD* (except for *AD 0*). The final address of devices inside of *AD* is composed of the address assigned to a device and the address derived from the topology of a network. Once an *AD* is either reconnected to another parent *AD* or connected to multiple parent *ADs*, the devices inside of the *AD* are automatically reached via a new address(es). The IP45 addresses are *hierarchical (topological)*, and *independent*. Those two attributes bring some interesting benefits:

The number of routing information can be decreased to a minimum. Even if the routing inside of *AD* can be very complex with thousands of routes, a child *AD* can be designated by a single route record available in the parent *AD*. In current Internet routing architecture with the BGP *Autonomous System (AS)* as the top of its hierarchy, every *AS* can be represented by just one routing entry in the global routing table (*AD 0*)[6].

The size of the routing entry is limited to four bytes (the size of IPv4 address) which is enough to cover all routing needs inside of the *AD*. At the same time, it is favourable to routing devices which require lower memory and CPU in comparison with when the full address has to be processed.

The change of upstream ISP represents a simple reconnection of *BGW* to a different parent *AD* without any need of readdressing all devices inside the *AD*. The site multihoming can be implemented similarly when the *BGW* has configured multiple *Upstream Addresses* to different parent *ADs*.

In IPv6, creating the hierarchy of address space was an integral part of IPv6 from the very beginning [42]. The first 16 bits for *top level aggregation*, the next 32 bits for *next level aggregation*, 16 bits for aggregation within the site and, the remaining 64 bits are determined to address nodes within the L2 subnet. However, IPv6 works on every level of aggregation with complete IPv6 addresses. From the resource perspective, every entry in the BGP database, routing table or neighbour cache occupies all 16 bytes, even if only a small part of

[5]Only for servers as discussed in section V-D

[6]Traffic engineering or business needs might break that rule.

TABLE II: Features and Network Architectures

Feature	IP45	IPv4	IPv4 + NAT	IPv6	IPv4+4	IPNL
Hierarchical address space	Yes	No	No	Yes	No	Yes
Address space independency	Yes	No	Yes	No	Yes	Yes
End-to-end addressing	Yes	Yes	No	Yes	Yes	Yes
IPv4 protocol compatibility	Yes	Yes	Yes	No	Yes	Yes
NAT (IPv4) compatibility	Yes	Yes	Yes	No	No	Yes
IPv4 API compatibility	*[5]	Yes	Yes	No	No	No
IPv6 API compatibility	Yes	No	No	Yes	No	No
Session independency	Yes	No	No	No	No	No

IPv6 address is relevant for a particular function. The IPv6 addressing scheme is *hierarchical* but it does not provide *address space independency*.

The address space independency used in IP45 is to approach taken in RINA [33] architecture. The address space used within *AD* resembles the one used within *DIF*, *BGW* may refer to the IPC facility managing the routing between (N)-DIFs, and *Upstream Address* is similar to *Point-of-Attachment*.

Nevertheless, *address space independency* brings a problem. The IP45 host does not exactly know his own IP45 address. This is not fundamental for most of the common protocols like HTTP, IMAP, POP3, RDP, but the protocols which use the IP address inside of the protocol can have some problems. An application can learn its own IP45 address from the communicating peer on its own as a part of the communication exchange. Another option and our proposed solution is the extension of ICMP protocol with functionality that can be simply called "tell me my IP45 address". Any IP45 host who enables ICMP messages can pass the IP45 address of querying host back.

B. Protocol and Application Compatibility

IP45 was intentionally designed to be compatible with existing IPv4 protocols and, where possible, compatible with NATs. This architectural choice has quite interesting outcomes with a significant impact on the process of IP45 deployment. It was our intention not to use the word transition, here. Contrary to IPv6, IP45 does not require to switch the Internet to a new protocol completely. IP45 can be deployed incrementally in small steps according to need. At the first stage *BGWs* must be deployed only for servers. It does not matter if a client is placed behind a NAT or several NATs. This approach is very different in comparison to IPv4+4 in which middle boxes - *realm gateways* - must be deployed for both clients and servers.

