Routing

część 3: wewnątrz routera

Sieci komputerowe

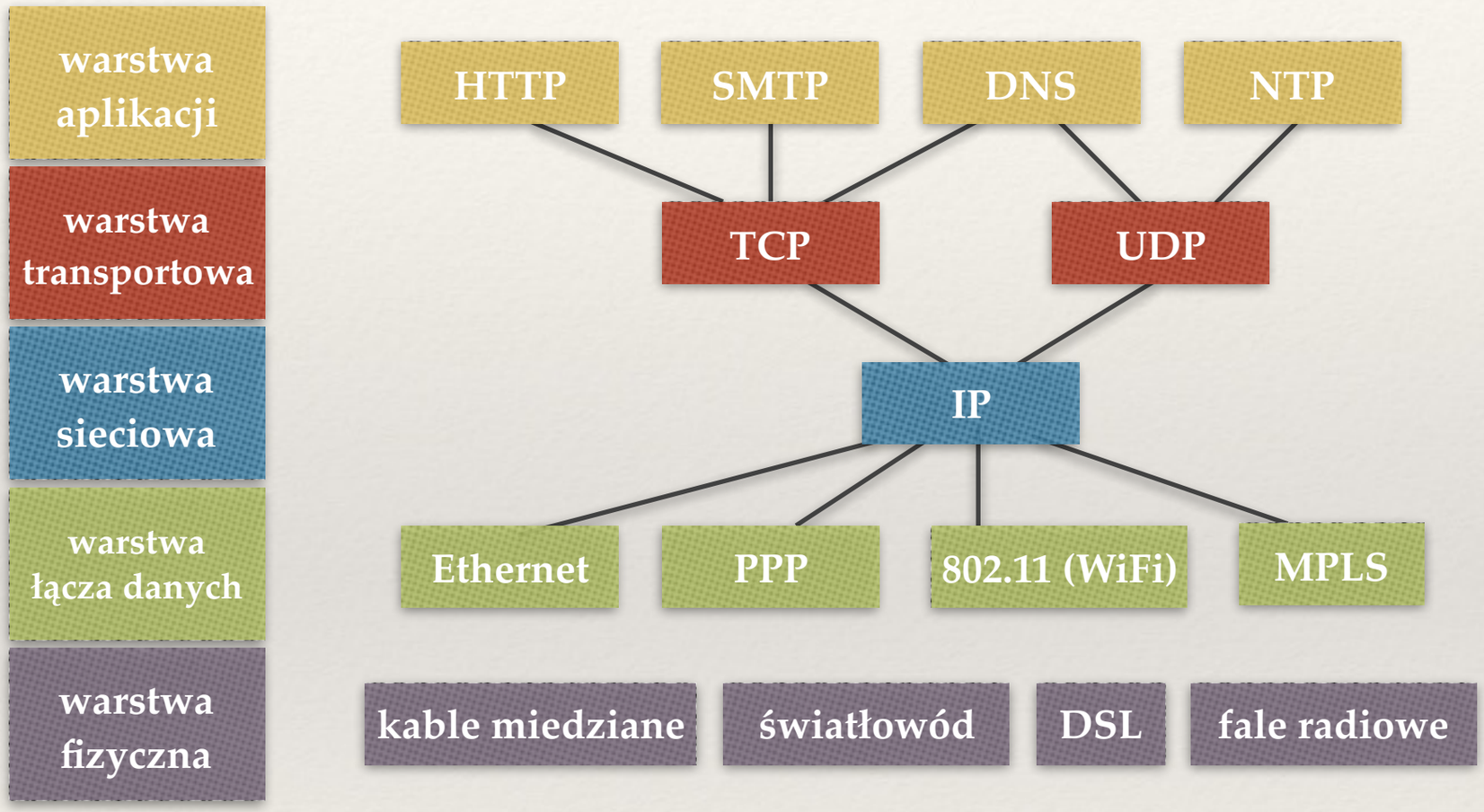
Wykład 4

Marcin Bieńkowski

Ale najpierw: piszemy prostą aplikację (gniazda UDP)

Jedna warstwa sieci i globalne adresowanie

- Każde urządzenie w sieci posługuje się **tym samym protokołem warstwy sieci**. W Internecie: protokół IP.



- Każde urządzenie ma **unikatowy adres**. W Internecie: adresy IP

Gniazda

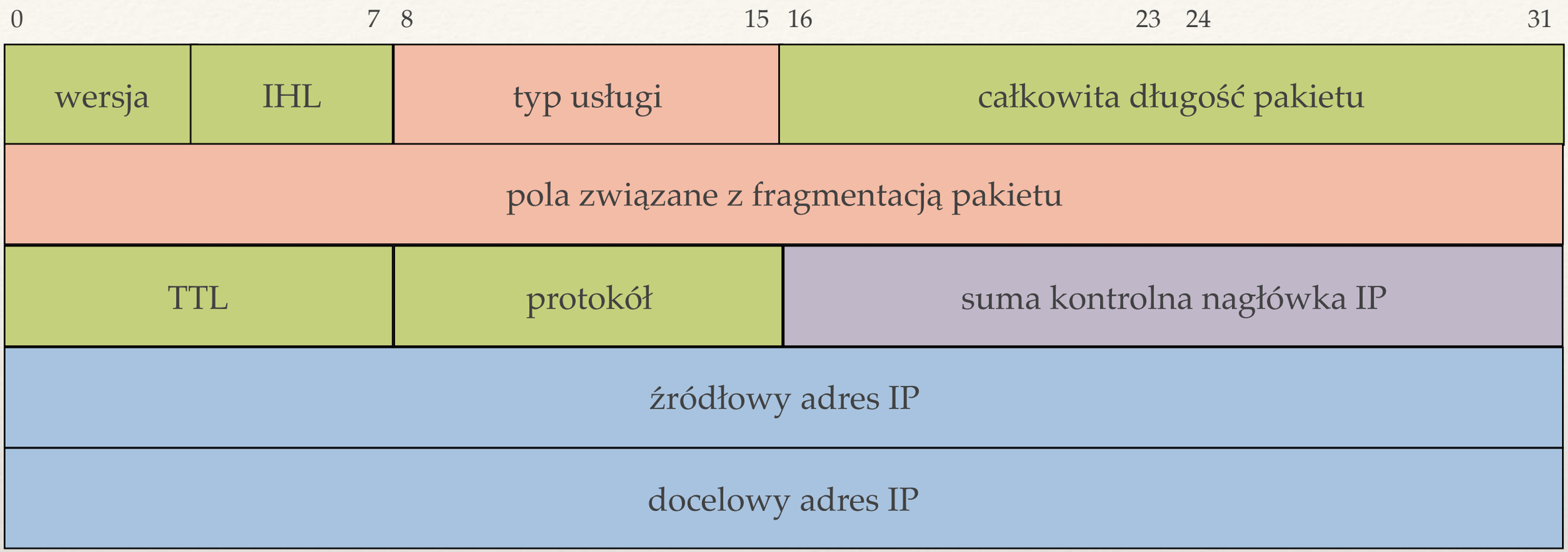
Interfejs programistyczny do nadawania i odbierania pakietów

- Umożliwiają podawanie **danych** do umieszczenia w datagramach UDP lub segmentach TCP.

