Linux i sieci: Sieci komputerowe

Projekt „Matematyka dla Ciekawych Świata”,
Robert Ryszard Paciorek
<rrp@opcode.eu.org>

2024-10-03

1 Podstawy TCP/IP

Sieci komputerowe działają na zasadzie przesyłania informacji w postaci porcji, z których każda posiada co najmniej informację o adresie odbiorcy (zwykle też nadawcy), nazywanych ramkami lub pakietami. Kierowanie pakietów w odpowiednie miejsce odbywa się na podstawie adresu pakietu i nie jest związane z fizycznym zestawianiem łącza pomiędzy nadawcą a odbiorcą - każdy pakiet jest kierowany niezależnie, a w ramach pojedynczego łącza (kanału transmisji) mogą być przekazywane pakiety adresowane do różnych odbiorców. Nazywane jest to komutacją pakietów, w odróżnieniu od komutacji łącza (która występowała np. w klasycznej, analogowej telefonii, gdzie przekaźniki w centralach dokonywały zestawienia połączeń elektrycznych między dwoma aparatami telefonicznymi).

To właśnie umieszczanie w nagłówku każdego z pakietów adresów umożliwia kierowanie ruchem takich pakietów bez wcześniejszego fizycznego zestawiania łącz. W oparciu o ten adres host jest w stanie rozpoznać czy pakiet jest przeznaczony dla niego czy nie, a przełączniki i routery mogą kierować pakiety do odpowiednich fragmentów sieci.

Jeżeli w sieci komputerowej host A chce się komunikować z hostem B, a host C z hostem D nie musimy zapewnić im osobnych łączy do przeprowadzania takiej komunikacji, tak jak to było na przykład w klasycznej telefonii analogowej, gdzie dwie takie ”rozmowy” wymagałyby zestawienia osobnych fizycznych kabli od jednego abonenta do drugiego (odbywało się to za pomocą przekaźników w centralach). W sieci komputerowej hosty te będą wytwarzały odpowiednio zaadresowane pakiety i reagowały tylko na pakiety zaadresowane do nich. Dzięki temu pakiety, stanowiące osobne strumienie komunikacji, mogą być z łatwością przesyłane tym samym fizycznym łączem. Host C może słyszeć lub nie komunikację pomiędzy hostami A i B (zależy to od różnych czynników, na przykład od wykorzystywanej sieci i względnego położenia tych hostów), natomiast wie że to ”nie do niego”.

Jeżeli w ramach sieci mamy jakieś urządzenie dzielące ją na mniejsze kawałki, podział ten będzie się odbywał w oparciu o adresy pakietów – urządzenie takie do danej sieci będzie przekazywało tylko pakiety adresowane do hostów w tej sieci. Mamy do czynienia z przełączaniem, czyli komutacją pakietów – w odróżnieniu od (omówionej pokrótce na przykładzie telefonii) komutacji łączy, która polegała na zestawianiu fizycznego łącza pomiędzy komunikującymi się hostami.

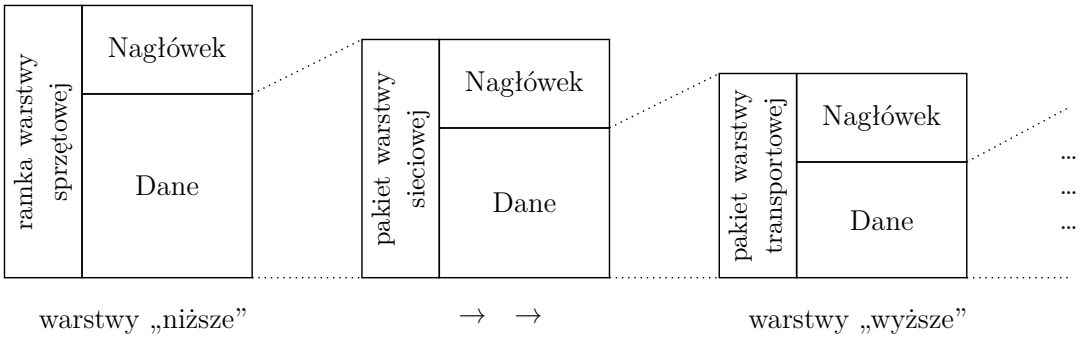
1.1 Struktura warstwowa

Protokoły sieciowe na ogół tworzą tak zwane stosy protokołów, gdzie dane opakowywane są kolejno w nagłówki kolejnych protokołów. Każdy z protokołów w takim stosie pełni dedykowane mu funkcje i na ogół nie ingeruje ani w protokoły warstwy niższej, ani w przenoszoną przez niego zawartość, czyli protokoły warstwy wyższej z danymi.

Komunikacja sieciowa typowo posiada strukturę warstwową. W modelu OSI wyróżnia się 7 warstw:

1. fizyczną (pierwszą) definiującą aspekty związane z fizycznym przesyłem sygnału takie jak częstotliwości radiowe, poziomy napięć, etc.; określa sposób transmisji kolejnych bajtów
2. łącza danych (drugą) definiującą aspekty związane z formatem ramki, protokoły ustalania zasad dostępu do medium transmisyjnego, itd.; określa sposób transmisji porcji danych pomiędzy hostami w jednej sieci

3. sieciową (trzecią) definiującą aspekty związane z formatem pakietu, adresacją i zasady routingu umożliwiające zapewnienie łączności pomiędzy różnymi sieciami; określa sposoby transmisji porcji danych pomiędzy sieciami
4. transportową (czwartą) odpowiedzialną za podział strumienia na porcje informacji, kontrolę nad poprawnością transmisji, adresację usług w ramach hosta
5. sesji (piątą)
6. prezentacji (szóstą)
7. aplikacji (siódmą)



Rysunek 1: Struktura warstwowa protokołów sieciowych

W modelu TCP/IP wyróżnia się 4 warstwy:

1. Dostępu do sieci - obejmującą warstwy 1 i 2 modelu OSI
2. Internetu - obejmującą warstwę 3 modelu OSI
3. Transportową - obejmującą warstwę 4 modelu OSI
4. Aplikacji - obejmującą warstwy 5, 6 i 7 modelu OSI

Z punktu widzenia modelu TCP/IP można powiedzieć o enkapsulacji danych kolejnych warstw w ramach warstwy niższej, czyli „surowe” dane (np. strona HTML) obudowywane są strukturą opisywaną przez warstwę aplikacji (np. nagłówkami HTTP), następnie całość ta umieszczana jest w polu danych pakietu warstwy transportowej (np. TCP), ten z kolei w polu danych pakietu IP (warstwy sieciowej), na koniec pakiet IP jest umieszczany w polu danych ramki warstwy dostępu do sieci (np. ramki ethernetowej). W ramach podróży przez kolejne sieci pakiet IP jest wyjmowany i wkładany w kolejne ramki warstwy dostępu do sieci, na ogół tylko z niewielkimi ingerencjami w zawartość tego pakietu (prawie zawsze nie dochodzącymi do pola danych pakietu TCP lub datagramu UDP, czyli nie wykraczającymi poza warstwę 4 OSI).

1.2 Protokół IP

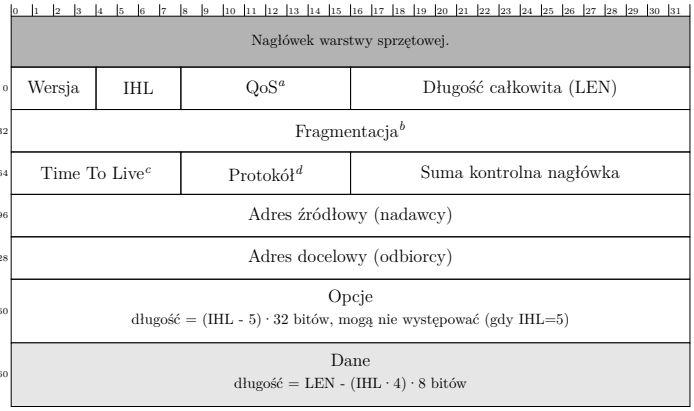
Protokół IP (Internet Protocol) odpowiedzialny jest przede wszystkim za sposób adresacji hostów oraz reguły komutacji pakietów (routing). Jest on wspomagany przez kolejny protokół z tej rodziny - ICMP (Internet Control Message Protocol), którego zadaniem jest przekazywanie informacji kontrolnych np. o nieosiągalności hosta docelowego, odrzuceniu przetwarzania pakietu ze względu na zbyt dużą liczbę skoków (gdy wartość pola TTL z nagłówka IP wyniesie zero) a także pingi (zarówno żądanie jak i odpowiedź).

Jako że IP stworzony jest do łączenia różnych fizycznych sieci w jedną sieć logiczną (internet), to pozwala on na wydzielanie w oparciu o zasady adresacji mniejszych fragmentów sieci, nazywanych niekiedy podsieciami. Oczywiście zapewnia też mechanizmy przekazywania pakietów pomiędzy takimi podsieciami.

