VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IP45: DESIGN SPECIFICATION

BRNO 2014

Contents

Chapter 1

The status of the document

The aim of the document is to provide a technical description of the IP45 protocol architecture and design. The IP45 is an experimental protocol developed as a part of the university research. This document does not contain the motivation for the design, discussion of features and evaluation of the results.

Comments or questions, please, send to Tomas Podermanski (tpoder@cis.vutbr.cz).

Chapter 2

The Design of IP45 Protocol

2.1 Entry Conditions

The starting-points of the design were defined to fulfil several key requirements:

- Preserve the backward compatibility with IPv4 protocol. The IP45 traffic can be delivered through existing IPv4 infrastructure and, where it is possible, even pass NAT devices.
- Preserve the compatibility with existing IPv6 socket operations for clients and IPv4, IPv6 socket operations for servers. An application able to use IPv6 sockets can start using IP45 protocol without any need for modifying, redesigning or rebuilding.
- Follow the Saltzers's End-to-End Argument, the network devices processing IP45 packets should be kept as simple as possible. The IP45 packet header must use the fixed header structure to be processed easily by specialized hardware like ASIC or FPGA chips.

2.2 Overview of the Design

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)*. 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 2.1. 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 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.

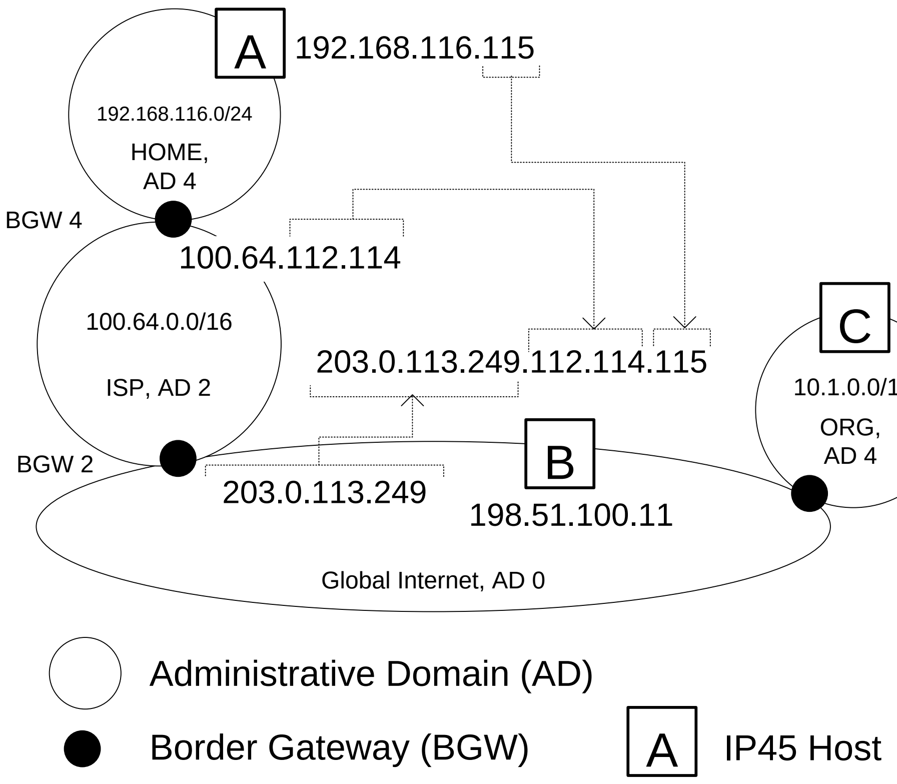

Figure 2.1: Structure of IP45 Networks

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 2.1, 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.

Chapter 3

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. *0.0.0.0.0.0.0.0.0.0.0.0.203.0.113.249.115*). 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 chapter 6), 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 [1]. This means that such addresses can be easily reused for addressing in IP45 without any conflicts with existing IPv6 addressing schemes[1].