dostęp do niektórych pól za pomocą funkcji gniazd

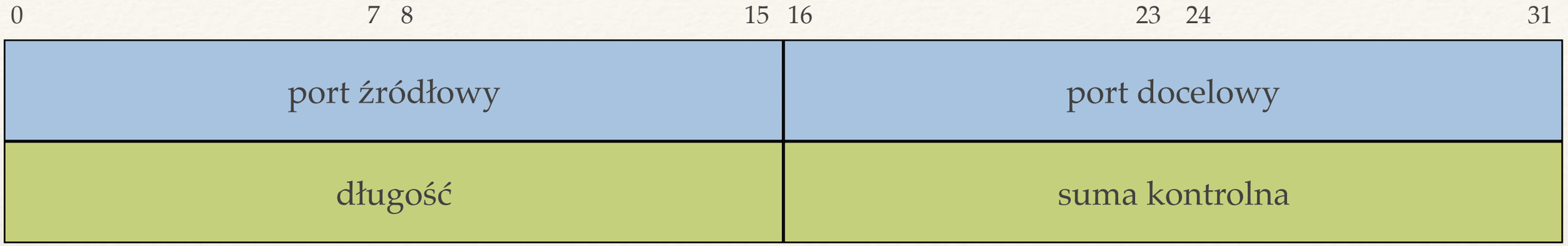
- **Gniazda surowe**: umożliwiają podawanie danych do umieszczenia bezpośrednio w danych pakietu IP.

Nagłówek pakietu IP



- Protokół = datagram jakiego protokołu przechowywany jest w danych pakietu (np. 1 = ICMP, 6 = TCP, 17 = UDP).

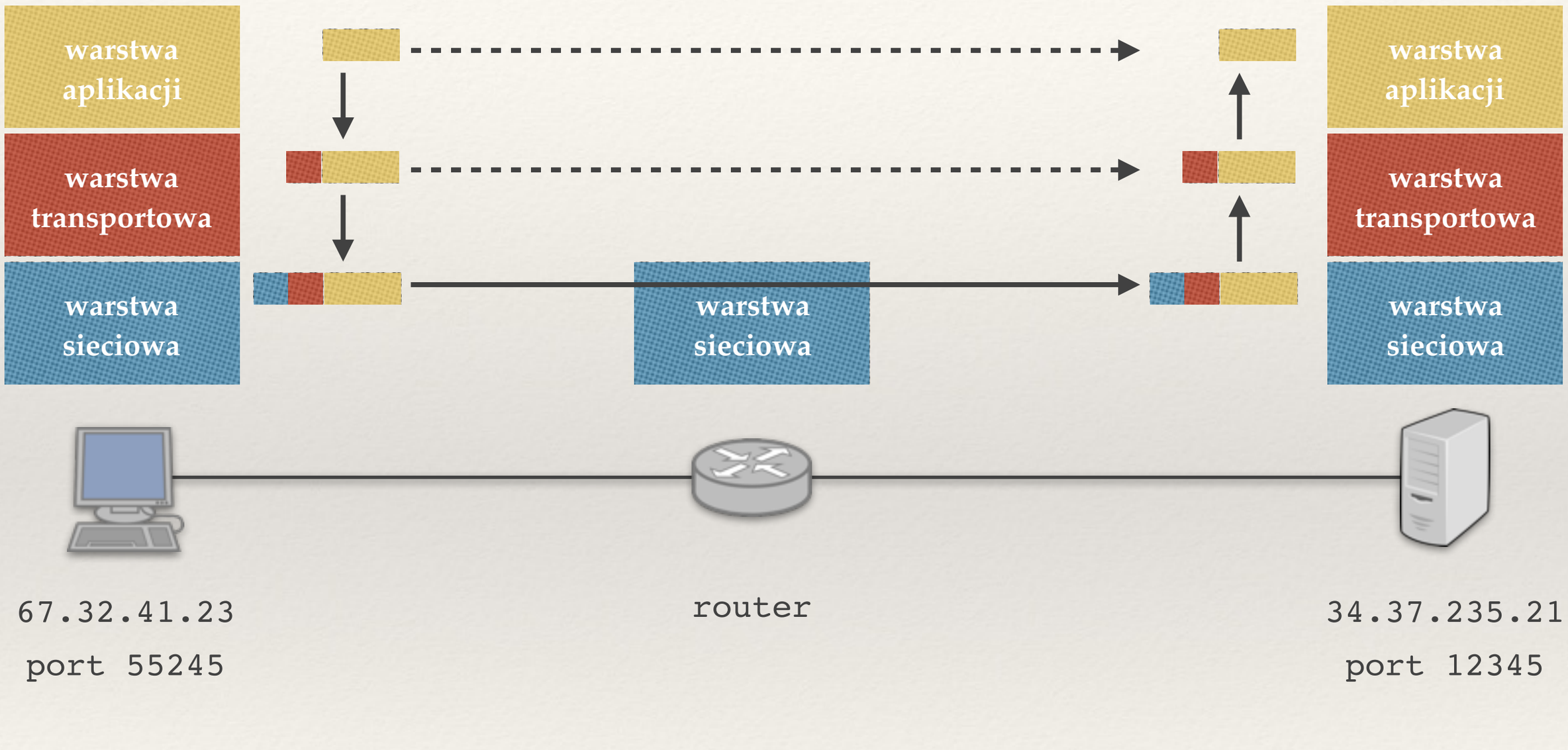
Nagłówek UDP



- **Port:**
  - liczba 16-bitowa;
  - identyfikuje aplikację wewnątrz danego komputera;

- Warstwa sieciowa zapewnia dostarczanie pakietów pomiędzy komputerami, warstwa transportowa pomiędzy aplikacjami.

Enkapsulacja i dekapsulacja



Gniazdo UDP

- Identyfikuje jeden koniec komunikacji UDP.
- Opisywane przez parę (lokalny adres IP, lokalny port).
- Związane z konkretnym procesem.

Tworzenie gniazda

```
#include <arpa/inet.h>
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

Wiązanie gniazda z adresem i portem

Struktura adresowa jak w przypadku gniazda surowego, ale wypełniamy w niej też port.

```
struct sockaddr_in server_address;
bzero (&server_address, sizeof(server_address));
server_address.sin_family      = AF_INET;
server_address.sin_port        = htons(12345);
server_address.sin_addr.s_addr = htonl(INADDR_ANY);


bind (
   sockfd,
   (struct sockaddr*)&server_address,
   sizeof(server_address)
);
```

demonstracja

Odbieranie pakietu z gniazda

Identycznie jak w przypadku gniazd surowych.

```
struct sockaddr_in   sender;
socklen_t            sender_len = sizeof(sender);
u_int8_t             buffer[IP_MAXPACKET+1];

ssize_t packet_len = recvfrom (
    sockfd,
    buffer,
    IP_MAXPACKET,
    0,
    (struct sockaddr*)&sender,
    &sender_len
);
```

pakiet jako ciąg bajtów

informacje o nadawcy

Wysyłanie pakietu przez gniazdo

Identycznie jak w przypadku gniazd surowych, ale `recipient` musi zawierać również port.

```
char* reply = "Thank you!“;
ssize_t reply_len = strlen(reply);
```

```
ssize_t bytes_sent = sendto (
  sockfd,
  reply,
  reply_len,
  0,
  (struct sockaddr*)&recipient,
  sizeof(recipient)
);
```

informacje o odbiorcy, np. to co wpisaliśmy do struktury `sender`

Zamykanie gniazda

Zwalnia zasoby związane z gniazdem.