In IPv6 the situation is even more complicated. IPv6 protocol was designed as incompatible with IPv4. The decision provided an advantage of designing IPv6 from scratch, however, both protocols must run side by side for a certain time and everything has to be implemented, configured and managed twice - routing, peering, firewalls, address management/assignment, CPEs, first-hop-security, QoS, etc. That obviously increases the cost of running network and its complexity. As a result, IPv6 traffic often passes via different paths in comparison to IPv4 and some services may indicate slower performance via IPv6 [43], [44], [45]. The problem is even more significant when any of the common transition techniques (Teredo [46], ISATAP [47], 6to4 [48]) is used.

The performance issues and failing of IPv6 connections were a frequent excuse for service and content providers not to deploy IPv6, the IETF introduced the *Happy Eyeballs* [49] mechanism. The application tries to establish a parallel connection via both, IPv4 and IPv6, and the one that is faster is kept for transferring data. However, the application has to be redesigned to use *Happy Eyeballs*, so only a few applications (e.g. web browsers) support *Happy Eyeballs* today. Trying to decrease the network complexity, IETF brought also DNS64 and NAT64 [50], [51]. These techniques allow clients to use only IPv6 protocol for accessing IPv4 services. Unfortunately, many applications do not work properly in an IPv6 only environment [52]. So the IETF created another mechanism, 464XLAT [53], that provides an illusion of IPv4 connectivity to the application. Remaining solution is DS-Lite [54] which tries to solve the same problem as the combination of DNS64, NAT64 and 464XLAT.

With the IP45 architecture, many of these obstacles can be avoided. For IP45, the core of the Internet can remain untouched. There is no reason to upgrade all routers and maintain two routing hierarchies across the whole Internet. There is also no reason for adding sophisticated transition mechanisms which are present in IPv6. In IP45, there are only two places to be modified or upgraded - hosts and NAT device (BGW) in front of servers. Another positive side effect of the IPv4 compatible design is in sharing the same data path by both IPv4 and IP45 packets. This minimizes the risk that the delivery of IP45 packets will have different parameters, e.g. reliability, round trip time, etc.

The expected position of *BGW* is on the existing NAT device (home CPE, corporate CGN). Since the operation performed by *BGW* is quite simple, the extension of the existing implementations should not be a big issue. When a device begins to support BGW operations, the IP45 traffic will be automatically processed by *BGW* code. Unlike NAT, *BGW* is a stateless engine that saves resources on the device (memory, lookups) and it is easy to implement. So the motivation to extend NAT devices by *BGW's* code is quite apparent.

The disadvantage of backward IPv4 and NAT compatibility is in a not well-arranged packet header. If we look at the packet's fields, as was clarified in chapter V-B, we can find that almost 14 bytes of the IP45 packet header do not carry any meaningful information. This problem could be solved in future, if the IP45 traffic was significant. The IP45 packet header can be reshaped into a more efficient structure by removing useless filed inclusive checksum. Contrary to IPv6, such shortened packets will not be used along the whole path between the source and destination, but only between *IP45*

hosts or *IP45 capable routers* to exchange packets on the link layer. In some suitable way (for example through modified ARP protocol), the nodes inform each other that they are able to process packets with a shortened IP45 header. If the nodes do not indicate such capability, the standard IP45 header will not be used. This way, the bandwidth on links can be saved and routers do not need to check and recompute checksums.

So far we have discussed only protocol compatibility - whether IP45 packets can be delivered over existing IPv4 and NAT infrastructure. A different matter is the application compatibility. Every new protocol typically proposes a new socket API for using the protocol. It means that all applications have to be redesigned or at least recompiled to use such API. In the same way IPv6 API was designed. Since the past decade, most of operating system, programming languages and applications have already adopted it and started using it with no significant troubles. For that reason IP45 reuses the same API. Any application that has already been redesigned for using IPv6 API can use IP45 immediately without an additional modification (discussed in V-D).

