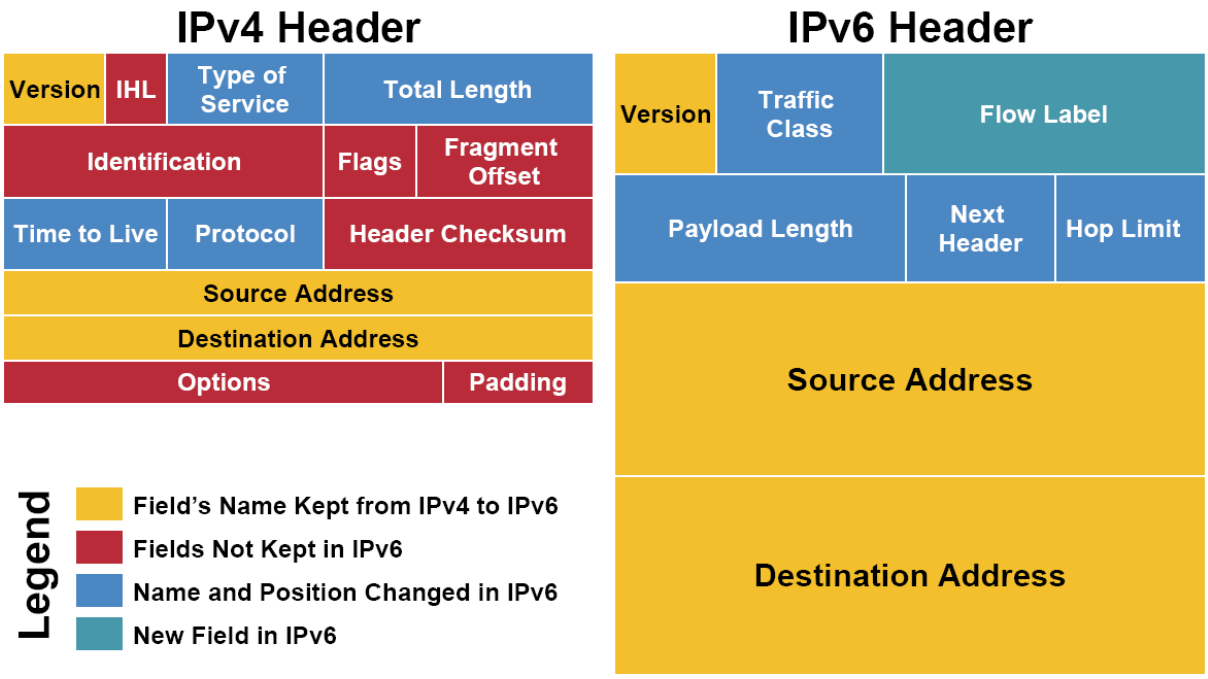
IPv6

IPv6 (Internet Protocol Version 6) je protokol mrežnog nivoa ISO-OSI modela, što znači da je njegov osnovni zadatak identifikacija uređaja u mreži i rutiranje paketa između tih uređaja. Razlog za uvođenje naslednika IPv4 protokola je taj što opseg 32-bitnih IPv4 adresa nije bio dovoljan veliki da pokrije sve uređaje čiji broj je ogromnom brzinom rastao sa ekspanzijom globalne svetske mreže - Interneta. Sam poduhvat zamene verzije Internet Protokola je izuzetno težak i dugotrajan jer zahteva da novu verziju sem novih uređaja podrži i celokupna postojeća mrežna infrastruktura. To je i glavni razlog što tranzicija između v4 i v6 verzija IP-a traje toliko dugo godina. Pored toga što IPv6 pruža 128 bita za identifikaciju adrese, on takođe sadrži neka nova polja u zaglavlju koja omogućavaju bolje performanse rutiranja paketa.

1. IPv6 paket

U nastavku je data struktura IPv4 i IPv6 zaglavlja kao i opis polja istih (Slika 1).



Slika 1. IPv4 i IPv6 zaglavlje

IPv4 zaglavlje bez opcionih okteta ima minimalnu dužinu od 20 bajta i sastoji iz sledećih polja:

- ***Version*** – Verzija IP protokola koji je generisao datagram.
- ***IHL (Internet Header Length)*** – dužina IP zaglavlja izražena u umnošcima 32-bitnih reči. Kako IP zaglavlje ima minimalno 20 bajta, najmanja vrednost ovog polja može biti 5 (20 bajta = 20 x 8 bita = 5 x 4 x 8 bita = 5 x 32-bitnih reči), dok sa sa opcionim poljima zaglavlje može dostići veličinu od 24 bajta.
- ***Type of service*** – Ukazuje na željeni kvalitet usluge, odnosno specificira željeni način tretiranja datagrama tokom njegovog prenosa kroz mrežu. Ovo polje je veličine 1 bajt, a postavljanjem određenih bita na vrednost 0/1 definiše se kako postupati sa paketom u smislu prioriteta slanja, kašnjenja, propusne moći mreže i pouzdanosti slanja.
  Podešavanje bita:
  - ***Biti 0-2 – prioritet***
    111 – Kontrola od strane mreže
    110 – Kontrola unutar mreže
    101 – CRITIC/ECP
    100 – Kratkotrajno opterećenje
    011 – Brzo
    010 – Direktno
    001 – Prioritet
    000 – Uobičajen postupak
  - ***Bit 3- kašnjenje***
    0 – Normalno kašnjenje
    1 – Malo kašnjenje
  - ***Bit 4 – propusna moć***
    0 – Normalna propusnost
    1 – Velika propusnost
  - ***Bit 5 – pouzdanost***
    0 – Normalna pouzdanost
    1 – Velika pouzdanost
  - ***Biti 6-7 – za buduću upotrebu***
- ***Total length –*** Dužina datagrama izražena u bajtima – uključuje IP zaglavlje i podatke.
- ***Identification*** – Identifikacioni broj koji je isti za sve fragmente koji potiču od istog datagrama. Pomoću njega je omogućeno prijemnoj strani da ponovo sastavi originalni datagram.
- ***Flags*** – 3 kontrolna bita od kojih se dva koriste u postupku fragmentacije.
  - Bit 0 je rezervisan i ima vrednost 0.
  - Bit 1 ima oznaku DF (Don't Fragment) i opisuje da li je fragmentacija dopuštena (0 – dopuštena fragmentacija, 1 – nije dopuštena).
  - Bit 2 ima oznaku MF (More Fragments) i opisuje status pristiglog fragmenta (0 – poslednji fragment, 1 – ima još fragmenata). Kada se koristi fragmenatcija datagrama, svi fragmenti osim zadnjeg imaju ovo polje postavljeno na vrednost 1. Kada nema fragmentacije, tada se šalje ceo datagram odjednom i ovo polje ima vrednost 0.
- ***Fragment offset*** – Ako je MF setovan, onda "Fragment offset" pokazuje poziciju tekućeg fragmenta unutar originalnog datagrama. Prvi fragment ima "offset" 0.
- ***Time to live*** – Određuje maksimalnu dužinu "opstanka" datagrama u mreži, u smislu broja rutera kroz koje će proći. Na izvornom računaru se postavlja na početnu vrednost. Svaki ruter duž putanje datagrama do odredišta umanjuje vrednost TTL

polja za 1. Ako u nekom ruteru vrednost TTL polja dostigne 0, taj datagram se odbacuje (smatra se da predugo kruži kroz mrežu), a izvorišni računar se obaveštava o tome.

- ***Protocol*** – Ovo polje sadrži oznaku protokola sledećeg sloja kome pripadaju podaci u polju "data". Primeri: 1 za ICMP protocol, 6 za TCP i 17 za UDP.
- ***Header checksum*** – Kontrolna suma IP zaglavlja.
- ***Source address*** – IP adresa izvorišnog računara.
- ***Destination address*** – IP adresa odredišnog računara.
- ***Options*** – Opciono polje datagrama.
- ***Padding*** – Opciono polje datagrama – dodaje se eventualno da bi se osiguralo da zaglavlje sadrži ceo broj (integer) 32-bitnih reči.
  Data – podaci koji se šalju.

IPv6 zaglavlje se sastoji od fiksnog dela (*Fixed Header*) sa minimalnom funkcionalnošću potrebnom za sve pakete i može biti praćeno mogućim dodatnim proširenjima (*Extension Header*) koja primenjuju specijalne funkcionalnosti. IPv6 adresa je 4 puta veća od IPv4 adrese, ali iznenađujuće, IPv6 zaglavlje je samo 2 puta veće od IPv4 zaglavlja. Sve potrebne informacije od suštinskog značaja za rutere se čuvaju u fiksnom delu IPv6 zaglavlja. Prošireno zaglavlje sadrži opcione informacije koje pomažu ruterima u procesu rukovanja sa paketima tj. mrežnim protokom. Fiksno zaglavlje zauzima prvih 40 okteta (320 bita) IPv6 paketa i ono se sastoji iz sledećih polja:

- ***Version –*** Ovo polje je veličine 4 bita, i identifikuje verziju IP protokola. Za IPv6 u ovom polju piše 6. (Napomena, ako u ovom polju unesemo broj 4, to ne znači da smo napravili validan IPv4 datagram).
- ***Traffic class –*** Ima 8 bita i slično je polju TOS u IPv4. Može da služi da se odredi prioritet paketu.
- ***Flow label –*** Ovo polje ima 20 bita i koristi se za identifikaciju toka datagrama. IPv6 nudi mogućnost da se označi kojim će tokom paket da putuje. To se koristi u nekim posebnim slučajevima, kada pošiljalac zahteva specijalno rukovanje sa njegovim paketom, ili neko zakupi određeni tok zbog bolje usluge. Npr. video i audio prenos mogu da se tretiraju kao tok (eng. *flow*), dok email uglavnom ne.
- ***Payload length –*** Ova vrednost je 16 bita i tretira se kao unsigned integer. Označava broj bajta u IPv6 datagramu koji slede posle fiksne dužine zaglavlja od 40 bajta, jer IPv6 nudi i neka opciona polja.
- ***Next header –*** Indetifikuje protokol transportnog sloja koji se koristi za dostavu podataka, npr. TCP ili UDP. Ima isto značenje kao i polje "protocol" u IPv4 zaglavlju.
- ***Hop limit –*** Pošiljalac pri slanju odredi posle koliko skokova (paket kada putuje kroz mrežu ide od rutera do rutera, skok podrazumeva prelazak paketa iz jednog rutera u drugi) će njegov paket biti uništen, tj. odbačen. U svakom ruteru kroz koji paket prođe, polje Hop limit se umanjuje za jedan. Kada ovo polje dobije vrednost nula, paket se odbacuje, odnosno ne prosleđuje se sledećem ruteru. Ovo se radi zato da zalutali paketi ne bi opterećivali mrežu.
- ***Source address –*** 128-bitni zapis adrese pošiljaoca.
- ***Destination address –*** 128-bitni zapis adrese primaoca.

  Data – Kada IPv6 datagram dostigne svoje odredište, ovaj deo se prosleđuje protokolu transportnog sloja koji je naznačen u Next header polju (npr. TCP, UDP...).

2. IPv6 adresiranje

Koliko je proširenje adresnog prostora postignuto uvođenjem IPv6?

- IPv4 pokriva: 4,294,967,296 jedinstvenih adresa, dok
- IPv6 340,282,366,920,938,463,463,374,607,431,768,211,456 adresa!!!

Značenje ove poslednje cifre se može slikovito predstaviti na sledeći način: uvođenjem IPv6 adresa za svaki kvadratni metar zemaljske kugle dobili smo 1500 jedinstvenih adresa što je više nego dovoljno čak i u slučaju kada tosterima, peglama, i sličnim uređajima budu potrebne IP adrese.

Pošto IPv6 adresa ima 128 bita, decimalni način zapisa adresa kao kod IPv4 (primer: "128.89.31.226'') nije bio moguć iz praktičnih razloga. IPv6 adresa se zapisuje kao niz heksadecimalnih cifri grupisanih u 8 delova odvojenih dvotačkom. Svaki deo zapisa ima po 4 heksadecimalne cifre koje označavaju 16 bita adrese.

Primer: 47CD:1234:4422:ACO2:0022:1234:A456:0124.

Jedan ili više uzastopnih delova u zapisu IPv6 adrese sačinjenih samo od 0 se mogu izostaviti iz zapisa i zameniti sa dva uzastopna znaka ":". Ovakva zamena u adresi sme se izvršiti samo na jednom mestu.

(BA02:0:0:0:0:0:0:1) = (BA02::1)

Slika 2. Poređenje IPv4 i IPv6 sa aspekta adresiranja

3. Vrste IPv6 adresiranja

1. **Unicast adresiranje** – Unicast adresa identifikuje 1 mrežni interfejs. Dostava paketa je dizajnirana za host-to-host (1 na 1) komunikaciju. Unicast paketi su usmereni i dostavljeni od jedne tačke do samo jedne tačke u mreži.

   Primer: 3731:54:65fe:2::a7

2. **Multicast adresiranje -** Multicast adresa je dodeljena grupi od više tačaka/čvorova u mreži kojima se šalje paket. IPv6 multicast je po funkciji sličan IPv4 broadcast-u i opisan je kao "jedan prema mnogima", ali ne kao "jedan prema svima" IP komunikacija. Zato što IPv6 ne implementira broadcast adrese, multicast adresa 'svi čvorovi' se koristi da se napravi multicast grupa svih čvorova u lokalnoj mreži. U poređenju sa IPv4 broadcast-om, IPv6 multicast je efikasnije sredstvo u grupnoj komunikaciji. Više nije potrebno da svi primaju istu broadcast poruku, i tako se čuva dragoceno procesorsko vreme i smanjuje saobraćaj u mreži.