1.3 Adresacja IP

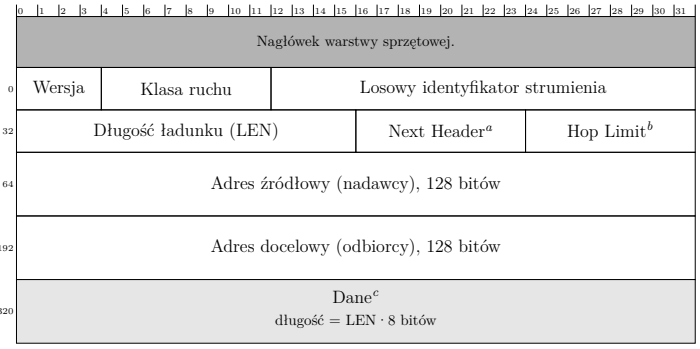
Adresy hostów (nazywane adresami IP) są to 32-bitowe (w IPv4) lub 128-bitowe (w IPv6) liczby. Adresy IPv4 zapisywane są najczęściej w notacji kropkowo-dziesiętnej, gdzie każdy bajt (ciąg 8 bitów) zapisywa-

Pakiet IPv4:



a. Dokładniej dwa pola: DSCP – priorytetyzacja ruchu DiffServ i ECN – informacja o przeciążeniu
b. Obsługa mechanizmu podziału pakietu IP na mniejsze części (gdy nie mieści się do ramki warstwy drugiej). Dokładniej są to trzy pola: Identification – numer identyfikujący zbiór fragmentów, Flagi – informujące o zakazie lub dokonaniu fragmentacji, oraz Offset – przesunięcie fragmentu względem całości pakietu.
c. Licznik zmiejszany gdy pakiet przechodzi prze router, służy eleiminacji zapętleń w sieci IP.
d. Numer identyfikujący protokół warswy wyższej znajdujący się w polu danych.

Pakiet IPv6:



a. Tak samo jak pole „Protokół" w nagłówku IPv4.
Jednak w odróżnieniu od IPv4 pole to może wskazywać nie tylko nagłówek warstwy wyższej, ale także opcjonalny, dodatkowy nagłówek IPv6, zastępujący informacje podawane w polu opcji IPv4. W takim wypadku posiada on także swoje pole „Next Header", itd., aż do momentu dotarcia do nagłówka warstwy transportowej.
b. Pełni rolę taką jak *Time To Live*.
c. Zależnie od wartości pola „Next Header" mogą to być kolejne opcjonalne nagłówki dodatkowe IPv6 i następnie „dane warstwy wyższej" lub od razu „dane warstwy wyższej".

Rysunek 2: Struktura pakietów IP

ny jest jako liczba dziesiętna rozdzielana kropką od pozostałych. Adresy IPv6 zapisywane są zazwyczaj w notacji dwukropokowej, polegającej na zapisywaniu 16 bitowych części adresu liczbami szesnastkowymi oddzielanymi dwukropkiem, dodatkowo jeden ciąg zer (o długości będącej wielokrotnością 16 bitów) może być skompresowany (pominięty) co daje w zapisie dwa dwukropki `::`.

1.3.1 Długość prefixu i maska

Adresy hostów grupuje się w adresy sieci, bazując na jednakowym (bitowo) początku takiego adresu (zwanym adresem sieci lub prefixem). Ilość bitów stanowiących adres sieci w danym adresie IP nazywana jest długością prefixu i zapisywana jest zazwyczaj po ukośniku[1]. Na przykład zapis `2001:db8::a17/48` oznacza że pierwsze 48 bity stanowią adres sieci a kolejne $128 - 48 = 80$ bitów stanowi adres hosta w tej sieci.

Długość prefixu jednoznacznie określa maskę danej podsieci, czyli liczbę odpowiadającą długości adresu (32 bity lub 128 bitów), złożoną z ciągu jedynek o długości prefixu oraz ciągu zer (o długości adresu hosta). W przypadku IPv4 spotykane jest także podawanie maski sieci w notacji kropkowo-dziesiętnej zamiast długości prefixu.

Przykład

```
adres IPv4 zapisany z informacją o długości prefixu: 10.23.45.56/13, czyli:
adres: 10.23.45.56 = 00001010000101110010110100111000
maska: 255.248.0.0 = 11111111111110000000000000000000
                     prefix sieci adres hosta w sieci
adres sieci        = 00001010000101000000000000000000 = 10.16.0.0

Adres sieci obliczany jest jako bitowy AND pomiędzy adresem i maską.

Maska zawsze jest złożona z ciągu samych bitów o wartości 1 a następnie o wartości 0.
Bitów o wartości 1 jest tyle ile wynosi długość prefixu (podawana po /), czyli w tym przykładzie 13.

W IPv6 działa to analogicznie, tyle że adres ma 128 bitów długości, stosuje się notacje dwukropkową
zamiast kropkowo-dziesiętnej i nie stosuje się jawnego zapisu maski (a jedynie długość prefixu).
```

Sieć może zostać podzielona na mniejsze sieci (z większą wartością prefixu), jak też grupa sieci może zostać zagregowana w jedną większą (2^n raza) sieć (z prefixem mniejszym o n). Agregacja hostów i sieci w większe całości jest wykorzystywana w mechanizmach routingu, co pozwala na redukcję wielkości tablic routingowych.

1. Jest to notacja *CIDR*. Przed wprowadzeniem tego mechanizmu w IPv4 funkcjonował klasowy sposób routingu (*classful*), gdzie wielkość maski była deteminowana wartością pierwszych bitów adresu – ale to już historia.

1.3.2 Przynależność do sieci

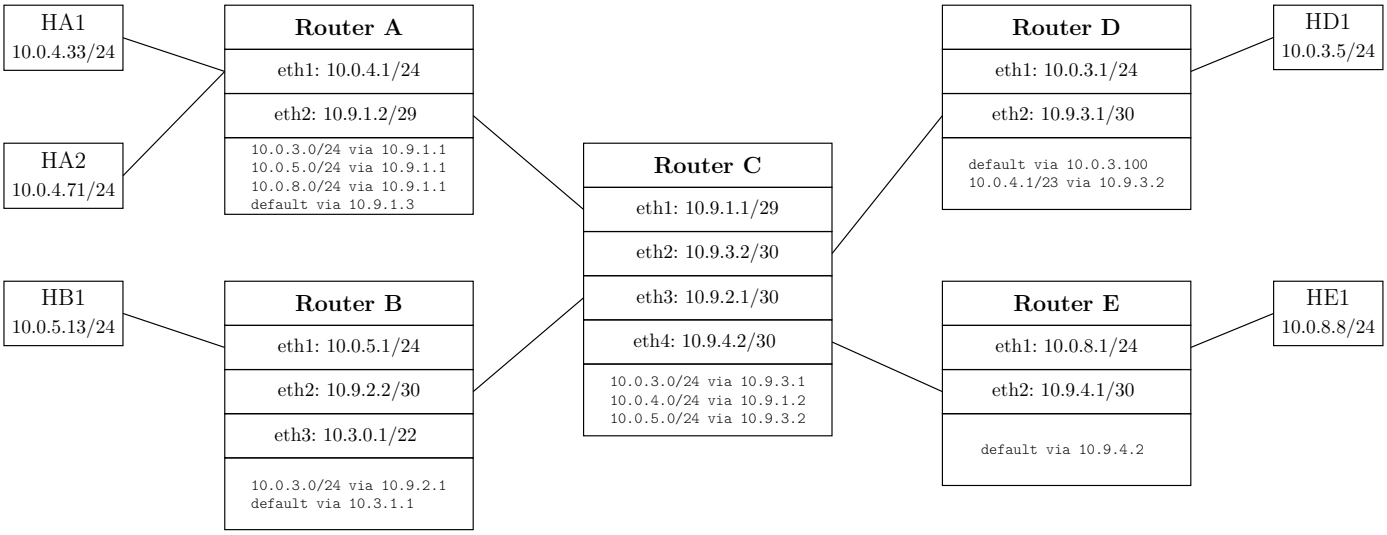
Adres sieci zapisuje się typowo z wyzerowanymi bitami stanowiącymi adres hosta (czyli po dokonaniu bitowego *and* z maską danej sieci) oraz podaną informacją o długości prefixu, dla powyższego przykładu będzie to `2001:db8::/48`. Informacja taka jest wystarczająca do sprawdzenia czy dowolny inny adres IP należy do tej sieci czy nie.

```
from ipaddress import *

# adres

adr     = ip_interface("2001:0db8::17:15")
adr_int = int(adr.ip)
print("Adress IPv6 jest 128 bitową liczbą całkowitą np.:")
print("  " + str(adr.ip) + " == " + hex(adr_int) + "\n")

# sieć - maska i długość prefixu

net         = ip_interface("::/112");
netmask     = net.network.netmask
netmask_int = int(netmask)
net_preflen = net.network.prefixlen

print("Maska podsieci IPv6 jest 128 bitową liczbą całkowitą np.:")
print("  " + str(netmask) + " == " + hex(netmask_int) + "\n")
print("Jako że maska jest liczbą, która zapisana binarnie, zawsze zawiera ciągły ciąg bitów")
print("o wartości 1, a po nim ciągły ciąg bitów o wartości 0 (mogą być zerowej długości), to")
print("często stosowany jest zapis polegający na podawaniu długości prefiksu: /" + str(net_preflen))
print("jest to ilość bitów o wartości 1 w masce, czyli im większy prefix tym mniejsza sieć.\n")

# adres w sieci

adr2     = ip_interface("2001:0db8::17:15/112");
net2     = adr2.network
net2_int = int(net2.network_address)

print("Aby obliczyć adres sieci (czyli wspólną dla wszystkich hostów w danej sieci część")
print("adresu IP) należy wykonać binarny AND pomiędzy adresem IP hosta a maską podsieci.")
print("Dla powyższego przykładu:")
print("  " + hex(netmask_int & adr_int) + " == " + str(net2) + " == " + hex(net2_int) + "\n")

# aby sprawdzić czy adres IP należy do danej sieci trzeba obliczyć adres sieci tego hosta
# w oparciu o maskę sieci którą sprawdzamy
def sprawdzSiec(n, a):
        nn = int(a) & int(n.netmask)
        if nn == int(n.network_address):
                print(str(a) + " należy do sieci " + str(n))
        else:
                print(str(a) + " NIE należy do sieci " + str(n))

sprawdzSiec(net2, ip_interface("2001:0db8::17:ab13").ip)
sprawdzSiec(net2, ip_interface("2001:0db8::13:a").ip)
```