```
close(sockfd);
```

Kod serwera UDP

```
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in server_address;
bzero (&server_address, sizeof(server_address));
server_address.sin_family      = AF_INET;
server_address.sin_port        = htons(12345);
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
bind (sockfd, (struct sockaddr*)&server_address, sizeof(server_address));

for (;;) {
  struct sockaddr_in sender;
  socklen_t          sender_len = sizeof(sender);
  u_int8_t           buffer[IP_MAXPACKET+1];

  ssize_t datagram_len = recvfrom (sockfd, buffer, IP_MAXPACKET, 0,
                                   (struct sockaddr*)&sender, &sender_len);

  char sender_ip_str[20];
  inet_ntop(AF_INET, &(sender.sin_addr), sender_ip_str, sizeof(sender_ip_str));

  printf ("Received UDP packet from IP address: %s, port: %d\n", sender_ip_str, ntohs(sender.sin_port));

  buffer[datagram_len] = 0;
  printf ("%ld-byte message: +%s+\n", datagram_len, buffer);

  char* reply = "Thank you!";
  ssize_t reply_len = strlen(reply);
  sendto(sockfd, reply, reply_len, 0, (struct sockaddr*)&sender, sender_len);
}

close (sockfd);
```

Brak obsługi błędów, plików nagłówkowych, etc.

demonstracja

cały kod serwera

Wiązanie z portem c.d.

- Serwer związuje się z danym portem funkcją `bind()`.
  - Do związania z portem ≤ 1024 potrzebne uprawnienia administratora.

- Jeśli wyślemy coś przez gniazdo nie związując go z lokalnym portem, jądro przydzieli do tego gniazda automatycznie port.
  - Port emeferyczny (zazwyczaj ≥ 32768).
  - Tak działa większość klientów (np. program `nc`).

Kod klienta UDP

Brak obsługi błędów, etc.

```
int main()
{
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    struct sockaddr_in server_address;
    bzero (&server_address, sizeof(server_address));
    server_address.sin_family      = AF_INET;
    server_address.sin_port        = htons(12345);
    inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr);

    char* message = "Hello server!";
    ssize_t message_len = strlen(message);

    sendto(sockfd, message, message_len, 0,
           (struct sockaddr*) &server_address,
           sizeof(server_address));

    close (sockfd);
}
```

demonstracja

cały kod klienta

Wysyłanie pakietu UDP na adres rozgłoszeniowy

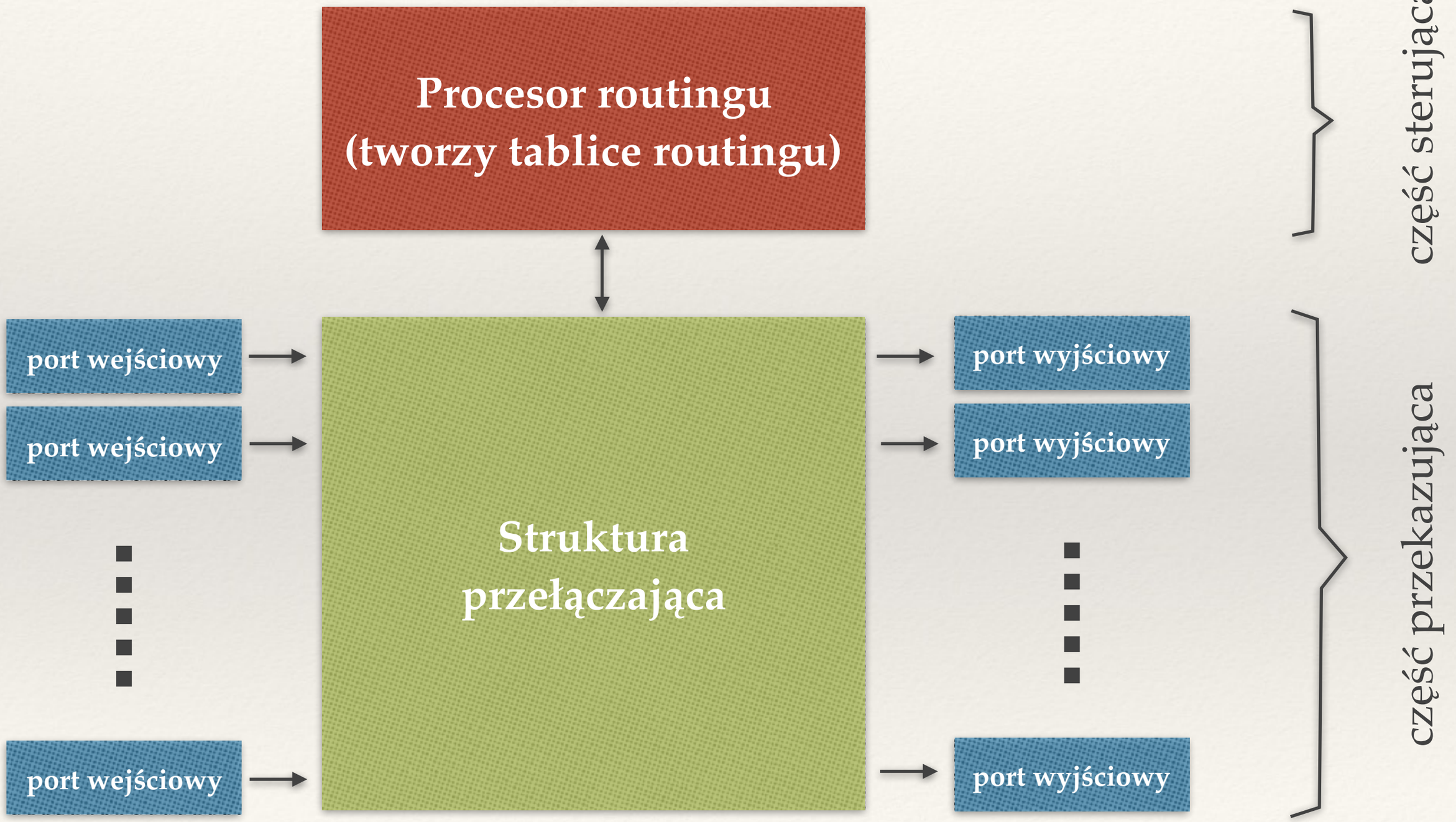
Wystarczy włączyć odpowiednią opcję gniazda.