C. Session Independency

In IPv4 and IPv6 every session is defined by the source and destination address, protocol and protocol specific identifier such as UDP or TCP port. In IP45 the source and destination address is replaced by the single session identifier *SID*. This brings several interesting options. The *SID* can be used for the construction of load balancers or a simplified statefull firewall with no need to know the content of the transport protocol. Mobility and multihoming is another, more interesting use of *SID*. The IP45 address of the session can smoothly change without breaking the running session. When an *AD* is reconnected to some other parent *AD*, all sessions keep running. If the protocol supports some kind of pervasion mechanism such as *TCP keepalive*, the simplified multihoming and mobility of the client can work automatically. The solution is very similar to small site multihoming provided by SHIM6 [55] or DoA architecture [27]. Some simmilar functionality can also be provided by *IPv6-to-IPv6 Network Prefix Translation* [14]. However, when the site is reconnected to another one, all ongoing sessions are broken.

To provide fully-fledged mobility in IP45, it must be equipped with extra mechanisms. The problem of server mobility comprises two partial problems. Firstly, to make an initial connection to a proper IP45 address, secondly, to properly "redirect" the ongoing session to the new IP45 address. While the former can be solved via DNS, the latter case requires extending by a simple keep-alive protocol to inform peers about a new IP45 address. Contrary to the traditional multihoming approach like IPv6 mobility [56], or LISP [29], there is no need of a meeting point (e.g. *home agent* in mobile IPv6) or a distributed mapping database (RLOC:EID database in LISP).

The *SID* may also bring some problematic issues. The *SID* is created by clients as a randomly generated 16 byte number when the session is initialized. If two hosts create the same conflicting *SID* at the same time and try to connect to the same server using the same source and destination ports, the server will not be able to distinguish between them. Although, the probability of such collision is extremly low, it is still possible. For that reason IP45 extends ICMP messages by a new type of message indicating the collision in *SID*. If the server during the initial handshake recognises that the same *SID* is already used, an ICMP message is sent back to the client. The client can create a new *SID* and make a new attempt to establish the connection. A problem that remains is that this method can be used only in protocols performing the initial handshake.

Another problem of the *SID* is related to the security. If an attacker has an access to the data of running communication, the *SID* can be easily wiretapped. Then, the attacker can redirect the traffic from both the client and server, to his own IP45 address. A similar situation, but less presumable, may happen if the attacker guesses the *SID* and other communication parameters like ports, tcp sequence number etc. To prevent such situations, the socket implementation on both the client's and server's side can use a special socket option which disallows the changing of the address during the session. If this option is set, the packets from different peers are ignored. Such an option enables the application to decide whether preffers to avoid possible hijacking or use features regarding mobility. The security of the session can be solved on the higher level, for example, via IPSEC or SSL/TLS.

VII. Implementation and Testing

The description of implementation and evaluation of the protocol is relatively a complex topic which would exceed the limitation of this paper. For that reason we decided to provide only basic information and the in-depth description will be provided in a separate paper. Based on the proposed design, we have implemented all necessary components as follows:

The **IP45 Host** is implemented as a patch for Linux kernel sources and it is available for kernel 3.x.x. After applying the patch and rebuilding the kernel sources, the host is able to act as a fully capable IP45 host. The userspace *IP45 Host* implementation is available for remaining platforms. It is designed as the deamon for Unix-like POSIX platforms (Linux, OS X, FreeBSD) or as the system service for a Windows platform. For Linux and Windows the pre-built binary packages are prepared.

The **Border Gateway** is implemented as a module to Linux iptables. The module can be easily added into any Linux distribution using a non-archaic kernel (2.6 and higher). For routers based on OpenWrt we created pre-built packages that can be easily installed into any OpenWrt router via a router's web interface.

