IPv4 vs IPv6

Internet Protocol

IP (nazwane później IPv4):

- znormalizowane przez **RFC760** (wrzesień 1981)
- zaktualizowane przez **RFC791** (styczeń 1980)
- rozmiar adresu: 4 oktety = 32 bity
- teoretyczna przestrzeń adresowa: 2^{32} = 4.294.967.296

Internet Protocol

- definiuje użycie wybranych adresów jako **prywatnych** → ok. 18 milionów adresów
- wprowadza adresowanie **multicast** (tzn. adresowanie wybierające więcej niż jednego odbiorcę) → ok. 270 milionów adresów

Prezentacja i reprezentacja

- fizycznie adres IP jest ciągem **32 bitów** przechowywanych i transmitowanych w konwencji **MSB** (bardziej znaczące bity przechowywane są na młodszych adresach)
- według RFC adres taki może być prezentowany na wiele sposobów, jednak nie wszystkie aplikacje honorują to ustalenie

Zapis i reprezentacja

notacja	zapis
dotted-decimal	192.0.2.235
dotted-hexadecimal	0xC0.0x00.0x02.0xEB
dotted-octal	0301.0250.0002.0353
hexadecimal	0xC00002EB
decimal	3221226219
octal	030000001353

Maska

- 32 bity adresu IP są logicznie dzielone na **adres sieci** i **adres** i **identyfikator hosta –** podział ten ułatwia routowanie
- wg starszej notacji (używanej tylko dla Ipv4) podział ten definiowany jest poprzez tzw. maskę zapisywaną zwyczajowo jako dotted-decimal
- bity maski ustawione na **1** wydzielają **adres sieci**, ustawione na **0 –** identyfikator **hosta**
- np. w adresie 192.168.2.4 z maską 255.255.255.0:
  - 192.168.2.0 jest adresem sieci
  - 4 jest identyfikatorem hosta

Maska

- w nowszej notacji (RFC1519, sierpień 1993) mającej zastosowanie także w Ipv6, a nazywanej CIDR (od Classless Inter-Domain Routing), maskę zapisuje się jako wartość określającą liczbę bitów adresu sieci, np:

  192.168.2.4/24

Adresy o specjalnym znaczeniu

- **10.0.0.0/8**
- znaczenie: sieć prywatna (16.777.216 adresów)
- znane od: RFC1918 (luty 1996)

Adresy o specjalnym znaczeniu

- **127.0.0.0/8**
- znaczenie: loopback
- znane od: RFC1122 (październik 1989)

Adresy o specjalnym znaczeniu

- **169.254.0.0/16**
- znaczenie: link-local (izolowana sieć lokalna)
- znane od: RFC3927 (maj 2005)

Adresy o specjalnym znaczeniu

- **172.16.0.0/12**
- znaczenie: sieć prywatna (1.048.576 adresów)
- znane od: RFC1918 (luty 1996)

Adresy o specjalnym znaczeniu

- **172.16.0.0/12**
- znaczenie: routery ip6/ip4
- znane od: RFC3068 (czerwiec 2001)

Adresy o specjalnym znaczeniu

- **192.168.0.0/16**
- znaczenie: sieć prywatna (65.536 adresów)
- znane od: RFC1918 (luty 1996)

Adresy o specjalnym znaczeniu

- **224.0.0.0/4**
- znaczenie: IP multicast
- znane od: RFC5771 (marzec 2010)

Adresy o specjalnym znaczeniu

- **255.255.255.255**
- znaczenie: adres rozgłoszeniowy
- znane od: RFC919 (październik 1984)

Wyczerpywanie adresacji IP

- od początku lat 80-tych trwa nieprzerwana walka z głodem adresów IPv4, spowodowana rozrostem sieci na skalę, która nie była brana pod uwagę w chwili tworzenia założeń protokołu IP
- metody walki:
  - NAT
  - sieci prywatne
  - DHCP
  - hosting wirtualny
  - zacieśnienie współpracy między rejestratorami adresów IP

IPv6

IPv6

- zaproponowane przez **RFC1883** (grudzień 1995)
- zaktualizowane przez **RFC2460** (grudzień 1998)
- w chwili obecnej mniej niż 4% hostów w Internecie używa IPv6
- Google twierdzi, że w lutym 2014 ok. 3% jego klientów przybywało z adresów IPv6

IPv6

- adres IPv6 zajmuje **16 oktetów → 128 bitów**
- daje to teoretyczną pojemność adresacji równą $\mathbf{2^{128}} \rightarrow 3{,}4 * 10^{38}$ ...
- ...czyli ok. $7{,}9 * 10^{28}$ raza większą od pojemności Ipv4
- z założenia IPv4 i Ipv6 nie są interoperatywne – w jedej sieci może być stosowane wyłącznie jedno adresowanie

Prezentacja i reprezentacja

- pełny adres IPv6 zapisuje się jako 8 czterocyfrowych grup szesnastkowych, a grupy rozdziela się dwukropkami, np:

  2001:0db8:85a3:0000:0000:8a2e:0370:7334

- RFC zaleca używanie małych liter, jednak ewentualna implementacja powinna dopuszczać również wielkie litery

Prezentacja i reprezentacja

- wiodące zera wewnątrz grupy mogą być pomijane, np:

  2001:0db8:85a3:0:0:8a2e:370:7334

Prezentacja i reprezentacja

- jeśli jedna lub więcej kolejno po sobie następujących grup zawiera zera, można ją (je) zastąpić parą dwukropków, np:

  2001:0db8:85a3::8a2e:370:7334

- uwaga – zamianę taką można wykonać tylko raz – w przeciwnym wypadku zapis byłby niejednoznaczny!