Zadanie 1.3.1

Ustal czy adres `2001:db8:0:a17::123` należy do sieci `2001:db8::/48`. Możesz posłużyć się narzędziami do obliczania zakresów sieci IP (np. `sipcalc`) lub obliczyć to ręcznie.

1.4 Routing



Uwaga: W tablicach routingu na ilustracji pominięto wpisy związane z pokazanymi interfejsami i ich adresami IP. Pominięto też tablice routingowe hostów H*, należy zakładać że oprócz wpisu związanego z adresem IP ustawionym na interfejsie mają one jedynie wpis default wskazujący na „ich" router np. tablica HA1 ma dwa wpisy: `default via 10.0.4.1` i `10.0.4.0/24 dev eth0`.

Rysunek 3: Przykładowe sieci wraz z trasami routingu.

Router kieruje każdy z pakietów do kolejnego routera lub bezpośrednio do hosta docelowego na podstawie jego adresu docelowego i tablicy routingu. Tablica taka zawiera adresy sieci wraz z adresami następnych routerów do nich prowadzących bądź wskazaniem lokalnego interfejsu sieciowego poprzez który powinny być osiągalne hosty z danej sieci. W tym celu korzysta z sprawdzania przynależności adresu do sieci, w celu ustalenia adresu następnego routera i/lub interfejsu sieciowego na który ma zostać przekazany pakiet.

Tablica przeglądana jest od wpisów najbardziej precyzyjnych, czyli z największym prefixem do wpisów najbardziej ogólnych (ostatnim wpisem jest na ogół trasa domyślna czyli sieć ::/0 dla IPv6 lub 0.0.0.0/0 dla IPv4). Dzięki czemu jeżeli kilka wpisów (sieci) z tablicy routingu pasuje do adresu docelowego z nagłówka pakietu, wybierany jest wpis najbardziej precyzyjny (o najdłuższym prefixie), a pasująca do każdego adresu trasa domyślna wybierana jest tylko gdy nie ma żadnej lepszej. Może się zdarzyć że kilka wpisów (nawet z tą samą maską) pasuje do adresu docelowego hosta, w takiej sytuacji do wyboru ścieżki używane są inne dane z tablicy routingu (takie jak metryka).

Tablice routingu mogą zawierać wpisy dodawane statycznie (wpisane do konfiguracji danego urządzenia), jak też wpisy dodawane dynamicznie w oparciu o protokołu wymiany informacji routingowych (protokoły routingu) takie jak: IGRP, OSPF, BGP. Protokoły routingu dynamicznego mogą być wykorzystywane m.in. do rozkładania obciążenia na różne łącza, zapewnienia redundancji łącz, blokowania ataków (D)DoS.

Także każdy z hostów ma tablice routingu, typowo składa się z dwóch pozycji – trasy do sieci lokalnej (tej sieci z której adres posiada dany host) wskazującej bezpośrednio na urządzenie sieciowe oraz trasy domyślnej wskazującej na router zapewniający dostęp do innych sieci, nazywany bramką (gateway). Jeżeli router nie posiada adresu w tej samej sieci co host konieczna jest dodatkowa trasa wskazująca poprzez jakie urządzenie dostępny jest router domyślny.

Oprócz opisanego powyżej routingu unicastowego (kierowania do jednego odbiorcy) realizowane są także transmisje:

- *anycast* – do dowolnego / najbliższego hosta o danym adresie; zasadniczo jest to transmisja unicast, tyle że adres docelowy nie jest unikalny w skali globalnej a różne routery kieruje te pakiety do różnych hostów docelowych (typowo wybierając najbliższy taki host)
- *multicast* – do grupy hostów, w tym wypadku (multicastowy) adres IP identyfikuje "kanał nadawczy" a nie unikalny host docelowy
- *broadcast* – do wszystkich hostów (w ramach danej sieci – nie są routowne), transmisje rozgłoszeniowe można traktować jako szczególny przypadek transmisji multicastowych w których grupa multicastowa obejmuje wszystkie hosty (można je zastąpić takimi transmisjami multicastowymi)

Na ilustracji 3 pokazano kilka przykładowych sieci wraz z ustawieniami routerów zapewniających komunikacje między nimi. Przykłady:

- Pakiet wysłany z HA1 do `10.0.3.13`:
  - zostanie przesłany przez *Router A*, który ma wpis `10.0.3.0/24 via 10.9.1.1`, do *Router C*
  - z *Router C*, który ma wpis `10.0.3.0/24 via 10.9.3.1`, zostanie przekazany do *Router D*
  - *Router D* przekaże go do odpowiedniego hosta (HD1) poprzez interfejs eth1
- Pakiet wysłany z HD1 do `10.0.4.33`:
  - zostanie przesłany przez *Router D*, który ma wpis `10.0.4.1/23 via 10.9.3.2`, do *Router C*
  - z *Router C*, który ma wpis `10.0.4.0/24 via 10.9.1.2`, zostanie przekazany do *Router A*
  - *Router A* przekaże go do odpowiedniego hosta (HA1) poprzez interfejs eth1
- Pakiet wysłany z HD1 do `10.0.5.13`:
  - zostanie przesłany przez *Router D*, który ma wpis `10.0.4.1/23 via 10.9.3.2`, do *Router C*
  - z *Router C*, który ma wpis `10.0.5.0/24 via 10.9.3.2`, zostanie przekazany do *Router B*
  - *Router B* przekaże go do odpowiedniego hosta (HB1) poprzez interfejs eth1

Zwróć uwagę że:

- aby możliwa była komunikacja trasa routingowa musi być:
  - skonfigurowana w obie strony (tak jak ma to miejsce między HA1 i HD1)
    - z tego powodu HE1 nie może komunikować się np. z HD1
  - skonfigurowana zarówno na routerach brzegowych (takich jak A i D), jak i wszystkich routerach pośrednich (takich jak C)
- *Router D* posiada zagregowany wpis w tablicy routingu obejmujący zarówno sieci podłączone do *Router A*, jak i do *Router B*, ale np. wpisy w *Router A* nie są i nie mogą być zagregowane

Polecenia wyświetlające trasy routingowe mogą wypisywać więcej informacji niż w powyższym przykładzie. Na przykład (jest wynik `ip r`, pokolorowany celem ilustracji, w innych, czy przy użyciu różnych opcji poleceniach może to wyglądać inaczej):

```
default via 213.135.60.1 dev eth-pub
192.168.0.0/23 via 10.0.1.100 dev eth-rsc
10.0.1.0/24 dev eth-rsc  proto kernel  scope link  src 10.0.1.13
```

W poszczególnych wpisach są to m.in.:

- sieć docelowa, default oznacza to samo co 0.0.0.0/0 lub ::/0
- via x.x.x.x – adres routera do którego ma trafić pakiet
- dev ... – urządzenie którym ma zostać wysłany
- src x.x.x.x – adres który ma zostać użyty jako adres nadacy

Ogólnie wpisy z via x.x.x.x oznaczają przesłanie do innego routera a wpisy bez tej informacji bezpośrednie wysłanie do hosta docelowego na wskazanym interfejsie.