Besides this, there are some optional components making the testing or debugging of IP45 more comfortable:

Libc Extension for Glibc updates system functions which are responsible for converting the human notation of the IP address into a binary number and back. This patch extends the standard system calls (*inet_ntop*, *inet_pton*) to be able to convert the IP45 address as well.

Extension for tcpdump and wireshark which allows both tools to display IP45 packets in human-readable format.

All developed components, project repository and documentation are permanently available on the project's web page [57]. For software packages, we developed a cross-platform build system that automatically rebuilds all components and packages.

For testing the behaviour of IP45, we started with a small laboratory setup consisting of several servers, border gateways and NAT devices. In a very short time we discovered that there is little to test under artificial conditions. The performance of all components was almost the same as for IPv4 traffic (several Gb/s). Therefore we decided to step outside of a laboratory and started to use IP45 in the wild. To demonstrate the basic functionality of IP45, a short video record was prepared. The record demonstrates two communicating *IP45 Hosts* placed in three different ISP's networks behind 3 NATs and BGWs. The first host (Linux with the kernel patch) connects to the remote desktop of the second host (OS X with userspace daemon) using VNC. The second host connects back to the first one with secure shell (ssh). During the session, the link was connected to the ISP, then it was intentionally broken and the record demonstrates how the traffic was redirected to the backup ISP without breaking both established sessions. The record is also publicly available on the project's web site [57].

Today, all servers on which the development of IP45 is performed are accessible only via IP45, so we are made to use IP45 and the IP45 traffic is currently mixed with a production traffic and production environment.

VIII. Summary and Future work

This paper describes a hierarchical network architecture which extends IPv4 address space by regaining end-to-end connectivity in NATed networks. It is backwards compatible with IPv4 and NATs and can be deployed only in places where it is needed. Contrary to IPv6, it does not require a complete, challenging and costly transition to a new protocol. From the architectural perspective, the IP45 implements address space and session independency what brings interesting build-in features like simplified mobility and multihoming.

To demonstrate the feasibility of the IP45 architecture, we implemented all necessary components for commonly used platforms (Windows, Linux, OS X) and we started using the protocol on daily basis together with ordinary applications (mail, web, ssh, rdp). Our implementation also proved that the IP45 host support can be easily implemented even on "closed" platforms without any need for modifications of existing applications.

In the near future, there are several areas we would like to focus on. Firstly, extend the spread of *IP45 host* implementation to as many platforms as possible. In order to obtain broader experience in a heterogeneous environment, we would like to extend the support to mobile platforms and make packages available on standard application delivery platforms such as the *App Store*, *Google Play* and *Windows Phone Store*. We would also like to pursue a profound research of possibilities provided by *SID*. We feel that this area is full of the potential to unearth interesting and inspirational capabilities which have not been explored enough. In the area of standardization, we would like to submit the protocol specification into IETF as the experimental RFC. However, it may be quite difficult since it is not likely that IETF will be eager to approve the solution which prolongs the existence of NAT.

References

[1] J. Curran, "An Internet Transition Plan." RFC 5211 (Informational), July 2008.

[2] G. Houston, "IPv6: IPv6 / IPv4 Comparative Statistics." Online http://bgp.potaroo.net/v6/v6rpt.html. Accessed May 02, 2014.

[3] "Google IPv6 Statistics." Online http://www.google.com/intl/en/ipv6/. Accessed May 02, 2014.

[4] "6lab.cz - Statistics." Online http://6lab.cz/live-statistics/web/. Accessed May 02, 2014.

[5] "6lab.cisco.com - Statistics." Online http://6lab.cisco.com/. Accessed May 02, 2014.

[6] "IPv4 Market Group." Online http://ipv4marketgroup.com/home/, 2013. Accessed January 19, 2014.

[7] E. Osterweil, S. Amante, D. Massey, and D. McPherson, "The great IPv4 land grab: resource certification for the IPv4 grey market," HotNets-X, (New York, NY, USA), pp. 12:1–12:6, ACM, 2011.