   Primer: FF01:0:0:0:0:0:0:1

3. **Anycast adresiranje -** Anycast adresa je adresa pridružena grupi interfejsa kao kod multicast-a, ali razlika je u tome što u Anycast komunikaciji će samo jedan od tih čvorova primiti paket. Paket koji se šalje na anycast adresu se isporučuje samo jednom (najčešće najbližem ili najboljem) odredištu. Anycast adresiranje koristi istu sintaksu kao unicast, i primalac ne može primetiti razliku između ova dva načina slanja.

   Primer: 3731:54:65fe:2::a8

Navešćemo neke od primera specifičnih IPv6 adresa:

1. **Loopback adresa** - koristi se za testiranje transportne mreže. Ova adresa se znači da je odredište paketa identično izvorištu paketa (tj. na datom lokalnom računaru).

   (0:0:0:0:0:0:0:1) ili (::1)

2. **Link-local adrese** (FE80::/10) - Adrese za slanje unutar lokalne mreže. Prva grupa bitova počinje sa FE u heksadecimalnom prikazu privatnih adresa. Ove adrese se koriste kao i privatne adrese u IPv4 (169.254.0.0/16.) i ne mogu biti rutirane. Paketi koji su poslati ovakvoj privatnoj adresi ostaju u lokalnoj mreži i ne idu preko rutera van organizacije, tj. te lokalne mreže.

3. **Multicast adrese** (FF00::/8) **-** Prva grupa bitova (1111 1111) počinje sa FF u heksadecimalnom prikazu označava multicast adresiranje.

   (FF01:0:0:0:0:0:0:1) - Rezervisana multicast adresa svih čvorova;
   (FF01:0:0:0:0:0:0:2) - Rezervisana multicast adresa svih rutera;

4. Tranzicija između IPv4 i IPv6

IPv6 tako unapređen i poboljšan je imao još samo jednu prepreku. Kako prebaciti internet sa IPv4 na IPv6. Čak i u to vreme internet je bio ogroman i decentralizovan i bilo je nemoguće celu globalnu mrežu ugasiti makar na jedan dan, i nadograditi sve uređaje na IPv6. Razumno rešenje je bilo da se to uradi postepeno i da uređaji mogu i dalje komunicirati preko IPv4.

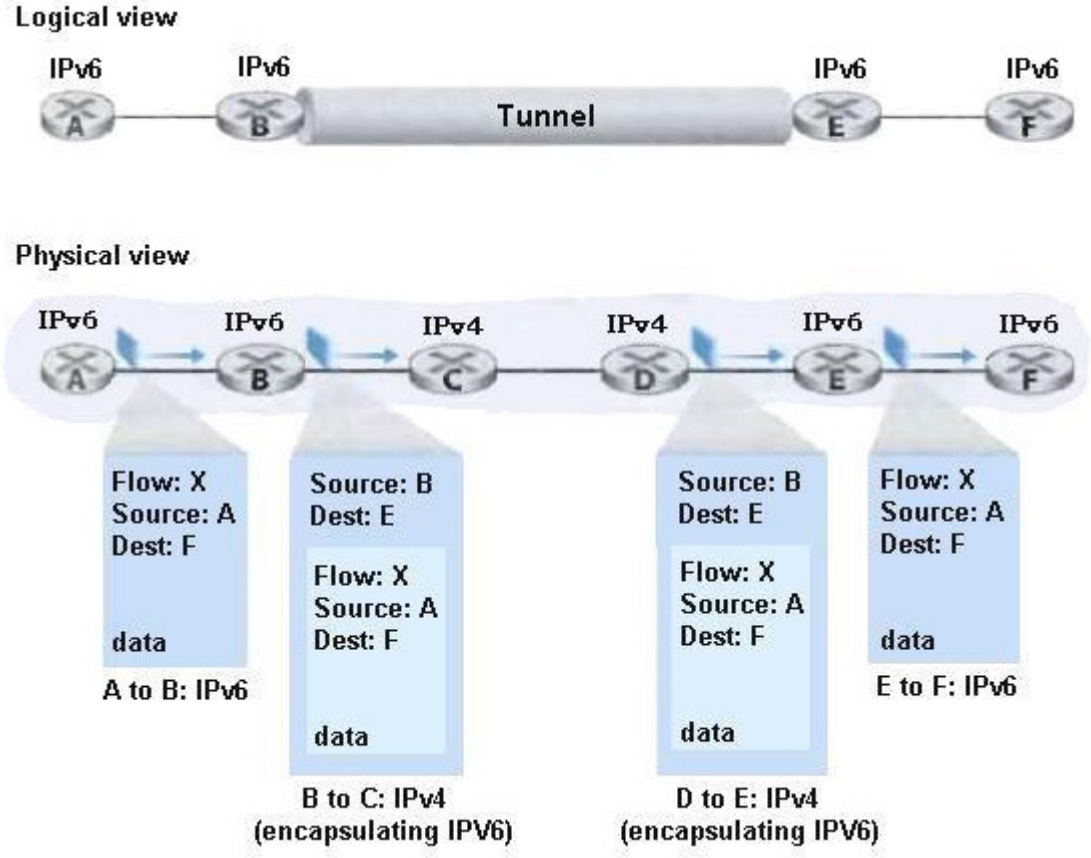
Usledila su sledeća rešenja, *dual-stack operation* i *tunneling*.