```
int broadcastPermission = 1;
setsockopt (sockfd, SOL_SOCKET, SO_BROADCAST,
(void *)&broadcastPermission,
sizeof(broadcastPermission));
```

Wewnątrz routera

Budowa routera

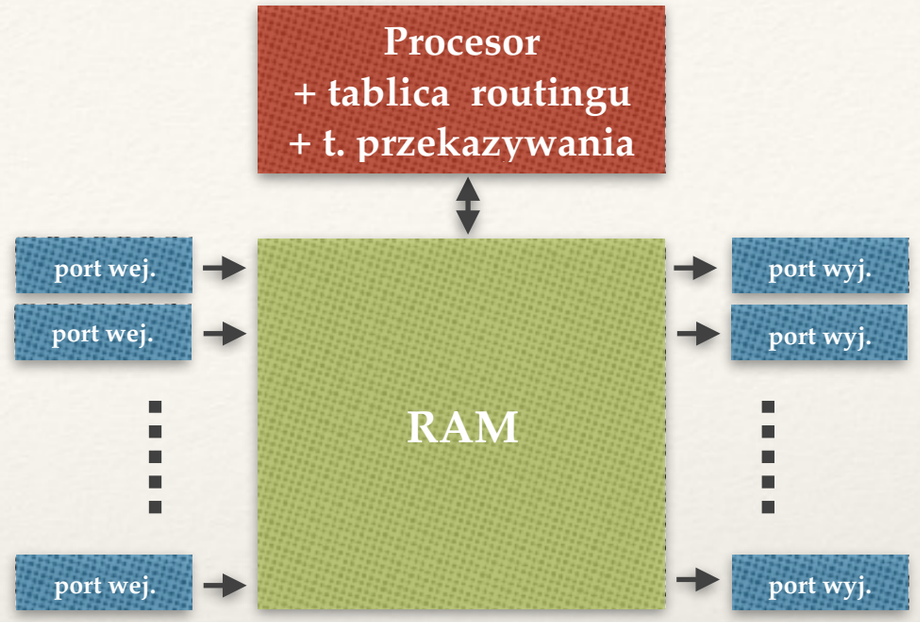
Router podejmuje decyzję na podstawie nagłówka pakietu w oparciu o tablicę przekazywania.



Przełączanie pakietów za pomocą RAM

Wczesne generacje routerów (jak PC).

- Brak struktury przełączającej.
- Tablica przekazywania w części sterującej.

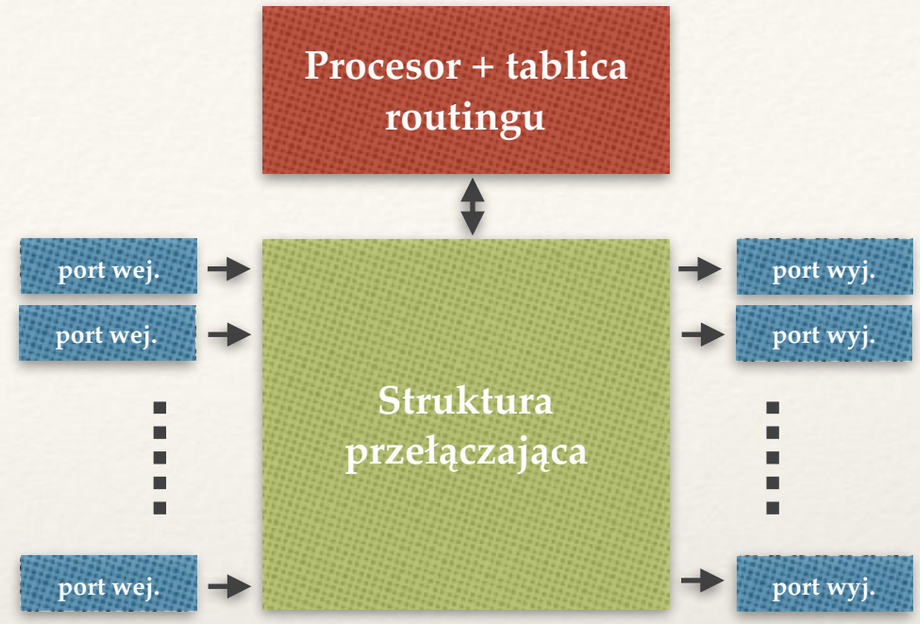


- **Działanie:**
  - Port wejściowy odbiera pakiet i zgłasza przerwanie.
  - Procesor kopiuje pakiet do RAM.
  - Wolny port wyjściowy zgłasza przerwanie.
  - Procesor zapisuje pakiet do RAM.

Późniejsze generacje routerów: szyna zamiast RAM

Przełączanie pakietów za pomocą sieci przełączającej

Współczesne generacje routerów.



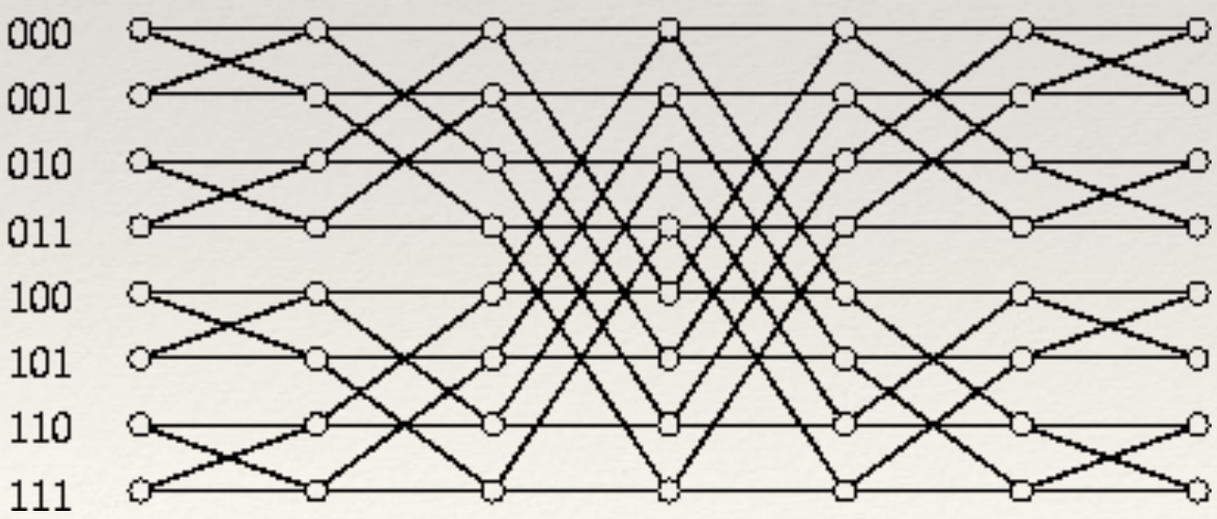
- **Procesor:**
  - Otrzymuje niektóre pakiety (RIP, OSPF).
  - Tworzy tablice przekazywania i wysyła je do portów wejściowych.

- **Port wejściowy:**
  - Odbiera pakiet z łącza.
  - Uaktualnia nagłówek IP (TTL, suma kontrolna).
  - Sprawdza, do którego portu wyjściowego go przesłać.

Struktura przełączająca

- **Cel: Przekazywać pakiety z prędkością łącza (lub zbliżoną).**
  - N portów wejściowych o prędkości R → chcemy przepustowość N x R (typowe wartości to 10 Gbit/s - 1 Tbit/s).

- **Sieci połączeń znane z sieci procesorów w systemach multiprocesorowych.**
  - każdy z każdym: $O(N^2)$ połączeń (niepraktyczne);
  - sieci Benesa i pochodne: O(N log N) połączeń (potrafią bezkolizyjnie przesłać dowolną permutację).

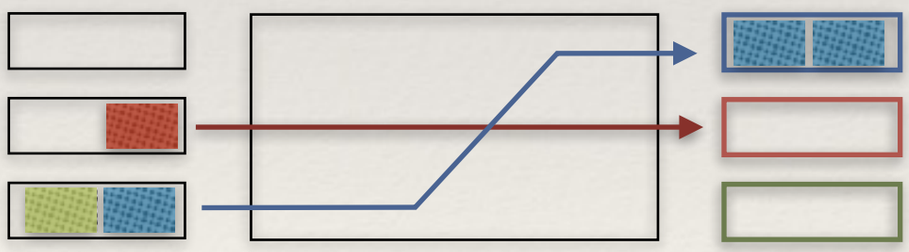


Obrazek ze strony https://www.cs.cmu.edu/afs/cs.cmu.edu/project/phrensy/pub/papers/AroraLM94/node7.html

Bufory z kolejkami pakietów

- **Przy portach wyjściowych.**
  - Utrata pakietów przy przeciążeniach (wykład 1).

- **Przy portach wejściowych.**
  - Jeśli przepustowość struktury przełączającej jest za mała.