The key benefit of reusing addresses originally dedicated for IPv6 is in keeping the compatibility with existing applications which support IPv6 sockets. The only problem that remains is in the notations of address in IP45 and IPv6 which are different. Although, the proper format is more or less a cosmetic issue not affecting the functionality of the protocol, it is useful when the *IP45 Host* is able to accept and return IP45 address in the proper format. This can be done by updating the system functions which are responsible for converting IP(v4,v6,45) into human-friendly notation and back (typically inet_ntop,

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

inet_pton).

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: $A4$ is IPv4 Address
Require: $M$ is Mark Value
Require: $S$ is IP45 Stack
Require: $A45$ is IP45 Address
 1: procedure Stck45ToIn45($A4, M, S$)
 2:     $A45[0..11 - M] \leftarrow 0$
 3:     $A45[15 - M..15] \leftarrow S[11 - M..11]$
 4:     $A45[11 - M..15 - M] \leftarrow A4$
 5:     return $A45$
 6: end procedure
```

Algorithm 3.1: Converting IPv4 Address, Mark and IP45 Stack into IP45 Address

```
Require: $A45$ is IP45 Address
Require: $A4$ is IPv4 Address
Require: $M$ is Mark Value
Require: $S$ is IP45 Stack
 1: procedure In45ToStck45($IP45$)
 2:     $M \leftarrow 0$
 3:     while $A45[M] = 0$ do
 4:         $M \leftarrow M + 1$
 5:     end while
 6:     $A4[0..3] \leftarrow IP45[M..(M + 3)]$
 7:     $M \leftarrow M + 4$
 8:     if $M < 12$ then
 9:         $S[(M - 4)..11] \leftarrow A45[M..15]$
10:     end if
11:     $M \leftarrow 16 - M$
12:     return $A4, M, S$
13: end procedure
```

Algorithm 3.2: 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 (3.2) converted into *IPv4 address*, *IP45 Stack* and *Mark*, and vice versa. The relationship between those fields is depicted in 3.1.