1) **Dual-stack operation**:

   Uređaj radi i sa IPv4 i IPv6 i koristi polje *Version* da bi znao kako da procesira paket.

2) **Tunneling**:

   Tunelovanje se koristi da se IPv6 datagram pošalje preko dela mreže koji razume samo IPv4. To se radi tako što se IPv6 datagram enkapsulira u IPv4 datagram čije zagljavlje sadrži adresu čvora na kraju tunela. To znači da u slučaju tunelovanja ceo IPv6 datagram predstavlja podatke (*Data)* u IPv4 datagramu dok ne stigne na kraj 'tunela' gde drugi čvor otpakuje i prosledi ga dalje kao IPv6 datagram.



Slika 3. Tunelovanje IPv6 datagrama

5. WinSock podrška korišćenju IPv6 protokola

Zbog različitosti IPv4 i IPv6 adresa neophodna je nova struktura za zapis IPv6 adresa, kao i funkcije za konverziju iz standardnog (heksadecimalnog) zapisa u binarni, i obrnutno. Rukovanje sa portovima i utičnicima ostaje bez promena, osim što se i pri kreiranju soketa prosledi odgovarajuća vrednost koja označava o kojoj adresnoj familiji se radi, tj. AF_INET6 za IPv6.

```
// Create IPv6 socket
SOCKET serverSocket = socket(AF_INET6, // IPv6 address family
       SOCK_DGRAM,                     // Datagram socket
       IPPROTO_UDP);                   // UDP
```

Konverzija između stringa i IPv6 adrese

Kao i kod IPv4 potrebne su nam funkcije koje će adresu iz stringa pretvoriti u binarni zapis i obrnuto. Funkcije *inet_addr i inet_ntoa* ne mogu raditi sa IPv6 adresama. Koristićemo funkciju *inet_pton* za konverziju adrese iz stringa u binarni zapis, i funkciju *inet_ntop* za prevođenje binarnog zapisa u standardni heksadecimalni zapis IPv6. Treba naglasiti da ove dve funkcije mogu vršiti i konverziju IPv4 adresa zato što im se kao ulazni parametar prosleđuje podatak o kojoj adresnoj familiji se radi. Simboličke konstante za oznaku adresnih familija su AF_INET6 za IPv6, i AF_INET za IPv4.

Deklaracija *inet_pton* funkcije, njen detaljan opis i primer su dati u nastavku:

```
int inet_pton( int af, const char *buffer, void *addr );
```

Funkcija:	Opis:
inet_pton	Pretvaranje IP adrese iz standardnog heksadecimalnog zapisa u binarni zapis.
Parametri:	**Opis:**
int af[in]	Funkcija može da radi sa IPv6 i IPv4, u ovom parametru prosleđujemo sa kojom verzijom radimo. AF_INET6 za IPv6.
const char* buffer [in]	Pokazivač na bufer koji sadrži IP adresu u standardnoj notaciji. U slučaju IPv6 adresa je u standarnoj heksadecimalnoj notaciji, zapisana kao niz znakova (karaktera) koji se završava sa '\0'. Moguće je proslediti i samo string, ili makro.
void* addr [out]	Pokazivač na bafer u kome će biti smešten rezultat, tj. binarno zapisana IPv6/IPv4 adresa u mrežnom redosledu bajtova.
Povratna vrednost:	**Opis:**
int	Vraća 1 ako se funkcija uspešno izvrši. Vraća 0 ako prosleđen string nije standardni zapis IPv6 ili IPv4 adrese. Vraća -1 ako se desila greška. Pozvati funkciju *WSAGetLastError()* za detaljniji opis.