  - Pakiety kierowane do zajętych łącz wyjściowych są blokowane.
  - Problem blokowania przodu kolejki:



Niebieski pakiet musi czekać i blokuje wysłanie pakietu zielonego.

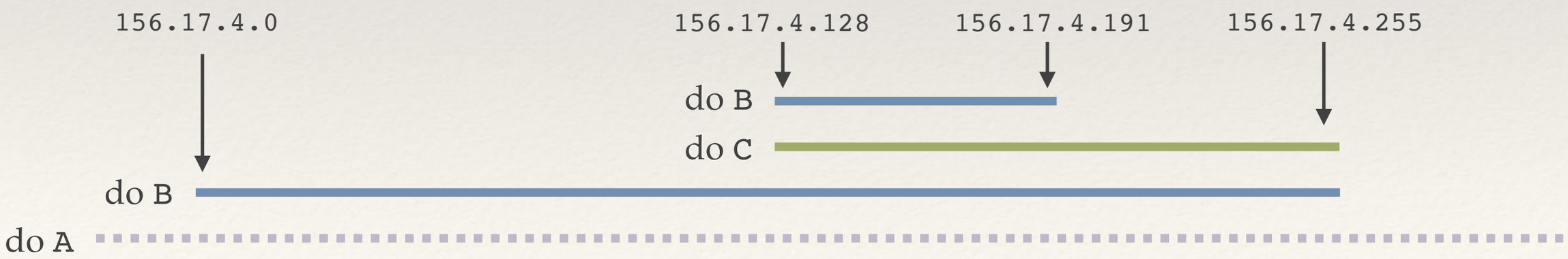
Rozwiązywane przez wirtualne kolejki pakietów: jedna kolejka dla kadego portu wyjściowego.

Porty wejściowe

Tablice przekazywania

Jeśli więcej niż jedna reguła pasuje, wybierana jest ta, która jest najdłuższym prefiksem = **mechanizm LPM** (longest prefix match)

prefiks CIDR	akcja
`0.0.0.0/0`	do portu A
`156.17.4.0/24`	do portu B
`156.17.4.128/25`	do portu C
`156.17.4.128/26`	do portu B



Struktury danych dla LPM

- Struktura danych dla LPM musi obsługiwać:
  - **lookup (adres)** — miliony razy / sek.
  - **insert (prefix)** / **delete (prefix)** — setki razy / sek.

- Notacja:
  - n - liczba prefiksów w tablicy;
  - w - rozmiar adresu (adres mieści się w słowie).

Implementacja LPM (1)

- **Lista prefiksów**
  - pamięć: O(n)
  - lookup: O(n)
  - insert: O(1), delete: O(n)

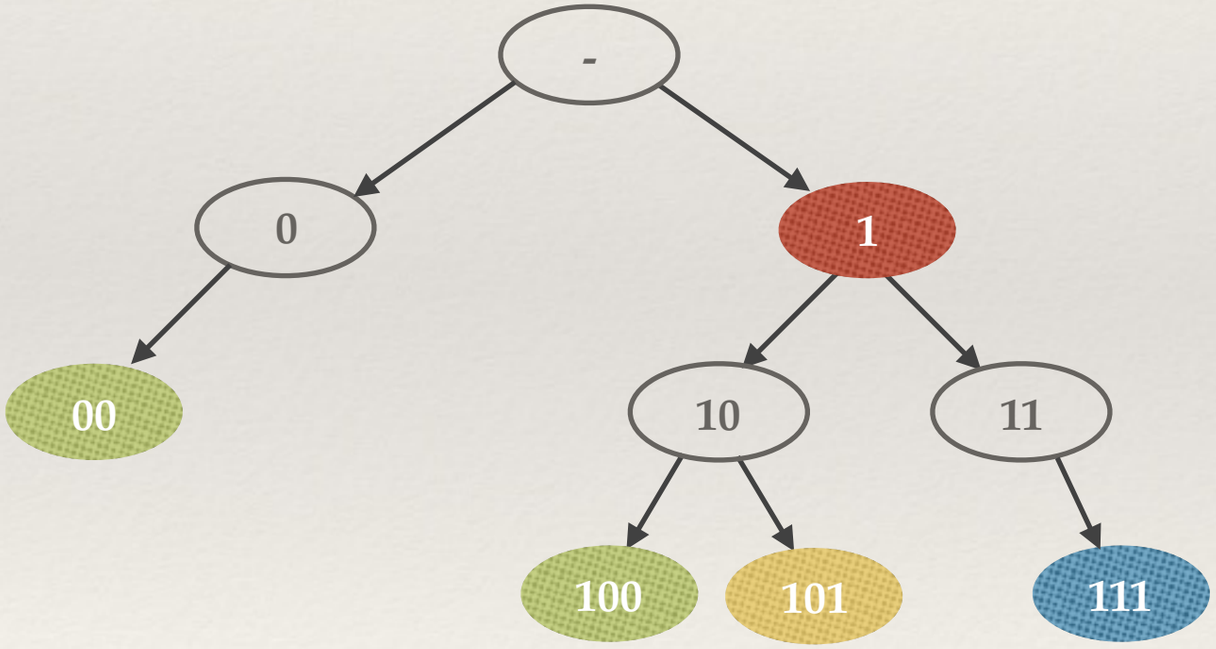
Implementacja LPM (2)

- **Tablice haszujące**
  - w+1 tablic (dla każdej długości prefiksu)
    - w czasach klas adresów IP wystarczało 5 tablic
  - pamięć: O(n)
  - lookup: O(w) (oczekiwany)
  - insert, delete: O(1) (oczekiwany)

Implementacja LPM (3)

- **Drzewa trie (faktycznie stosowane)**
  - pamięć: O(n x w)
  - lookup: O(w)
  - insert, delete: O(w)

Kompresja ścieżek bez rozgałęzień daje pamięć O(n).



Przechodzimy drzewo w dół i zwracamy ostatnią pasującą regułę:

- dla adresu `10000…`
  → port zielony;
- dla adresu `11000…`
  → port czerwony.

Implementacja LPM (4)

- **Trie ze dodatkowymi krawędziami skracającymi**
  - lookup: O(log w)
  - insert, delete: O(n) (przynajmniej w najgorszym przypadku)
  - Problem otwarty: czy da się wszystkie operacje w O(log w)?

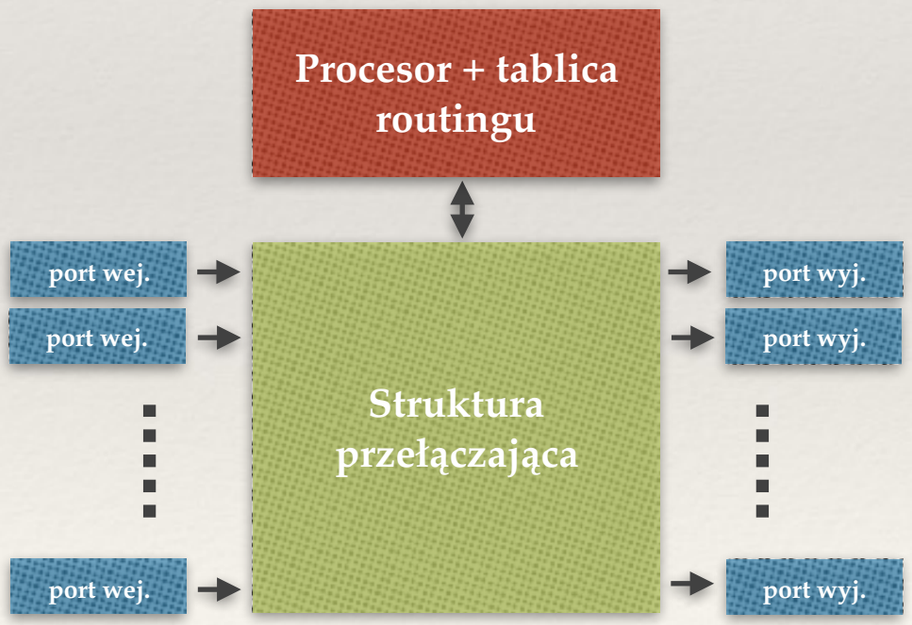
Implementacja LPM (5)