The algorithmic conversion from IPv4, IP45 Stack into IP45 Address is shown in 3.1 and in the reverse direction in 3.2. The role of *IP45 Stack*, *Mark* and *IPv4 Address* will be clarified in the following sections.

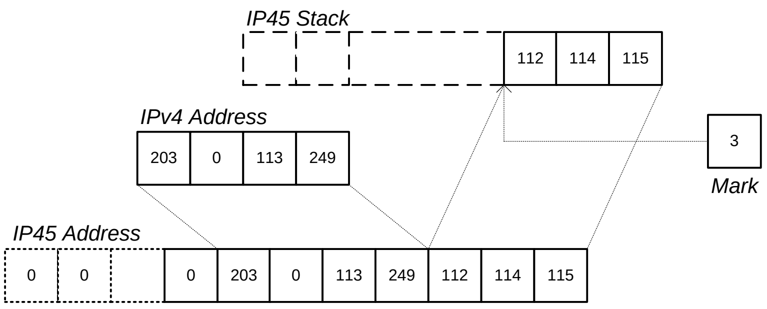

Figure 3.1: Relationship Between IP45 Stack, IPv4 address, Mark and IP45 Address

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 chapter 6.

Chapter 4

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 4.1.

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 [2]. Similarly, the fields of the second part of the header are equivalent to the definition of UDP header format described in RFC768 [3]. 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.

Version, IHL (4 bits): The fields indicate *Version* and *Header Length*. The value of version must be set to 4 for the *Version* and 5 for the *IHL*. The semantics of both fields is same as in IPv4 header defined in RFC791 [2]. In IPv4 packet, the value of the IHL 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[1]. 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 chapter 6.

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.

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

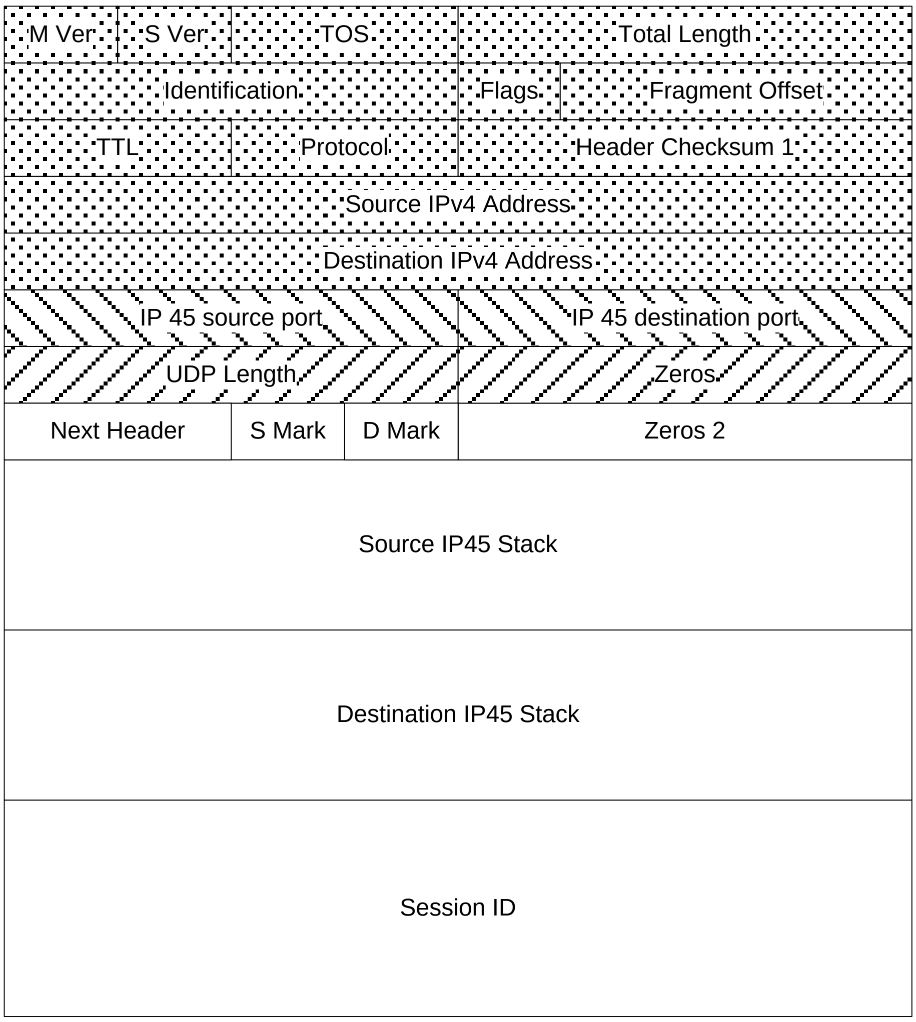

Figure 4.1: The IP45 Header Format

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. Chapters 5 and 6 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 chapter 5.

SID (128 bits): A unique randomly generated ID to identify the session between two *IP45 Hosts*. SID will be discussed in detail in chapter 6.

Chapter 5

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 5.1 shows the operation performed by *BGW* according to the model described in chapter **??**.

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 5.1 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.

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 *Down-*