Deklaracija *inet_ntop* funkcije, njen detaljan opis i primer su dati u nastavku:

```
const char *inet_ntop(int af, const void *src, char *addr, int length);
```

Funkcija:	Opis:
inet_ntop	Pretvaranje IP adrese iz binarnog u standardni heksadecimalni zapis.
Parametri:	Opis:
int af[in]	Funkcija može da radi sa IPv6 i IPv4, u ovom parametru prosleđujemo sa kojom verzijom radimo. AF_INET6 za IPv6 i AF_INET za IPv4
const void* src [in]	Pokazivač na IP adresu u binarnom zapisu (u slučaju IPv6 128 bita),pri čemu zapis ima mrežni redosled bajtova.
char* addr [out]	Pokazivač na bafer u koji će se smestiti adresa zapisana u obliku stringa. Napomena: za IPv6 bafer treba da bude dovoljno velik da smesti 46 karaktera. Za veličinu bafera može se iskoristiti konstanta INET6_ADDRSTRLEN.
int length [in]	Veličina bafera na koji pokazuje *addr*.
Povratna vrednost:	Opis:
const char *	Ako se uspešno izvrši, vraća nenulti pokazivač na bafer addr. Ako se desi greška vraća NULL. Pozvati funkciju *WSAGetLastError()* za detaljniji opis.

Primer:

```
/*-------------------------------------------------------------------------------*/

// Server address structure
sockaddr_in6 serverAddress;

// Enough big buffer for IPv6 address
char ipAddress[65];

// Store IP in socketAddress
inet_pton(AF_INET6, "fe80::5507:bce1:e6a:7195", &serverAddress.sin6_addr);

// Return the IP
inet_ntop(AF_INET6,&serverAddress.sin6_addr, ipAddress, sizeof(ipAddress));

// Show IPv6 address on the screen. Prints "fe80::5507:bce1:e6a:7195"
printf("%s\n", ipAddress);

/*-------------------------------------------------------------------------------*/
```

Zapisivanje IPv6 adrese

Za zadavanje IPv6 adresa koristi se *sockaddr_in6* struktura.

```
struct sockaddr_in6 {
        short sin6_family; // AF_INET6
        unsigned short sin6_port; // Transport level port number
        unsigned long sin6_flowinfo; // IPv6 flow information
        struct in6_addr sin6_addr; // IPv6 address
        unsigned long sin6_scope_id; // Scope of the IPv6 address
}

struct in6_addr {
        union {
                unsigned char Byte[16];
                unsigned short Word[8];
        } u;
}
```

Unija unutar *in6_addr* strukture sadrži dve različite reprezentacije IPv6 adrese: kao niz od 16 unsigned char vrednosti ili kao niz od 8 unsigned short vrednosti.

Sledi opis pojedinih polja *sockaddr_in6* strukture:

sin6_family:
: Ovo polje određuje adresnu familiju. Postavlja se na vrednost AF_INET6 što označava da Winsock koristi IPv6 adresnu familiju. Kod IPv4 adresne familije koristila se vrednost AF_INET.

sin6_port:
: Ovo polje se ne razlikuje od onog u IPv4 strukturi. Definiše port koji će TCP ili UDP koristiti da identifikuje protokol aplikativnog sloja. Aplikacije prilikom biranja porta moraju da znaju da je skup vrednosti od 0-1023 skup rezervisanih portova za postojeće protokole.

sin6_flowinfo:
: Sadrži informacije o IPv6 toku. Tok (eng. flow) označava putanju kojom će paket da se kreće kroz mrežu. Koristi se ako neko želi posebno rukovanje sa njegovim paketom, ili je npr. platio za bolje usluge.

sin6_addr:
: Ovo polje predstavlja IPv6 adresu (128 bita) u binarnom zapisu. Funkcija *inet_pton* pretvara string u binarni zapis (u *network-byte* redosledu) i direktno smešta u ovo polje.

sin6_scope_id:
: Ovo polje označava doseg IPv6 adrese. Neki od primera dosega su: doseg na nivou lokalnog interfejsa, na nivou lokalnog linka, na nivou podmreže, globalni doseg.

Sledi primer kako da *sockaddr_in6* strukturu popunimo sa određenom IPv6 adresom ("fe80::5507:bce1:e6a:7195") i brojem porta (5150).