Zadanie 1.4.1

Wynik polecenia `ip -6 r` pokazującego tablicę rotingową wygląda następująco:

```
2001:db8:0:21::/64 dev eth1  proto kernel  metric 256
2001:db8::/32 via 2001:db8:0:21::100 dev eth2  metric 1024
2001:db8:fff:21::/64 dev eth2  proto kernel  metric 256
2001:db8:abc:21::/64 via 2001:db8:fff:21::1 dev eth2  metric 1024
fe80::/64 dev eth1  proto kernel  metric 256
fe80::/64 dev eth2  proto kernel  metric 256
default via 2001:db8:0:21::1 dev eth1  metric 1024
```

Ustal gdzie zostanie skierowany pakiet adresowany do `2001:db8:abc:21:123::ff3`.

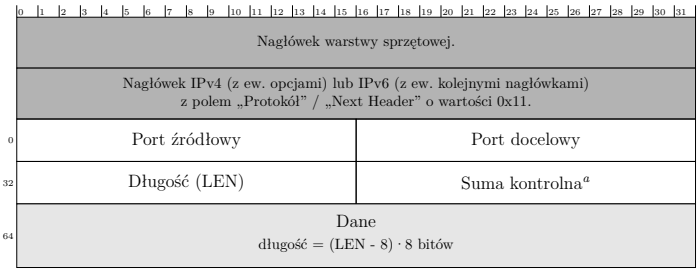
2 Komunikacja TCP/IP

W oparciu o protokół IP działają protokoły warstwy transportowej takie jak UDP, TCP, czy też (mniej znany protokół dedykowany dla strumieniowych transmisji czasu rzeczywistego) SCTP.

Jednym z zadań tych protokołów jest identyfikowanie usługi (procesu) w ramach systemu posiadającego dany adres IP, do którego mają trafić dane. W tym celu zarówno UDP jak i TCP na każdym z hostów wyróżniają numeryczny identyfikator dla aplikacji/procesu/usługi będącego odbiorcą czy też nadawcą informacji zwany **numerem portu**.

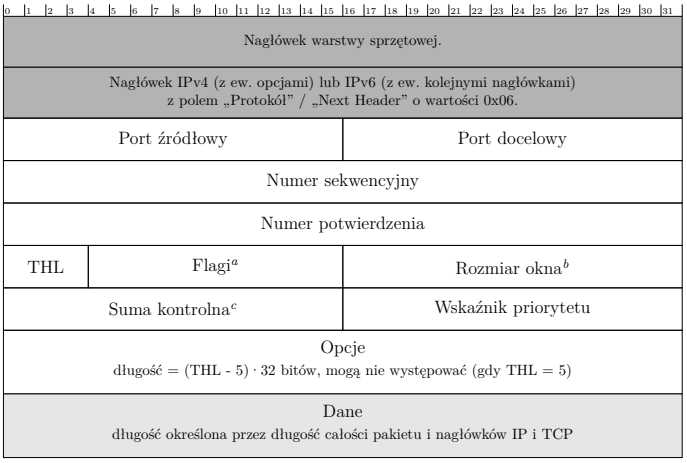
Najprostszym protokołem warstwy transmisji wydaje się być UDP, protokół ten umożliwia przesłanie informacji pomiędzy dwoma hostami IP i nie kontroluje on tego czy została ona przesłana poprawnie. Natomiast TCP, w odróżnieniu od UDP, kontroluje to czy przesłana informacja dotarła do adresata i nie została uszkodzona, a w przypadku problemów informacja wysyłana jest ponownie. TCP w związku z tym w przeciwieństwie do UDP musi otworzyć połączenie i wykorzystywać je do kontroli poprawności przesłania informacji, wymaga zatem przesłania większej liczby pakietów (co może prowadzić do pewnych opóźnień itp). W związku z tym TCP używany jest tam gdzie konieczna jest kontrola poprawności transmisji (oraz ponowne wysłanie zgubionego pakietu), UDP tam gdzie nie jest to potrzebne (a liczy się czas). Z połączeniem występującym w protokole TCP związane jest także pojęcie wielkości okna, czyli tego co ile (tysięcy) bajtów odbiorca musi potwierdzać odbiór pakietów. Wielkość ta jest dynamicznie dostosowywana do parametrów łącza, pozwalając na sterowanie przepływem - jeżeli wysycana jest dostępna przepustowość łącza, dochodzi do strat pakietów i wielkość okna jest zmniejszana.

Datagram UDP:



a. Uwzględnia też wybrane pola z nagłówków IPv4 lub IPv6.

Pakiet TCP:



a. 3 bity rezerwy i flagi związane ze stanem połączenia TCP.
b. Informuje o ilości danych które odbiorca może aktualnie przyjąć.
c. Uwzględnia też wybrane pola z nagłówków IPv4 lub IPv6.

Rysunek 4: Struktura pakietów UDP i TCP

2.1 Popularne usługi

W ramach sieci mogą być realizowane różne usługi w oparciu o różne protokoły warstwy aplikacyjnej. Standardowe usługi posiadają zdefiniowane domyślne adresy portów dla swoich protokołów. Wśród usług i protokołów sieciowych należy wymienić przynajmniej:

- DNS (Domain Name System) - odpowiedzialny za system mapujący nazwy alfanumeryczne hostów na adresy IP.
- mechanizmy auto konfiguracji hostów - DHCP, rozgłaszanie informacji routingowej poprzez ICMPv6 (protokół warstwy 3)
- WWW - udostępnianie treści z użyciem protokołu HTTP
- pocztę elektroniczną - przesyłanie wiadomości (protokoły SMTP, IMAP, POP)
- komunikację natychmiastową i telefonię IP (protokoły SIP, XMPP, IAX)
- SSH - zdalny, szyfrowany dostęp do systemów IT, przesył plików oraz tunelowanie innych usług

2.1.1 Domain Name System

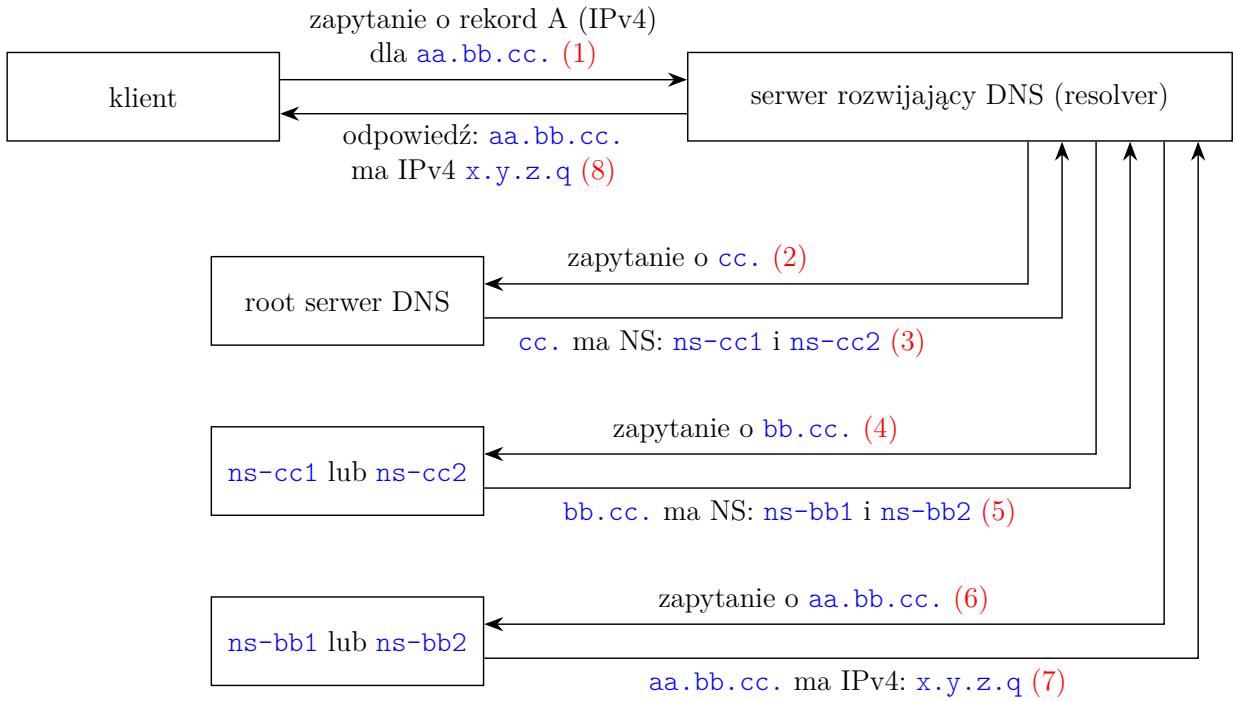
DNS umożliwia mapowanie nazwy na adres IP (lub wiele adresów IP) oraz przechowywanie dodatkowych informacji na temat domeny i znajdujących się w niej usług.