Prezentacja i reprezentacja

- adres loopback w IPv6 to:

  0:0:0:0:0:0:0:1

  czyli po prostu

  ::1

Prezentacja i reprezentacja

- tzw. adres nieokreślony w IPv6 to:

  0:0:0:0:0:0:0:0

  czyli po prostu

  ::

Notacja „dotted-quad“

- ponieważ cała sieć adresów IPv4 jest traktowana jak podsieć IPv6, możliwe jest takie prezentowanie adresu, które uwypukla fakt, że jest on de facto adresem v4, np:

  ::ffff:c000:0280

  może być zapisany jako:

  ::ffff:192.0.2.128

URI a IPv6

- ponieważ używanie dwukropków w adresach IPv6 może prowadzić do niejednoznaczności zapisu URI, w takich przypadkach cały adres IP bierze się w nawiasy kwadratowe, np:

  http://[2001:db8:85a3:8d3:1319:8a2e:370:7348]/

URI a IPv6

- Ewentualny numer portu zapisuje się poza adresem IP, np:

  https://[2001:db8:85a3:8d3:1319:8a2e:370:748]:430/

UNC a IPv6

- Ponieważ znak dwukropka jest zabroniony w nazwach UNC, Microsoft wprowadził własny format zapisu IPv6 wewnątrz UNC, np:

  2001:db8:85a3:8d3:1319:8a2e:370:7348

  zapisuje się jako:

  2001-db8-85a3-8d3-1319-8a2e-370-7348.ipv6-literal.net

  (Microsoft wykupił dla siebie domenę **ipv6-literal.net**, a systemy Windows rozwikłują nazwy w tej domenie bez korzystania z DNS).

IPv6 z punktu widzenia programisty

- znacząca większość bytów używanych przy programowaniu gniazd ma nowsze wersje dostosowane do IPv6, opatrzone po prostu przyrostkiem '6'

IPv6 z punktu widzenia programisty

- zamień:

  `AF_INET` → `AF_INET6`
  `PF_INET` → `PF_INET6`

getaddrinfo()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo(  const char *node,
                  const char *service,
                  const struct addrinfo *hints,
                  struct addrinfo **res  );
```

getaddrinfo()

```
const char *node
```

adres hosta wyrażony nazwą domenową lub adresem IP4/IP6 w postaci tekstu; jeśli jest równe NULL, może być uznane za adres pusty lub *loopback* (zależnie od *hints*)

getaddrinfo()

```
const char *service
```

specyfikacja usługi wyrażona jako liczba w postaci łańcucha tekstowego (np. "80") albo jako standardowa nazwa (np. "http")

getaddrinfo()

```
const struct addrinfo *hints
```

adres struktury typu addinfo, określającej dokładniejsze życzenia odnośnie szukanej informacji; może być równe NULL, wtedy przyjmuje się wartości domyślne

getaddrinfo()

```
const addrinfo **res
```

wskaźnik, pod który zostanie wpisany adres nowo utworzonej sktruktury addrinfo, wypełnionej szukanymi wartościami; strukturę taką należy zwalniać funkcją:

```
#include <sys/socket.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai);
```

getaddrinfo()

```
struct addrinfo {
    int      ai_flags;
    int      ai_family;
    int      ai_socktype;
    int      ai_protocol;
    size_t   ai_addrlen;
    struct   sockaddr *ai_addr;
    char     *ai_canonname;
    struct   addrinfo *ai_next; /* lista! */
};
```

getaddrinfo()

```
struct addrinfo *result;
struct addrinfo *res;
int error;

error = getaddrinfo("www.example.com", NULL, NULL, &result);
if (error != 0)
     exit(EXIT_FAILURE);
for (res = result; res != NULL; res = res->ai_next) {
    :
    :
}
freeaddrinfo(result);
```

IPv6 z punktu widzenia programisty

- zamień `INADDR_ANY` na `in6addr_any`

IPv6 z punktu widzenia programisty

- możesz także użyć wartości **IN6ADDR_ANY_INIT**, którą możesz zainicjować całą strukturę **struct sockaddr_in6**

  np:

```
struct in6_addr ia6 = IN6ADDR_ANY_INIT;
```

IPv6 z punktu widzenia programisty

- w strukturze **`struct sockaddr_in6`**

  również nazwy pól zawierają sufiks '6', ale pamiętaj, że pole `sin6_zero` nie istnieje.

IPv6 z punktu widzenia programisty

- zamiast **`inet_aton()`** oraz **`inet_addr()`** używaj **`inet_pton()`**

IPv6 z punktu widzenia programisty

- zamiast `inet_ntoa()` używaj `inet_ntop()`

IPv6 z punktu widzenia programisty

- zamiast `gethostbyname()` używaj `getaddrinfo()`

IPv6 z punktu widzenia programisty

- **`gethostbyaddr()`** jest nadal OK, ale jeśli możesz, używaj **`getaddrinfo()`**