```
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 $not\ CheckIP54Fields$ 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 5.1: Operations Performed on IP45 Border Gateway

stream 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 5.2.

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.

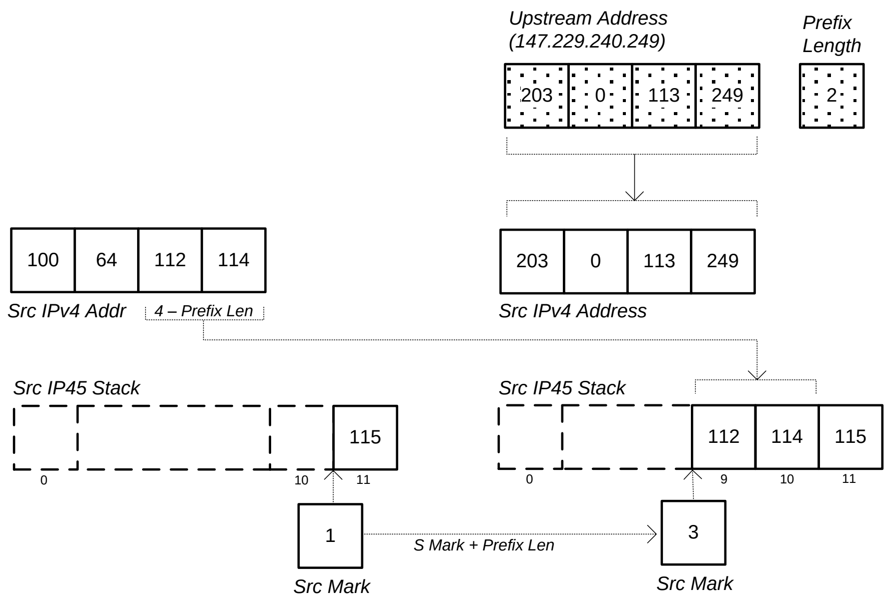

Figure 5.1: Modification of the Fields in *IP45 header* when a packet is received on *Downstream Prefix* interface

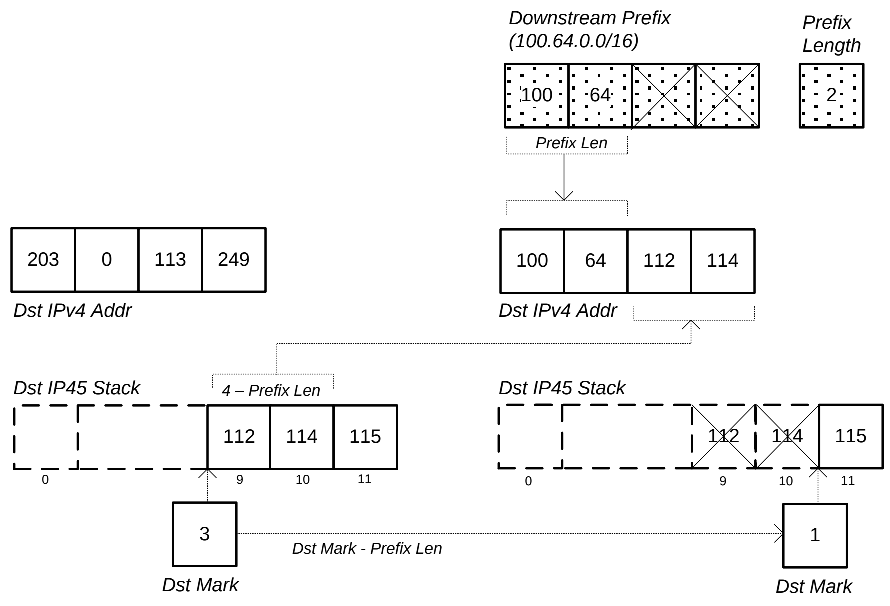

Figure 5.2: Modification of the Fields in *IP45 header* when a packet is received on *Upstream Address* interface

Chapter 6

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 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 3. 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.

The algorithm 6.1 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 chapter 3. 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.

As it is shown in algorithm 6.2, 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 6.3. After the entry check detects whether the packet is a

```
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 6.1: Initialization of IP45 Socket on IP45 Host

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*. The main advantage of this approach remains that the IP45 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 6.2 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.

```
Require: $S$ is Socket Structure
Require: $P$ is Packet Buffer
 1: procedure SendPkt
 2:     if $S.type\ is\ ip45$ then
 3:         $P.SessionID \leftarrow S.SessionID$
 4:         $P.DstIPv4Addr \leftarrow S.RemoteAddr$
 5:         $P.DstIP45Stack \leftarrow S.RemoteStack$
 6:         $P.DstIP45Mark \leftarrow S.RemoteMark$
 7:         $P.DstIP45Port \leftarrow S.RemotePort$
 8:         $P.SrcIP45Port \leftarrow 4$
 9:         $P.SrcMark \leftarrow 0$
10:         $P.Protocol \leftarrow UDP(16)$
11:         $P.Zeros2 \leftarrow 0$
12:         $SendIPv4Pkt(P, S)$
13:     else
14:         $SendIPv6Pkt(P, S)$
15:     end if
16: end procedure
```

Algorithm 6.2: Sending IP45 Packet