Primer:

```
/*------------------------------------------------------------------------------*/

sockaddr_in6 socketAddress;
int nPortId = 5150;
// Za pocetak cemo inicijalizovati sva polja structure na 0
memset((char*)&socketAddress, 0, sizeof(socketAddress));

// Koristimo IPv6 adresnu familiju
socketAddress.sin6_family = AF_INET6;

// Pomocu funkcije inet_pton konvertujemo datu IPv6 adresu iz standardne
// heksadecimalne notacije u 128 bita i dodelimo tu vrednost polju sin6_addr.
inet_pton(AF_INET6, "fe80::5507:bce1:e6a:7195", &socketAddress.sin6_addr);

// Promenljiva nPortId je zapisana u host-byte redosledu.
// Pomocu funkcije htons zapisujemo nPortId u network-byte redosledu, i smestamo tu
vrednost polju sin_port.
socketAddress.sin6_port = htons(nPortId);

// Ako ne zelimo posebno rukovanje, tj. poseban tok za nas paket polju
// sin6_flowinfo dodelimo vrednost nula.
socketAddress.sin6_flowinfo = 0;

/*------------------------------------------------------------------------------*/
```

Poput i slučaja sa IPv4 adresama, moguće je koristititi opciju da se koriste sve dostupne logičke adrese uređaja, ako se za vrednost adrese navede.

```
socketAddress.sin6_addr = in6addr_any;
```

6. Realizacija Dual Stack UDP servera

Navešćemo dve mogućnosti za realizaciju dual stack UDP servera odnosno servera koji istovremeno može da prima poruke generisane i preko IPv4 i preko IPv6 protokola.

I.

Na serveru se kreiraju dve utičnice, jedna za IPv4, a druga za IPv6 adresnu familiju. Zatim se popune dve adresne strukture sa podacima server: jedna tipa *sockaddr_in* za IPv4, a druga *sockaddr_in6* za IPv6. Povezivanje soketa sa adresama obavljamo tako što bind-ujemo IPv4 soket sa IPv4 adresom, a IPv6 soket sa IPv6 adresom. Potrebno je još oba soketa staviti u neblokirajući režim, i koristiti neki model neblokiranja (pooling ili select) da bi server mogao da prima poruke istovremeno sa obe vrste klijenata.

II.

Serverska utičnica koja je kreirana za IPv6 adresnu familiju po osnovnom (podrazumevanom) ponašanju šalje i prima isključivo IPv6 pakete. Ali, za takvu utičnicu možemo dodatno postaviti opciju tako da šalje i prima i IPv4 i IPv6 pakete.

Ova dodatna opcija se postavlja pomoću funkcije *setsockopt()*, čiji opis i primer upotrebe je dat u nastavku.

```
int setsockopt(SOCKET s, int level, int optname, char *optval, int len);
```

Funkcija:	Opis:
setsockopt	Koristi se za postavljanje/menjanje opcija utičnice.
Parametri:	**Opis:**
SOCKET s [in]	Utičnica za koju se menjaju opcije.
int level [in]	„Nivo“ na kom je opcija definisana
int optname [out]	Ime opcije čija vrednost će se postavljati
char *optval [in]	Pokazivač na bafer u kome se nalazi vrednost za izabranu opciju
int len [in	Veličina bafera (u bajtima) na koji pokazuje optval
Povratna vrednost:	**Opis:**
int	Ako se uspešno izvrši, vraća 0. Ako se desi greška vraća SOCKET_ERROR i u tom slučaju potrebno je pozvati *WSAGetLastError()* radi detalja.

Primer poziva:

```
/*-------------------------------------------------------------------------------*/

// Omoguciti da soket da moze da prima i IPv4 i IPv6 pakete tako sto opciju
// IPV6_V6ONLY postavimo na vrednost 0
char optval = 0;

int iResult;
iResult=setsockopt(serverSocket, IPPROTO_IPV6, IPV6_V6ONLY, &optval, sizeof(optval));

/*-------------------------------------------------------------------------------*/
```

Napomena: Poziv funkcije *setsockopt()* radi setovanja polja IPv6_V6ONLY na 0 treba uraditi pre poziva funkcije *bind()* za datu serversku utičnicu.

Kada IPv6 serverskoj utičnici omogućimo da prima i IPv4 pakete, tada će se automatski IPv4 adresa mapirati na odgovorajuću IPv6 adresu. Server nije svestan da radi sa IPv4 klijentom. Postoje rezervisane IPv6 adrese za mapiranje IPv4 i one u prvih 80 bita sadrže sve nule, sledećih 16 bita sve jedinice i onda sledi sama IPv4 adresa od 32 bita. Dakle, adresa IPv4 klijenta koju server vidi je u obliku IPv6 adrese.