Domeny posiadają budowę hierarchiczną / drzewiastą:

- precyzja rośnie od prawej do lewej
- kolejne poziomy oddzielane są kropkami
- najwyższym poziomem jest kropka będąca ostatnim znakiem w pełnej nazwie domenowej (np. `ciekawi.icm.edu.pl.`), którą najczęściej pomija się w zapisie
- hierarchia ta jest niezależna od hierarchii routingu i wynika z faktu posiadania/użytkowania danej (pod)domeny

Realizacja odpowiedzi na zapytanie DNS wygląda następująco:

1. host kieruje zapytanie do określonego w jego konfiguracji serwera "rozwijającego" DNS (DNS resolver),
2. serwer taki sprawdza w swojej pamięci podręcznej czy zna odpowiedź na to zapytanie (i nie jest ona przeterminowana - nie upłynął czas TTL od odnalezienia), jeżeli nie ma jej w swojej pamięci to
3. serwer taki zna adresy głównych serwerów DNS (root serwerów) zawierających informacje na temat serwerów obsługujących domeny najwyższego rzędu i kieruje do jednego z nich zapytanie o serwer obsługujący skrajnie prawą część adresu (np. *.org*),
4. do otrzymanego serwera kierowane jest zapytanie o większą część adresu (np. *eu.org*),
5. itd. aż do uzyskania odpowiedzi o pytany adres



(X) oznacza kolejność wykonywanych operacji. Przed wykonaniem zapytania *resolver* sprawdza czy nie posiada w swoim cache zapamiętanej (i nie przeterminowanej) odpowiedzi na to zapytanie.

Rysunek 5: Realizacja zapytania o rekord DNS

DNS przechowuje informacje w postaci rekordów mających określony typ (w większości przypadków dla danej nazwy domenowej może być zdefiniowanych wiele rekordów, tego samego lub innych typów). Wśród najważniejszych typów rekordów należy wymienić:

- `NS` – informacja o serwerach obsługujących DNS danej domeny

- `A` – mapowanie nazwy na adres IPv4
- `AAAA` – mapowanie nazwy na adres IPv6
- `MX` – informacja o serwerach obsługujących pocztę danej domeny
- `SRV` – informacje o hoście świadczącym usługę w tej domenie (usługa określana jest w nazwie domeny o którą pytamy)
- `PTR` – mapowanie adresów IP na nazwy domenowe, realizowane w specjalnym drzewie `in-addr.arpa` (dla IPv4) lub `ip6.arpa` (IPv6), gdzie adres IP zapisywany jest w odwróconej kolejności po bajcie dla IPv4 lub cyfrze szesnastkowej dla IPv6
- `TXT` – informacje dodatkowe (np. jakie serwery pocztowe, są upoważnione do wysyłania poczty z tej domeny)
- `SOA` – informacje podstawowe o strefie opisującej domenę
- `CNAME` – alias na inną domenę (domena którą aliasujemy nie może mieć innych wpisów, nawet SOA)

2.1.2 Standardowe numery portów

Popularne usługi (np. www) posiadają ustalone standardowe numery portów na których nasłuchiwac będzie serwer takiej usługi (np. dla wspomnainego www jest to port 80). Informacja o numerze portu usługi może być umieszczona także w rekordzie SRV systemu DNS.

3 Diagnostyka sieci

Istnieje wiele poleceń służących do diagnozowania ewentualnych problemów sieciowych lub mogących być w tym przydatnymi. Poniżej znajduje się zestawienie najbardziej popularnych / użytecznych narzędzi z podziałem wg zastosowań.

3.1 Adresy

- `ipcalc` oraz `sipcalc` – kalkulator IP (pozwalający na obliczanie adresów sieci rozgłoszeniowych, zmianę notacji itd)
- `whois` – informacje z bazy whois (o domenie lub adresie IP)

3.2 Dostępność i trasy do hostów

- `ping [opcje] host` lub `ping6 [opcje] host` – sprawdzanie dostępności hosta z użyciem protokołu ICMP (obecnie komenda `ping6` najczęściej jest równoważna poleceniu ping z opcją `-6` wymuszającą używanie jedynie IPv6, na starszych systemach komenda ping może nie obsługiwać adresów IPv6 i wtedy konieczne jest stosowanie do nich polecenia `ping6`), ważniejsze opcje:
  `-c n` wykonaj n zapytań (domyślnie pyta do momentu przerwania przy pomocy np. Ctrl-C, lub sygnału wysłanego z uzyciem komendy `kill`)
  `-n` nie zamieniaj adresu IP hosta który odpowiedział na nazwę domenową
- `traceroute`, `traceroute6`, `tracepath`, `tracepath6`, `tcptraceroute` lub `tcptraceroute6` – sprawdzanie ścieżki do hosta (wypisanie listy routerów przez które przechodzi pakiet w drodze do wskazanego hosta)
  Istnieją różne warianty tych poleceń (nawet pod tą samą nazwą), różnią się one stosowanymi mechanizmami i domyślnymi opcjami. Generalnie wszystkie uruchamia się na zasadzie `polecenie [opcje] host`. Warianty z `6` na końcu nazwy będą używały jedynie adresów IPv6, natomiast polecenia bez `6` na końcu nazwy mogą potrafić ich używać lub nie. Wszystkie popularne warianty pozwalają na podanie opcji `-n` wyłączającej zamienianie adresu IP hosta który odpowiedział na jego nazwę domenową.
  Może zdarzyć się że śledzenie urwie się na jakimś hoście (np. z powodu jego konfiguracji lub błędów

w jego oprogramowaniu sieciowym), może się zdarzyć że przy użyciu innej komendy z tej grupy (lub zmianie opcji) uda się prześledzić dalszą trasę pakietu.

- `mtr [opcje] host` – sprawdzanie ścieżki do hosta (czyli podobnie jak traceroute i tracepath) w trybie ciągłym (z ciągłym odświeżaniem) wraz z wypisywaniem informacji o stratach pakietów i opóźnieniach na poszczególnych odcinkach, ważniejsze opcje:
  `-n` nie zamieniaj adresu IP hosta który odpowiedział na nazwę domenową
- `nmap` – skaner sieciowy - sprawdzanie dostępnych hostów w sieci, otwartych portów, itd
- `arping` – narzędzie do pingowania z wykorzystaniem zapytań ARP zamiast ICMP
  istnieją dwie zasadnicze odmiany: z iputils oraz z synscan; ta druga zawarta w debianowym pakiecie "arping" umożliwia także pingowanie po adresie MAC (ale nie przez RARP, bo on nie do tego służy), aby to jednak działało host docelowy nie może ignorować pingów rozgłoszeniowych, metoda obejścia opisana jest w README arping'a
- `arp-scan` – wyszukiwanie hostów w oparciu o zapytania ARP (można powiedzieć że jest to równoważne uruchamianiu komendy arping w pętli)

3.3 DNS

- `dig [opcje] nazwa [typRekordu]` – narzędzia do uzyskania informacji z DNS, pozwala na określenie poprzez `@adres` serwera który chcemy odpytać oraz na określenie (poprzez drugi argument) typu rekordu który chcemy uzyskać, zamiast typu rekordu można podać: `ANY` (powoduje odpytanie o wszystkie rekordy) lub `AXFR` (powoduje wysłanie prośby o transfer całej strefy, działa jeżeli dany host ma prawo transferu całej strefy z danego serwera)
- `host [opcje] nazwa|ip [server]` – narzędzia do zamiany adresów domenowych na IP i odwrotnie oraz wyciągania innych informacji z DNS (np. rekordy MX)
- `nslookup [opcje] nazwa|ip [server]` – narzędzia do zamiany adresów domenowych na IP i odwrotnie oraz wyciągania innych informacji z DNS (np. rekordy MX)
- `dnstracer` – śledzenie trasy zapytań DNS
- `dnswalk` – debuger DNS

3.4 IPv6

- `ndisc6` – testowanie ICMPv6 Neighbor Discovery
- `rdisc6` – testowanie ICMPv6 Router Discovery
- `rltraceroute6` – trasa pakietów do danego hosta IPv6 z użyciem UDP/ICMP
- `tcpspray6` – pomiar prędkości łącza z użyciem TCP/IP Discard/Echo
- `na6` / `ns6` – wysyłanie pakietów Neighbor Advertisement / Solicitation
- `ra6` / `rs6` – wysyłanie pakietów Router Advertisement / Solicitation
- `ni6` / `rd6` – wysyłanie pakietów ICMPv6 Node Information / Redirect
- `scan6` – skanowanie sieci IPv6

3.5 debugowanie łączności sieciowej

- `netcat` lub `nc` lub `netcat6` – program pozwalający na wysyłanie pakietów TCP i UDP z definiowaną przez nas zawartością, oraz odbiór pakietów TCP i UDP (słuchanie na wskazanym porcie), umożliwia m.in. testowanie usług sieciowych (takich jak smtp, www, jabber, ...); uwaga: występuje w kilku wersjach różniących się opcjami
- `telnet` – program umożliwiający zdalny (nieszyfrowany, łącznie z hasłem!) dostęp do powłoki, a także (podobnie jak netcat) m.in. testowanie usług sieciowych
- `swaks` – narzędzie do testowania SMTP