```
Require: $P$ is Received Packet
Require: $S$ is Socket Structure
Require: $PS$ is Protocol Specific Structure
 1: procedure RecvIP45Pkt(P)
 2:     if $not\ CheckIP54Fields$ then
 3:         $RecvIPv4Packet(P)$
 4:     end if
 5:     $PS \leftarrow GetProtocolSpecific(P.NextHeader, P)$
 6:     $S \leftarrow LookupIPv6(P.SessionID, PS)$
 7:     if $not\ S$ then
 8:         $S \leftarrow LookupIPv4(P.SessionID, PS)$
 9:     end if
10:     if $S.type\ is\ ip45$ then
11:         $S.RemoteAddr \leftarrow P.SrcIPv4Addr$
12:         $S.RemoteStack \leftarrow P.SrcIP45Stack$
13:         $S.RemoteMark \leftarrow P.SrcIP45Mark$
14:         $S.RemotePort \leftarrow P.SrcIP45Port$
15:         $HandleToUpperLayer(P)$
16:     else
17:         $ICMPErrorMessage()$
18:     end if
19: end procedure
```

Algorithm 6.3: Receiving IP45 Packet

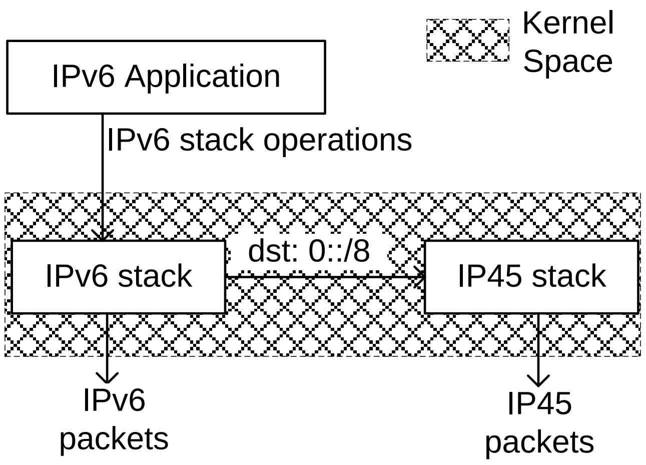

Figure 6.1: IP45 Data Flow in the Kernel

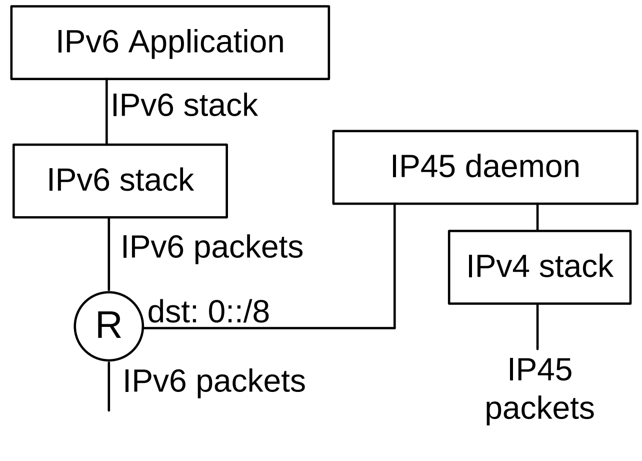

Figure 6.2: IP45 Data Flow Using IP45 Translator

Chapter 7

Addressing in Administrative Domains

Every *AD* is defined by networks sharing the same IPv4 network prefix defined as *Downstream Prefix (DP)*. 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.[1].

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 7.1 shows the division of existing address space in IP45.

Table 7.1: 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 [4]
3	reserved	-
4	home, organisation	Private, RFC 1918 [5]
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 [5]) 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

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

different *ADs* on a different level will use the size of *DP* according to their need. Returning back to figure 2.1 from chapter 2, 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.

Table 7.2: Number of devices within a single AD and max. number of ADs

Size of DP	devices in AD (max.)	num. levels
x.x.x.x/8	$2^{24} - 3 = 16{,}777{,}213$	$12 / 3 = 4$
x.x.x.x/16	$2^{16} - 3 = 65{,}532$	$12 / 2 = 6$
x.x.x.x/24	$2^{8} - 3 = 252$	$12 / 1 = 12$

Bibliography

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

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

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

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

[5] 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.