- **Rozwiązania sprzętowe oparte o TCAM (faktycznie stosowane)**
  - TCAM = ternary content addressable memory.
  - Przechowujemy pary *(p,m)*.
  - Dla adresu w można równolegle znaleźć wszystkie pary takie, że $w \,\&\, m = p \,\&\, m$ (bitowy and) = wszystkie pasujące prefiksy.
  - Sprzętowo wybieramy najdłuższy z nich.

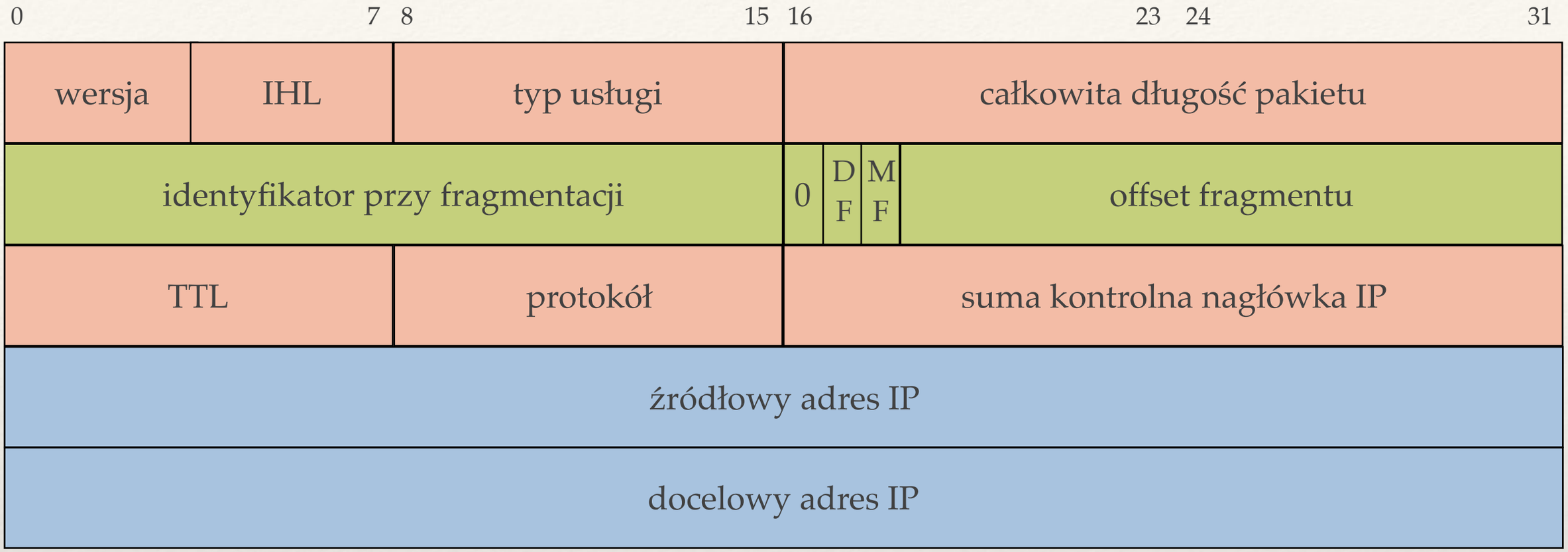
Porty wyjściowe

Fragmentacja (1)

- Jeśli rozmiar pakietu jest większy niż MTU (*maximum transmission unit*) łącza wyjściowego, to pakiet jest dzielony na fragmenty. Przykładowo:
  - MTU Ethernetu = 1500 bajtów,
  - MTU sieci bezprzewodowej 802.11 = 7981 bajtów.



Fragmentacja (2)



- **Dzielenie na dowolnym routerze na trasie**
  - Fragmenty dostają identyczny identyfikator.
  - MF = czy jest więcej fragmentów?
  - Offset = numer pierwszego bajtu w oryginalnym pakiecie.

- **Łączenie fragmentów dopiero na komputerze docelowym.**

Fragmentacja jest nieefektywna

- Dodatkowa praca dla routerów.

- Dodatkowe narzut (nagłówki pakietów):
  - do wysłania 140.000 bajtów, pierwsze łącze na trasie umożliwia przesłanie 1400 bajtów w pakiecie, najmniejsze na trasie — 1250 bajtów;
  - bez fragmentacji: 140.000 / 1250 = 112 pakietów
  - z fragmentacją: wysyłamy 140.000 / 1400 = 100 pakietów, ale każdy dzielony później na dwa.

- Jak poznać najmniejsze łącze na trasie?

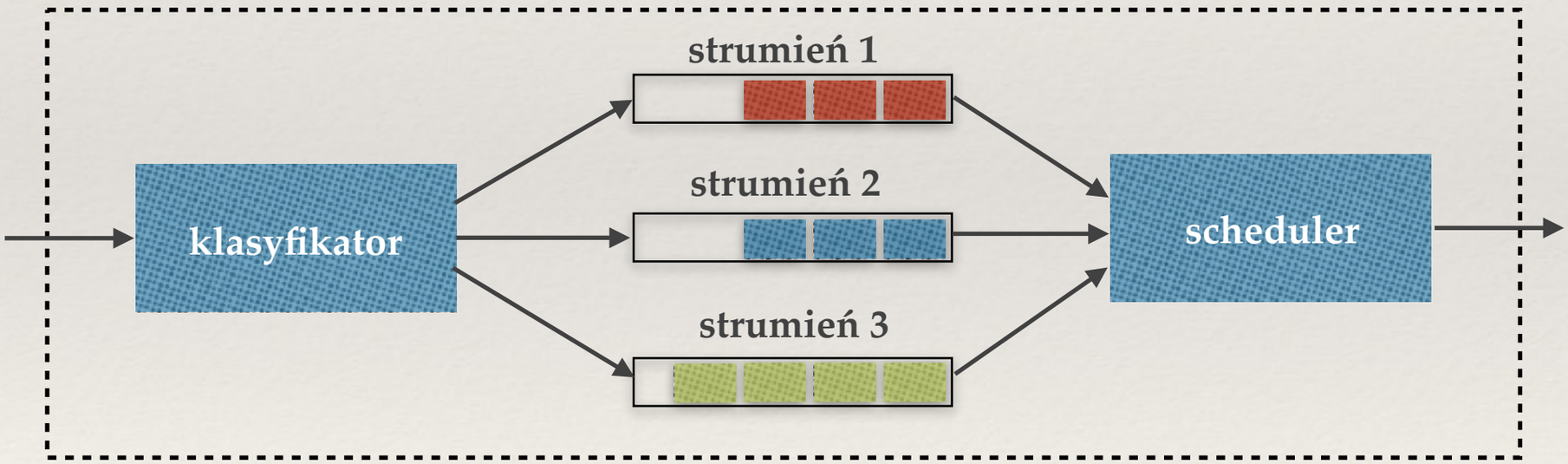
Wykrywanie MTU dla ścieżki

- Ustaw bit DF (*don't fragment*) w nagłówku IP.
- Jeśli konieczna fragmentacja na routerze:
  - pakiet wyrzucony;
  - router odsyła komunikat ICMP (*destination unreachable, can't fragment*) z rozmiarem MTU kolejnego łącza.
- Zmniejsz odpowiednio rozmiar pakietu i ponów wysyłanie.

Co się dzieje w buforze wyjściowym?

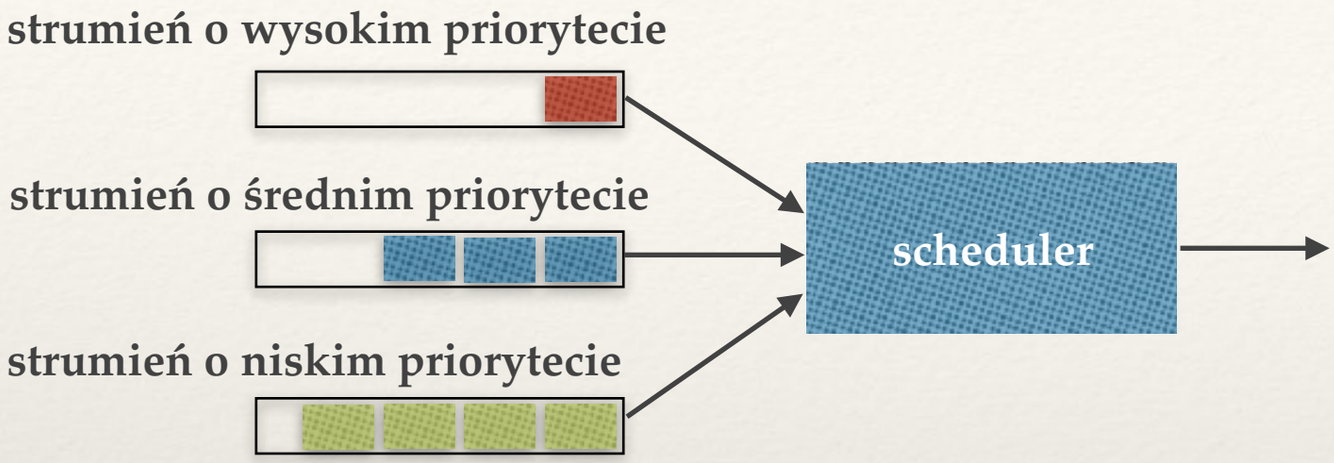
- **Kolejka FIFO:** pakiety wysyłane w takiej kolejności jak nadeszły.

- **Szeregowanie pakietów:** Przypisujemy pakiety do strumieni (na podstawie adresu i portu źródłowego + docelowego). Pakiety szeregowane w zależności od strumienia.