- `tcpdump` – przechwytuje komunikację sieciową celem analizy nagłówków lub pełnej zawartości pakietów (wsparcie dla niektórych z protokołów warst wyższych wymaga doinstalowania - np. obsługę DHCP zapewnia dhcpdump)
- `wireshark` lub `tshark` – przechwytuje komunikację sieciową celem analizy nagłówków lub pełnej zawartości pakietów, wspiera dekodowanie wielu protokołów warstwy aplikacyjnej, wireshark posiada graficzny interfejs użytkownika, tshark jest wersją konsolową

3.6 informacje o wykorzystaniu i prędkości sieci

- `netstat` – informacje o sieci (np. `netstat -l46np | sort -t / -k 2` wypisze informację o nasłuchujących (po IPv4 lub IPv6) usługach posortowane po nazwie procesu)
- `iptraf` – monitor IP LAN
- `nload` – graficzne (ncurses) pokazywanie wykorzystania (prędkości) interfejsów sieciowych
- `ttcp` – testuje prędkość połączenia sieciowego (strona domowa, najnowsza wersja oraz mutacja)
- `iperf` – pomiar prędkości połączenia sieciowego

Zadanie 3.0.1

Korzystając z narzędzi służących do diagnozowania sieci sprawdź czy host `ciekawi.icm.edu.pl` jest dostępny. Jakiego polecenia użyłeś(aś) w tym celu? Co jeszcze mówi wynik tego polecenia?

Zadanie 3.0.2

Korzystając z narzędzi służących do diagnozowania sieci ustal jaką trasą podróżują pakiety z Twojego komputera do `www.opcode.eu.org` oraz do `www.example.org`. Jakiego lub jakich poleceń użyłeś(aś) w tym celu? Co jeszcze mówi ich wynik? Co możesz powiedzieć porównując uzyskane trasy?

Zadanie 3.0.3

Ustal (wszystkie) adresy IPv4 i IPv6 serwera `www.bitbucket.org`. Zastanów się czemu może służyć to że niektóre nazwy domenowe rozwijają się na wiele różnych adresów IP.

Zadanie 3.0.4

Korzystając z dwóch instancji programu `nc` (`netcat`) – jednej w roli serwera, drugiej w roli klienta prześlij między nimi jakieś dane. Użyj programu `tcpdump` (z odpowiednimi opcjami) aby podsłuchać komunikację sieciową między tymi programami i zobaczyć przesyłane dane.

Zadanie 3.0.5

Korzystając bezpośrednio z poleceń protokołu HTTP i programu `nc` (`netcat`) lub `telnet`, pobierz i wyświetl kod strony `http://www.opcode.eu.org/`.
Wskazówka: Opis protokołu HTTP odnajdziesz bez problemu w sieci.
Ogólnie żądanie HTTP składa się z pierwszej linii określającej typ wykonywanej operacji, ścieżkę oraz wersję protokołu - np. ***`GET /abc.txt HTTP/1.1`*** *oznacza prośbę o zwrócenie zawartości pliku* ***`/abc.txt`****. Następnie podawane są nagłówki, w wersji HTTP 1.1 obowiązkowy jest nagłówek „Host" określający nazwę domenową serwera - np.* ***`Host: www.example.org`****. Po nagłówkach występuje pusta linia po której mogą być przesłane (przy niektórych typach żądań) jakieś dane (np. z wypełnionego na stronie formularza).*

Zadanie 3.0.6

Zadanie 3.0.5 można rozwiązać przy pomocy netcat'a bez dodatkowych opcji, jednak jeżeli stroną do pobrania byłoby np. `http://www.icm.edu.pl` to należałoby skorzystać z opcji `-C` netcat'a (w przeciwnym razie serwer zwraca błąd 400 "Bad Request"). Sprawdź co robi ta opcja i zastanów się dlaczego w przypadku niektórych serwerów jest konieczna a w przypadku innych nie? Co na ten temat mówi standard HTTP?

Zadanie 3.0.7

Zobacz czy rozwiązanie zadania 3.0.1 zadziała gdy użyjesz nazwy serwera zawierającej polskie znaki: `licealiści.icm.edu.pl`. Jak myślisz, dlaczego polskie znaki są tak rzadko używane w nazwach domenowych?

4 Ethernet

Od strony sprzętowej sieć składa z:

- hostów stanowiących nadawców i odbiorców informacji
- urządzeń sieciowych pośredniczących w ich przekazywaniu, takich jak nadajniki, switche, mediakonwertery
- okablowania miedzianego bądź światłowodowego (jeżeli nie jest siecią bezprzewodową)

W przypadku sieci w standardzie Ethernet stosowane są 48 bitowe adresy MAC (pierwsza część identyfikuje producenta karty) oraz wspólny dla wszystkich odmian (przewodowych i bezprzewodowych) format ramki (określający położenie w ramce adresów, informacji dodatkowych oraz danych). Pakiety protokołu warstwy wyższej (np. pakiety IP wraz ich strukturą zawierającą adresy itd) z punktu widzenia ramki ethernetowej są danymi, w które ta warstwa nie wnika. Do mapowania adresów IP na adresy MAC wykorzystywany jest protokół ARP (dla IPv4) lub Neighbor Discovery (dla IPv6) - odbywa się to poprzez wysłanie ramki ethernetowej na adres rozgłoszeniowy (odbierany przez wszystkie hosty) z pytaniem o to jaki MAC adres ma host o podanym numerze IP.

Sieć ethernetowa typowo posiada strukturę wielokrotnej gwiazdy (drzewa), w węzłach której stosowane są switche. Kierują one ramki do odpowiednich gałęzi na podstawie adresu docelowego i wpisów w tablicy adresów MAC, utworzonej w oparciu o źródłowe nadawców przechodzących przez dany switch ramek. W przypadku gdy adresu docelowego nie ma w tablicy ramka kierowana jest na wszystkie porty switcha z wyjątkiem tego na którym została odebrana. W taki sposób zawsze są też przesyłane ramki wysyłane na adres rozgłoszeniowy (bradcast).

W przewodowych sieciach Ethernet wykrywaniem zajętości medium transmisyjnego oraz wykrywaniem kolizji zajmuje się protokół CSMA/CD (Carrier Sense Multiple Access with Collision Detection - wielodostęp z rozpoznawaniem stanu kanału oraz wykrywaniem kolizji). Przed rozpoczęciem nadawania stacja musi sprawdzić czy medium jest wolne, jeżeli tak może zacząć nadawać, jeżeli dwie stacje zaczną nadawać równocześnie zostaje to wykryte, obie przerywają nadawanie i wznawiają po losowym czasie. Jednak ze względu na stosowanie głównie połączeń punkt-punkt full-duplex (osobne przewody do nadawania i osobne do odbioru), co ogranicza tzw. domenę kolizji do pojedynczego hosta, protokół ten nie odgrywa tutaj szczególnie istotnej roli.

4.1 Ramka

Przyglądając się przedstawionej ramce ethernetowej warto zauważyć iż długość przesyłanych w niej danych musi zawierać się w zakresie od 46 do 1500 bajtów. Ograniczenia te wynikają ze specyfikacji elektrycznej ethernetu i protokołu CSMA/CD. Dolne ograniczenie łatwo jest wyeliminować dopełniając pole dane jakimiś wartościami. Jednak górne ogranicza nam maksymalną długość pakietu IP przesyłanego taką ramką, wyznaczając wartość MTU (Maximum Transmission Unit). W różnych sieciach L2, wartość MTU może być różna. Nawet w Ethernecie istnieje możliwość zwiększenia MTU dla danej sieci (jeżeli sprzęt

	0 1 2 3 4 5 6 7	8 9 10 11 12 13	14 15 16 17 18 19	20 21	....		
	Preambuła + SFD	Adres docelowy	Adres źródłowy	Typ	Zawartość	Suma Kontrolna	Przerwa międzypakietowa
długość (bity)	64	48	48	16	$46 \cdot 8 - 1500 \cdot 8$	32	96
		ramka L2					

Preambuła wraz z SFD (start frame delimiter) i przerwa międzypakietowa są transmitowane na kablu (są elementem ramki L1), ale nie wchodzą w skład ramki L2.

Typ identyfikuje zawartość przenoszoną w pakiecie (gdy 0x600) lub określa długość pakietu.
Jeżeli wynosi 0x8100 to ramka należy do tagowanego VLANu i ma postać:

	Preambuła + SFD	Adres docelowy	Adres źródłowy	0x81 0x00	Tag	Typ	Zawartość	Suma Kontrolna	Przerwa międzypakietowa
długość (bity)	64	48	48	16	16	16	$46 \cdot 8 - 1500 \cdot 8$	32	96

Gdzie Typ identyfikuje właściwą zawartość przenoszoną w pakiecie (gdy 0x600) lub określa długość pakietu.
Natomiast Tag zawiera m.in. 12 bitowy numer VLANu 802.1q
W analogiczny sposób mogą zostać dodane kolejne tagi używane w standardzie IEEE 802.1ad.

w tej sieci wspierają większe ramki – *jumbo frames*), jak również zmniejszenia MTU na danym interfejsie (np. gdy ruch z niego będzie przesyłany dalej łączem o mniejszym MTU).