[8] "Verizon DSL moving to CGN, NANOG Mailing List Archive." Online http://www.gossamer-threads.com/lists/nanog/users/162187. Accessed May 02, 2014.

[9] "Customers fume as BT introduces IP sharing," *PC Pro magazine*, vol. 2013, no. 1, 2013.

[10] I. Pepelnjak, D. Markovič, and D. Spasojevič, "Small Site Multihoming." Online http://stack.nil.com/ipcorner/SmallSiteMultiHoming. Accessed January 19, 2014.

[11] L. Daigle and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation." RFC 3424 (Informational), Nov. 2002.

[12] P. Francis and R. Gummadi, "IPNL: A NAT-extended internet architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 31, pp. 69–80, Aug. 2001.

[13] O. Troan and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6." RFC 3633 (Proposed Standard), Dec. 2003. Updated by RFC 6603.

[14] M. Wasserman and F. Baker, "IPv6-to-IPv6 Network Prefix Translation." RFC 6296 (Experimental), June 2011.

[15] J. Day, *Patterns in Network Architecture - A Return to Fundamentals.* Prentice Hall, 2008. ISBN 978-0-13-225242-3.

[16] V. Jacobson, "LNAT — Large scale IP via Network Address Translation." Working Draft. Online ftp://ftp.ee.lbl.gov/.van/nat.pdf, 1992. Accessed May 02, 2014.

[17] B. M. Sousa, K. Pentikousis, and M. Curado, "Multihoming Management for Future Networks," *Mob. Netw. Appl.*, vol. 16, pp. 505–517, Aug. 2011.

[18] P. Francis, "PIP Near-term Architecture." RFC 1621, May 1994.

[19] M. McGovern and R. Ullmann, "CATNIP: Common Architecture for the Internet." RFC 1707, Oct. 1994.

[20] R. Callon, "TCP and UDP with Bigger Addresses (TUBA), A Simple Proposal for Internet Addressing and Routing." RFC 1347 (Informational), June 1992.

[21] Z. Wang, "EIP: The Extended Internet Protocol." RFC 1385 (Historic), Nov. 1992. Obsoleted by RFC 6814.

[22] C. Pignataro and F. Gont, "Formally Deprecating Some IPv4 Options." RFC 6814 (Proposed Standard), Nov. 2012.

[23] Z. Turanyi and A. Valke, "IPv4+4," *2012 20th IEEE International Conference on Network Protocols (ICNP)*, vol. 0, p. 290, 2002.

[24] A. Campbell, J. Gomez, S. Kim, A. Valko, C.-Y. Wan, and Z. Turanyi, "Design, Implementation, and Evaluation of Cellular IP," *Personal Communications, IEEE*, vol. 7, no. 4, pp. 42–49, 2000.

[25] C. Topal and C. Akinlar, "Implementing IPv4+4 Addressing Architecture with IPv4 LSRR Option for Seamless Peer-to-Peer (P2P) Communication," vol. 4742 of *Lecture Notes in Computer Science*, pp. 809–820, Springer Berlin Heidelberg, 2007.

[26] M. O'Dell, "GSE - An Alternate Addressing Architecture for IPv6." Internet-Draft, 1997. Draft draft-ietf-ipngwg-gseaddr, no. 00.

[27] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker, "Middleboxes no longer considered harmful," in *Proc. USENIX OSDI*, (San Francisco, CA), December 2004.

[28] V. Veselý and M. Švéda, "Comparison of proposals suggesting internet architecture change," in *Sborník příspěvků Mezinárodní Masarykovy konference pro doktorandy a mladé vědecké pracovníky 2013*, pp. 1–11, Siemens A.G., 2013.

[29] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "The Locator/ID Separation Protocol (LISP)." RFC 6830 (Experimental), Jan. 2013.

[30] I. Castineyra, N. Chiappa, and M. Steenstrup, "The Nimrod Routing Architecture." RFC 1992, Aug. 1996.

[31] D. R. Cheriton and M. Gritter, "TRIAD: A Scalable Deployable NAT-based Internet Architecture," tech. rep., Stanford University, 2000. Accessed May 02, 2014.