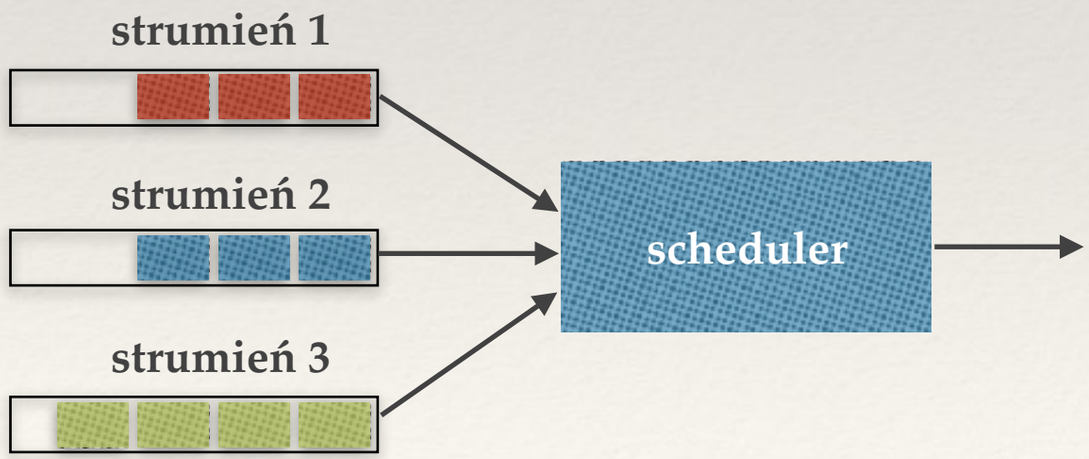


Szeregowanie pakietów w buforze

- Szeregowanie względem priorytetów strumieni



- Szeregowanie cykliczne (*round-robin)*: po tyle samo pakietów z każdego strumienia.

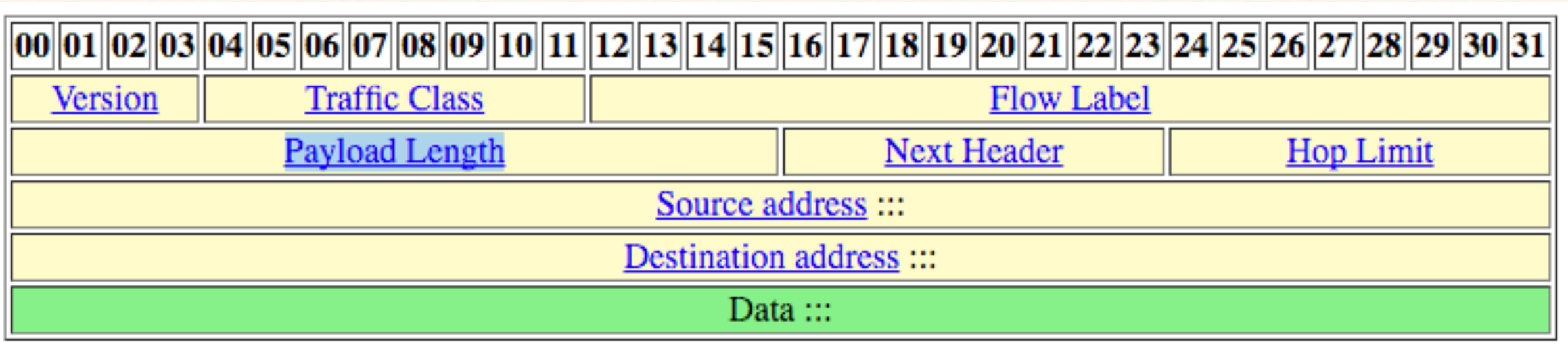


IPv6

Dlaczego nowa wersja?

- Adresy IPv4 wyczerpują się (IANA oddała ostatnią pulę regionalnym rejestratorom 3 lutego 2011 r.)
- 20 lat temu rozpoczęto pracę nad kolejną wersją protokołu (IPv6)
- 128-bitowe adresy.

Nagłówek IPv6



- 128-bitowe adresy.
- Mniejszy narzut dla routerów:
  - nagłówki stałej długości,
  - brak fragmentacji,
  - brak sumy kontrolnej,
  - etykieta strumienia (nie trzeba patrzeć na porty).

Obrazek ze strony http://www.networksorcery.com/enp/protocol/ipv6.htm

Adresy IPv6

- **Notacja = 8 bloków po 4 cyfry szesnastkowe, rozdzielonych przez dwukropek.**
  - A = `2001:0db8:0000:0000:0000:0000:1428:0000`
  - localhost = `0000:0000:0000:0000:0000:0000:0000:0001/128`
- **Uproszczenia zapisu:**
  - Można opuszczać wiodące zera w każdym bloku (zostawiając jedno).
  - Jeden z ciąg zerowych bloków zer można zastąpić przez `::`.
  - Przykłady:
    - A = `2001:db8::1428:0`
    - `localhost = ::1/128`

IPv4 a IPv6

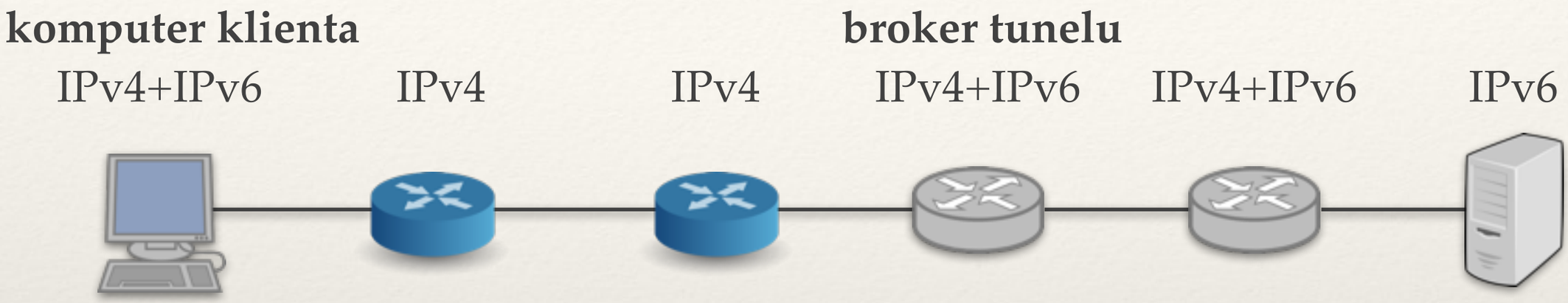
- Większość dużych serwisów (Google, Facebook, …) ma swoje wersje IPv6.
  - Osobne serwery lub serwery z podwójnym stosem (potrafią interpretować pakiety IPv4 i IPv6).

- Duża część routerów w rdzeniu Internetu potrafi przesyłać pakiety IPv6.

- Ale ISP udostępniający Internet użytkownikom końcowym rzadko dają im obecnie możliwość korzystania z IPv6 (wyjątek: sieci komórkowe).

IPv4 a IPv6: mechanizmy migracji

Tunelowanie 6in4 = pakiety IPv6 przesyłane jako dane pakietów IPv4.



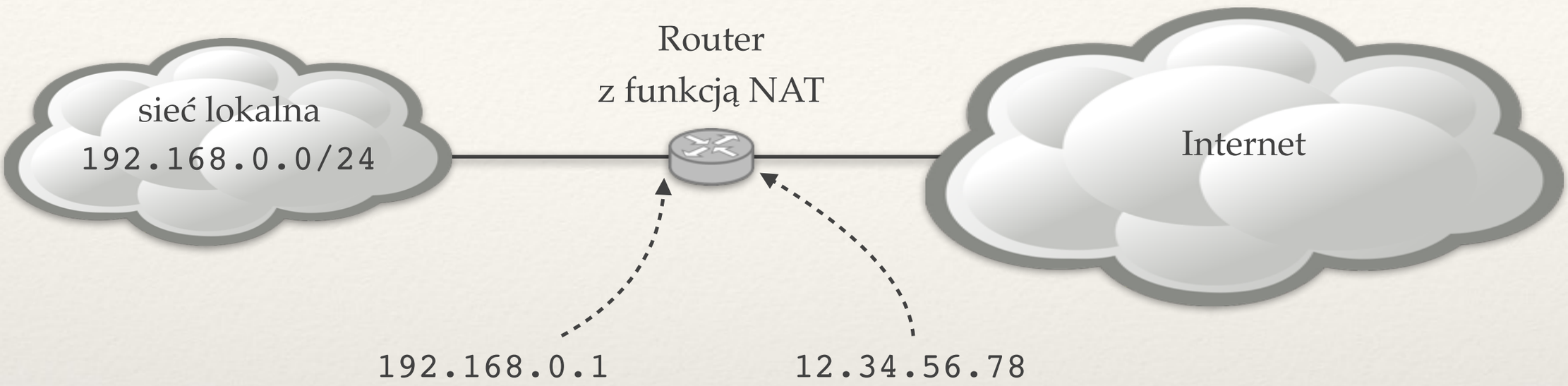
Pomiędzy komputerem a brokerem tworzony jest logiczny kanał.

NAT

Coraz większe zapotrzebowanie na adresy IP