Problem niedopasowania pakietu IP do MTU można rozwiązać na dwa sposoby – nie generować pakietów większych niż najmniejsze MTU na trasie (mechanizm detekcji MTU) lub dzielić większe pakiety na mniejsze w trakcie przesyłu (mechanizm fragmentacji) . Można się spotkać z obydwoma tymi podejściami, aczkolwiek raczej unika się fragmentacji pakietów.

4.2 VLANy, bonding, ...

Ethernet pozwala na wirtualne podzielenie pojedynczej sieci lokalnej na wiele niezależnych (nie komunikujących się ze sobą w warstwie ethernetu) sieci, nazywanych VLAN. Działanie tego mechanizmu opiera się na zastosowaniu zarządzalnych switchy, które programowo mogą być dzielone na części zapewniające separację ruchu poszczególnych VLANów. Ponadto wybrane porty takiego switcha mogą być przypisane do różnych części (celem udostępnienia do innego switcha lub hosta kilku sieci wirtualnych), w takim przypadku do ramek ethernetowych wysyłanych tym portem dodawana jest informacja do którego VLANu należą (dwu bajtowy numer), a w przypadku ramek otrzymywanych na podstawie tego numeru odbywa się ich kierowanie do odpowiedniej "części" przełącznika (mówimy o VLANach tagowanych). Możliwe jest aby jeden wybrany VLAN na takim porcie był nie tagowany (do jego ramek nie będzie dodawany numer, a otrzymywane pakiety bez numeru będą kierowane do niego.

Ethernet pozwala również na grupowanie kilku portów w jeden port wirtualny (tzw port trunking / bonding) celem zwiększenia przepustowości lub niezawodności łącza. A dzięki zastosowaniu w różnych typach sieci ethernet tego samego formatu ramki możliwe jest też stosunkowo proste zmienianie medium transmisyjnego (np. z kabla miedzianego na światłowód) z użyciem media-konwerterów.

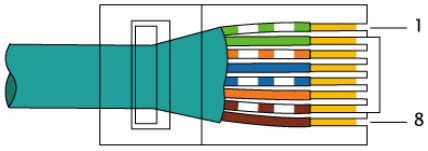
4.3 Kable

Sieć ethernetowa wykorzystuje 8 żyłowe kable złożone z 4 par. Najpopularniejszym przewodem jest kabel UTP kategorii 5e, czyli nieekranowana skrętka pozwalająca na pracę z częstotliwością 100 MHz. W przypadku instalacji okablowania strukturalnego często stosowane są wyższe kategorie okablowania a także kable dodatkowo ekranowane. Ekran może obejmować osobno każdą parę, jak też może być wspólny dla całego przewodu, może być wykonany z folii lub siatki. Np. SF/FTP oznacza kabel z ekranem z siatki i folii (SF/), gdzie dodatkowo każda para jest ekranowana folią (FTP).

W ramach poszczególnych par realizowana jest transmisja różnicowa, czyli istotne jest napięcie pomiędzy przewodami w parze, a nie napięcie na danym przewodzie (w odniesieniu do jakiegoś zewnętrznego poziomu odniesienia). Standard 100Mb/s (dokładniej 100BASE-TX) wykorzystuje jedynie dwie pary przewodów, standard 1Gb/s (1000BASE-T) wykorzystuje wszystkie 4 pary przewodów. Długość kabla pomiędzy dwoma urządzeniami nie powinna przekraczać 100 m. Wykorzystywanie skręconych par przewodów (jeden skręt na 6-10 cm kabla) ma na celu eliminację zakłóceń transmisji - (w uproszczeniu) zakłócenia wchodzą tak samo na oba przewody i różnica miedzy nimi nie zmienia się.

Kable zakańczane są gniazdami bądź wtykami typu RJ-45 montowanymi według jednego z dwóch schematów kolorystycznych: EIA/TIA 568A lub 568B. Pierwotnie (dla sieci 100Mb/s lub starszych) użycie różnych standardów na obu końcach kabla służyło stworzeniu kabla skrosowanego[2].

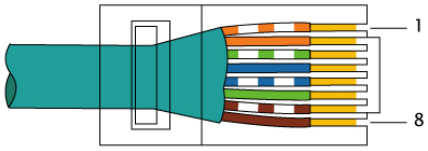
EIA/TIA 568A



Kolejność przewodów we wtyczce/gnieździe:

1. biało-zielony
2. zielony
3. biało-pomarańczowy
4. niebieski
5. biało-niebieski
6. pomarańczowy
7. biało-brązowy
8. brązowy

EIA/TIA 568B



Kolejność przewodów we wtyczce/gnieździe:

1. biało-pomarańczowy
2. pomarańczowy
3. biało-zielony
4. niebieski
5. biało-niebieski
6. zielony
7. biało-brązowy
8. brązowy

Wtyczki RJ-45 są wtyczkami zaciskanymi na przewodzie (bez konieczności odizolowywania żył). Gniazda RJ-45 najczęściej wykonywane są ze złączem typu IDC (Insulation Displacement Connector, KRONE/LSA) służącym do podłączenia przewodu również bez konieczności odizolowywania poszczególnych żył.

5 Konfiguracja sieci w Linuxie

Konfigurację interfejsów sieciowych w systemie Linux umożliwia polecenie `ip`. Przykłady użycia (ta lista w żaden sposób nie wyczerpuje dostępnych możliwości i dodatkowych opcji):

- wyświetlanie i ustawianie adresów IP
  - `ip addr` – wypisuje obecną konfigurację adresów i informacje o stanie interfejsu (`UP`/`DOWN` – interfejs włączony/wyłączony, `LOWER_UP`/`LOWER_DOWN` – link warstwy niższej na interfejsie / jego brak)
  - `ip addr add ADDRESS dev INTERFACE` – dodaje adres `ADDRESS` do interfejsu `INTERFACE`
  - `ip addr del ADDRESS dev INTERFACE` – usuwa adres `ADDRESS` z interfejsu `INTERFACE`
- włączanie i wyłaczanie interfejsów
  - `ip link set INTERFACE up` / `ip link set INTERFACE down` – włączenie / wyłączenie interfejsu `INTERFACE`
  - `ip link set INTERFACE address ADDRESS` – ustawienie adresu sprzętowego urządzenia `INTERFACE` na `ADDRESS`
- konfiguracja tagowanych VLANów
  - `ip link add link INTERFACE name INTERFACE.VLANID type vlan id VLANID` – dodanie interfejsu związanego z tagowanym VLANem o numerze `VLANID` na interfejsie `INTERFACE`, moduł 8021q powinien zostać załadowany automatycznie

2. Połączenie takie przy jednakowych urządzeniach, gdzie nadajnik i odbiornik trafia zawsze na te same piny, zamieniało na kablu nadajnik z odbiornikiem, umożliwiając transmisje między nimi. Aktualnie zdecydowana większość urządzeń obsługuje protokół *Auto MDI-X*, który umożliwia automatyczne ustalenie na których pinach odbywa się nadawanie, a na których odbiór. W rzadkich przypadkach konieczne może być jednak zastosowanie kabla skrosowanego

  - `ip link del INTERFACE.VLANID type vlan` – usunięcie interfejsu INTERFACE.VLANID (związanego z tagowanym VLANem VLANID na interfejsie INTERFACE)

- konfiguracja BRIDGE (programowego switcha)
  - `ip link add INTERFACE type bridge` – dodanie interfejsu bridge'owego o nazwie INTERFACE
  - `ip link set SLAVE master INTERFACE` – włączenie interfejsu SLAVE w skład bridge'owego INTERFACE
  - `ip link set SLAVE nomaster` – wyłaczenie interfejsu SLAVE z bridge'a
  - `ip link show master INTERFACE` – wyświetlanie portów składowych bridge'a o nazwie INTERFACE
  - przydatne może być także polecenie `bridge`
- konfiguracja BONDów (interfejsów agregujących inne w grupę celem zwiększenia prędkości lub niezawodności)
  - `ip link add INTERFACE type bond` – dodanie interfejsu bondingowego o nazwie INTERFACE
  - `ip link set SLAVE master INTERFACE` – włączenie interfejsu SLAVE w skład bondingu INTERFACE
  - `ip link set SLAVE nomaster` - wyłaczenie interfejsu SLAVE z bondingu
  - `ip link show master INTERFACE` – wyświetlanie portów składowych interfejsu bondingowego o nazwie INTERFACE
- konfiguracja routingu
  - `ip [-6] route` – wyświetlanie informacji na temat tras routingowych dla IPv4 (gdy wywołany bez opcji `-6`) / IPv6 (gdy wywołany z opcją `-6`)
  - `ip route add NETWORK via GATEWAY dev INTERFACE` – dodanie trasy routingowej do sieci `NETWORK` poprzez router o adresie `GATEWAY` na interfejsie `INTERFACE`
  - `ip route del NETWORK via GATEWAY dev INTERFACE` – usunięcie trasy routingowej do sieci `NETWORK` ...