[32] S. Guha and P. Francis, "An end-middle-end approach to connection establishment," SIGCOMM '07, (New York, NY, USA), pp. 193–204, ACM, 2007.

[33] J. Day, I. Matta, and K. Mattar, "Networking is IPC: A Guiding Principle to a Better Internet," CoNEXT '08, (New York, NY, USA), pp. 67:1–67:6, ACM, 2008.

[34] M. Boucadair, R. Penno, and D. Wing, "Universal Plug and Play (UPnP) Internet Gateway Device - Port Control Protocol Interworking Function (IGD-PCP IWF)." RFC 6970 (Proposed Standard), July 2013.

[35] S. Cheshire and M. Krochmal, "NAT Port Mapping Protocol (NAT-PMP)." RFC 6886, Apr. 2013.

[36] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)." RFC 5389 (Proposed Standard), Oct. 2008.

[37] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture." RFC 4291, Feb. 2006.

[38] J. Postel, "Internet Protocol." RFC 791, Sept. 1981.

[39] J. Postel, "User Datagram Protocol." RFC 768, Aug. 1980.

[40] J. Weil, V. Kuarsingh, C. Donley, C. Liljenstolpe, and M. Azinger, "IANA-Reserved IPv4 Prefix for Shared Address Space." RFC 6598, Apr. 2012.

[41] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets." RFC 1918 (Best Current Practice), Feb. 1996.

[42] R. Hinden, M. O'Dell, and S. Deering, "An IPv6 Aggregatable Global Unicast Address Format." RFC 2374 (Historic), July 1998. Obsoleted by RFC 3587.

[43] A. Dhamdhere, M. Luckie, B. Huffaker, k. claffy, A. Elmokashfi, and E. Aben, "Measuring the Deployment of IPv6: Topology, Routing and Performance," IMC '12, (New York, NY, USA), pp. 537–550, ACM, 2012.

[44] M. Nikkhah, R. Guérin, Y. Lee, and R. Woundy, "Assessing IPv6 Through Web Access a Measurement Study and Its Findings," CoNEXT '11, pp. 26:1–26:12, ACM, 2011.

[45] M. Grégr, T. Podermański, and M. Švéda, "Measuring quality and penetration of ipv6 services," in *ICNS '14*, pp. 96–101, Institute for Systems and Technologies of Information, Control and Communication, 2014.

[46] C. Huitema, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)." RFC 4380 (Proposed Standard), Feb. 2006.

[47] F. Templin, T. Gleeson, and D. Thaler, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)." RFC 5214 (Informational), Mar. 2008.

[48] B. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds." RFC 3056 (Proposed Standard), Feb. 2001.

[49] D. Wing and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts." RFC 6555, Apr. 2012.

[50] M. Bagnulo, A. Sullivan, P. Matthews, and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers." RFC 6147, Apr. 2011.

[51] M. Bagnulo, P. Matthews, and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers." RFC 6146, Apr. 2011.

[52] G. Chen, Z. Cao, C. Xie, and D. Binet, "NAT64 Operational Experience." (Internet-Draft), 2014. Draft draft-ietf-v6ops-nat64-experience, no. 08.

[53] M. Mawatari, M. Kawashima, and C. Byrne, "464XLAT: Combination of Stateful and Stateless Translation." RFC 6877, Apr. 2013.

[54] A. Durand, R. Droms, J. Woodyatt, and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion." RFC 6333 (Proposed Standard), Aug. 2011.

[55] E. Nordmark and M. Bagnulo, "Level 3 Multihoming Shim Protocol for IPv6." RFC 5533, June 2009.

[56] C. Perkins, D. Johnson, and J. Arkko, "Mobility Support in IPv6." RFC 6275 (Proposed Standard), July 2011.

[57] "ip45.org - Project Webpage." Online http://www.ip45.org/. Accessed May 02, 2014.