- Adresy IPv4 wyczerpują się.
- Wdrożenie IPv6 wciąż trwa.
- Adresy IP są dość kosztowne → pojedyncze IP dla całych firm.

NAT



- Z reszty Internetu cała sieć lokalna wygląda tak samo, jak pojedynczy komputer z adresem `12.34.56.78`.
- Nie można (w normalny sposób) dostać się z Internetu do komputerów z LAN. Jak sobie z tym radzić?

Co robi router z funkcją NAT?

- **Komputer z sieci `192.168.1.0/24` wysyła pakiet do Internetu.**
  - Pakiet ma:
    - źródłowy adres i port = A, P_A,
    - docelowy adres i port = C, P_C.
  - Pakiet przechodzi przez router NAT o zewnętrznym adresie B, który na podstawie krotki (A, P_A, C, P_C) wybiera port P_B.
  - W pakiecie adres i port źródłowy zostają podmienione na (B, P_B).

- **Tablica NAT:**
  - Przechowuje przez pewien czas przypisanie (A, P_A, C, P_C) → P_B.
  - Dla kolejnych podobnych pakietów przypisanie będzie takie samo.
  - Jeśli przychodzi pakiet do (B, P_B) to jego adres i port docelowy zostanie podmieniony na (A, P_A).

Adresy prywatne

Adresy przeznaczone do sieci lokalnych.

- Pakiety z takimi adresami nie powinny być przekazywane dalej przez routery.
- W różnych sieciach mogą być te same adresy.
- Pule adresów:
  - `10.0.0.0/8` (jedna sieć klasy A),
  - `172.16.0.0/12` (16 sieci klasy B),
  - `192.168.0.0/16` (256 sieci klasy C).

Zalety i wady NAT

Zalety:

- Rozwiązuje problem braku adresów IP.
- Można zmienić adresy IP wewnątrz sieci bez powiadamiania Internetu.
- Można zmienić ISP pozostawiając adresowanie IP wewnątrz sieci.

Wady:

- Nieosiągalność komputerów z Internetu (aplikacje P2P).
- Psucie modelu warstwowego (router modyfikuje treść pakietu).

Lektura dodatkowa

- Kurose & Ross: rozdział 4.
- Tanenbaum: rozdział 5.
- Stevens: rozdział 8.
- Dokumentacja online:
  - http://www.networksorcery.com/enp/protocol/ipv6.htm
  - http://www.networksorcery.com/enp/protocol/udp.htm
  - Beej's Guide to Network Programming: http://beej.us/guide/bgnet/