80 bits	16	32 bits
0000.....0000	FFFF	IPv4 address

Primer: ::FFFF:129.144.52.38

Dakle, nakon što server primi poruku od IPv4 klijenta, server njegovu adresu vidi kao IPv6 adresu koju bi ispisao u obliku kao u gornjem primeru. U slučaju da želimo da ispišemo tu adresu u vidu standarnog ispisa za IPv4, moramo pristupiti delu IPv6 adrese koji sadrži klijentovu IPv4 adresu, odnosno potrebna su nam samo poslednja 4 bajta adrese. A da bismo uopšte saznali da li je IPv6 adresa u stvari mapirana IPv4 adresa, potrebno je proveriti prvih 10 bajtova da li su nule i sledeća dva bajta da li su -1 (sve jedinice). U nastavku je dat primer koda koji proverava da li je adresa klijenta mapirana IPv4 adresa ili stvarna IPv6 adresa.

Primer:

```
/*-------------------------------------------------------------------------------*/

// INET6_ADDRSTRLEN 65 spaces for hexadecimal notation of IPv6
char ipAddress[INET6_ADDRSTRLEN];

// Copy client ip to local char[]
inet_ntop(clientAddress.sin6_family, &clientAddress.sin6_addr, ipAddress,
sizeof(ipAddress));

// Convert port number from network byte order to host byte order
unsigned short clientPort = ntohs(clientAddress.sin6_port);
bool isIPv4 = is_ipV4_address(clientAddress); //true for IPv4 and false for IPv6

if(isIPv4){
	// 15 spaces for decimal notation (for example: "192.168.100.200") + '\0'
	char ipAddress1[15];
	struct in_addr *ipv4 = (struct  in_addr*)&((char*)&clientAddress.sin6_addr.u)[12];

	// Copy client ip to local char[]
	strcpy_s(ipAddress1, sizeof(ipAddress1), inet_ntoa( *ipv4 ));
	printf("IPv4 Client connected from ip: %s, port: %d, sent: %s.\n",
		ipAddress1, clientPort, dataBuffer);
}
else
{
	printf("IPv6 Client connected from ip: %s, port: %d, sent: %s.\n",
	ipAddress, clientPort, dataBuffer);
}

/*-------------------------------------------------------------------------------*/
```

U nastavku je dat primer implementacije funkcije koja proverava da li adresa pripada IPv4 familiji protokola:

Primer:

```
/*-------------------------------------------------------------------------------*/

bool is_ipV4_address(sockaddr_in6 address)
{
    char *check = (char*)&address.sin6_addr.u;

    for (int i = 0; i < 10; i++)
        if(check[i] != 0)
            return false;

    if(check[10] != -1 || check[11] != -1)
        return false;

    return true;
}

/*-------------------------------------------------------------------------------*/
```

ZADATAK

Koristeći primer implementacije UDP klijenta i UDP servera koji je dat u prilogu materijala za vežbe, potrebno je realizovati dual stack UDP server koji će moći da prima poruke generisane i preko IPv4 i preko IPv6 protokola.

1. Na serveru napraviti jedan UDP soket zasnovan na IPv4 adresnoj familiji i jedan UDP soket zasnovan na IPv6 adresnoj familiji.

2. Kreirati dve promenljive, prva tipa *sockaddr_in*, a druga tipa *sockaddr_in6* strukture i popuniti ih sa adresnim podacima datim u nastavku:
   - serverAddress1
     - IP adresa: 127.0.0.1
     - Port: 27015
   - serverAddress2
     - IP adresa: sve dostupne IPv6 adrese (simbolička konstanta: *in6addr_any*)
     - Port: 27015

3. Povezati obe kreirane utičnice sa odgovarajućim adresnim podacima i staviti ih u neblokirajući režim.

4. Implementirati *polling* model neblokiranja na serveru, koji omogućava da se prilikom prijema poruka na dve serverske utičnice (IPv4 i IPv6 varijanta) ne blokira nit procesora.
   - Ako je poruka uspešno primljena, ispisati je na ekranu, kao i adresu i port klijenta koji je poslao poruku.
   - Ako se desi greška prilikom prijema, zatvoriti utičnice i ugasiti server.

5. Na osnovu implementacije UDP klijenta zasnovanog na IPv4 protokolu, kreirati još jedan projekat “Klijent2”, u kome:
   - Umesto postojeće UDP utičnice zasnovane na IPv4, treba kreirati klijentsku UDP utičnicu zasnovanu na IPv6 adresnoj familiji.
   - Kreirati promenljivu tipa *sockaddr_in6* strukture i popuniti je sa sledećim podacima o serveru kome će klijent slati poruke:
     - IP adresa servera: "0:0:0:0:0:0:0:1"
     - Port servera: 27015

6. Posmatrati razmenu paketa između klijenata i servera u Wireshark-u – analiza i tumačenje mrežnog saobraćaja.

Napomena: Slanje više poruka sa klijenta je implementirano u priloženom, osnovnom kodu.