Często dostępne są także klasyczne polecenia:

- `ifconfig` włączanie i wyłączanie interfejsów sieciowych (up i down), ustawianie adresu IP i wyświetlanie informacji o interfejsach.
- `route` konfiguracja tras routingowych
- `vconfig` dodawanie i usuwanie obsługi wskazanych VLANów z danego interfejsu
- `brctl` konfiguracja programowego switcha ethernetowego pomiędzy interfejsami (bridge)
- `ifenslave` konfiguracja bondów

Innym przydatnym poleceniem z pakietu iproute2 jest `tc`, które służy do konfiguracji ustawień kontroli przepływu (np. kolejkowania) na interfejsach sieciowych. Do konfiguracji samych ethernetowych interfejsów fizycznych (np. wymuszenia prędkości karty) używane jest polecenie `ethtool`. Natomiast do konfiguracji sieci wifi mogą być użye polecenia takie jak:

- `iwconfig` konfiguracja (większości - do części trzeba użyć wlanctl-ng) bezprzewodowych interfejsów sieciowych
- `wpa_supplicant` konfiguracja większości bezprzewodowych interfejsów sieciowych z wsparciem dla WPA
- `wpa_cli` konfiguracja większości bezprzewodowych interfejsów sieciowych z wsparciem dla WPA
- `iwlist` dodatkowe informacje z bezprzewodowych interfejsów sieciowych (przydatna zwłaszcza opcja scan pokazująca dostępne sieci

Warto zaznaczyć iż konfiguracja dokonywana poleceniami takimi jak `ip`, `tc`, `ifconfig`, itp. jest konfiguracją typu *runtime*, czyli jest tracona po wyłączeniu systemu. Aby konfiguracja sieci była trwała należy polecenia takie zapisać w postaci skryptu uruchamianego przy starcie systemu lub skorzystać z systemowych plików konfiguracyjnych związanych z siecią.

Zadanie 5.0.1

Napisz polecenie które ustawi adres ip `172.33.13.113` (maska sieci to `255.255.255.0`) na interfejsie `eth5`.

Zadanie 5.0.2

Napisz polecenie które ustawi trasę routingową do sieci `10.13.0.0/16` przez bramkę o adresie ip `172.33.13.13`.

5.1 Konfiguracja DNS

Za zamianę nazw domenowych na adresy IP odpowiadają funkcje biblioteki standardowej C. Korzysta ona do tego celu z konfiguracji zawartej w pliku `/etc/resolv.conf`. Powinien on zawierać co najmniej jeden wpis postaci `nameserver ip_serwera_dns`, określający serwer rozwiązujący nazwy DNS do którego będziemy kierować nasze zapytania. Wpisów tych może być kilka co pozwala na określenie serwerów uzywany w przypadku niedostępności podstawowego (obecnie używane są maksymalnie 3).

Dodatkowo plik ten może posiadać wpisy `domain` określający domenę lokalną (jeżeli nie jest tu określona a hostname zawiera domenę to używana jest ta z hostname; jeżeli nie chcemy używać można określić na `.`) oraz `search` określający listę domen do przeszukiwania. Określają one domeny, króre będą dodawane jako surfix do domeny o którą się pytamy. Na przykład gdy mamy `domain abc.def`, a pytamy się o `xyz` (bez kropki w środku lub na końcu), biblioteka najpierw spróbuje ustalić adres `xyz.abc.def.` a potem `xyz.`

Plik ten pozwala ustawić także inne opcje związane z odpytywaniem DNS - szczegóły w `man 5 resolv.conf`.

Innym plikiem związanym z rozwijaniem nazw jest `/etc/hosts`, który stanowi bazę mapowań nazw na numery IP. Jest on użyteczny dla lokalnie definiowanych nazw i adresów. Wpisy w nim zawarte mają priorytet wyższy od informacji z DNS (jeżeli host został znaleziony w tym pliku nie jest wykonywane zapytanie do serwera rozwijającego DNS).

5.2 konfiguracja automatyczna

W zależności od ustawień sieci do której podłączony jest konfigurowany host możliwe jest także skorzystanie z konfiguracji automatycznej DHCP i/lub autokonfiguracji IPv6.

5.2.1 DHCP

DHCP jest protokołem typu klient-serwer, pozwalającym klientowi uzyskać informacje na temat konfiguracji sieci takie jak adres ip, długość prefixu, trasy routingowe (w szczególności adres bramki domyślnej), adresy serwerów DNS zarówno dla IPv4, jak i IPv6.

Do pobrania konfiguracji z serwera DHCP i jej ustawienia służy najczęściej polecenie `dhclient` (dostępne są inne implementacje klienta dhcp, np: `udhcpc`, `dhcpcd`). Z ważniejszych opcji należy wspomnieć o:

- `-6` – korzystanie z DHCPv6, czyli DHCP dla protokołu IPv6,
- `-n` – nie ustawianie / używanie pobranej konfiguracji,
- `-d` – nie przechodzenie w tło (włącza też `-v`),
- `-v` – wypisywanie większej informacji o działaniu programu.

Dostępne są też różne narzędzia diagnostyczne związane z DHCP, np: `dhcping`, `dhcp-probe`. Linux może pełnić także funkcję serwera DHCP, przy pomocy aplikacji takich jak np.: *isc-dhcp-server*, *udhcpd*, *dnsmasq*, *odhcp6c*, *dhcpy6d*, *wide-dhcpv6*.

5.2.2 IPv6 autoconf

Innym sposobem automatycznej konfiguracji interfejsów sieciowych, wprowadzonym w IPv6 jest autokonfiguracja w oparciu o adresy link-local generowane w oparciu o MAC adres karty sieciowej. Polega ona na tym że dla podsieci będących LAN'em przydzielana jest pula z maską /64 co umożliwia tworzenie unikalnych numerów IP w oparciu o (niepowtarzalne) numery sprzętowe MAC. 64 bitowy prefiks sieci jest informacją rozgłaszaną przy pomocy ICMPv6 przez routery (mechanizm radvd), a host dokleja do niego część go identyfikującą związaną z adresem link-local. Radvd rozgłasza także informacje routingowe (takie jak adres bramy - dhcpv6 tego nie potrafi), niestety nie da się rozgłaszać w ten sposób innej od standardowej dla LAN długości prefixu.

Linux domyślnie ma włączony ten mechanizm, można go jednak wyłączyć poprzez `echo 0 > /proc/sys/net/ipv6/conf/${IFACE}/autoconf`, gdzie `${IFACE}` oznacza interfejs na którym chcemy wyłączyć ten mechanizm.

5.3 Konfiguracja w proc

Konieczne / przydatne może być dokonywanie pewnych ustawień poprzez jądrowe systemy plików `/proc` i `/sys`. Najczęstszym przypadkiem jest włączenie przekazywania pakietów pomiędzy interfejsami poprzez:

```
for f in /proc/sys/net/ipv*/conf/*/forwarding; do echo 1 > $f; done
```

(powyższy jednolinijkowiec włącza forwading pakietów IP dla IPv4 i IPv6 na wszystkich interfejsach)

Innym przykładem jest pokazane wcześniej wyłączenie automatycznej konfiguracji IPv6, przydatne gdy chcemy korzystać tylko z ręcznie przydzielanych adresów.

Z opisem poszczególnych ustawień w ramach systemu `/proc/sys` (w tym tych poświęceonych obsłudze sieci z `/proc/sys/net`) można zapoznać się m.in. na stronie https://sysctl-explorer.net/.

5.4 Filtracja pakietów

Oprócz wyżej omówionej konfiguracji interfejsów i tras routingowych, często potrzebna jest konfiguracja jądrowych mechanizmów filtracji pakietów.

Filtracja pakietów umożliwia m.in. ignorowanie (*drop*) lub odrzucenie z komunikatem błędu (*reject*) niepożądanego ruchu sieciowego – zarówno wchodzącego, wychodzącego, jak i przekazywanego (jeżeli uruchomiona jest funkcjonalność routera poprzez wpisanie wartości 1 do `/proc/sys/net/ipv*/conf/*/forwarding`). Pozwala także na śledzenie połączeń (np. w celu innego traktowania połączeń nawiązanych niż nowych) i manipulowanie przechodzącymi pakietami (np. modyfikację adresów IP i numerów portów w ramach mechanizmów *snat* modyfikującego adresy źródłowe i *dnat* modyfikujące adresy docelowe).

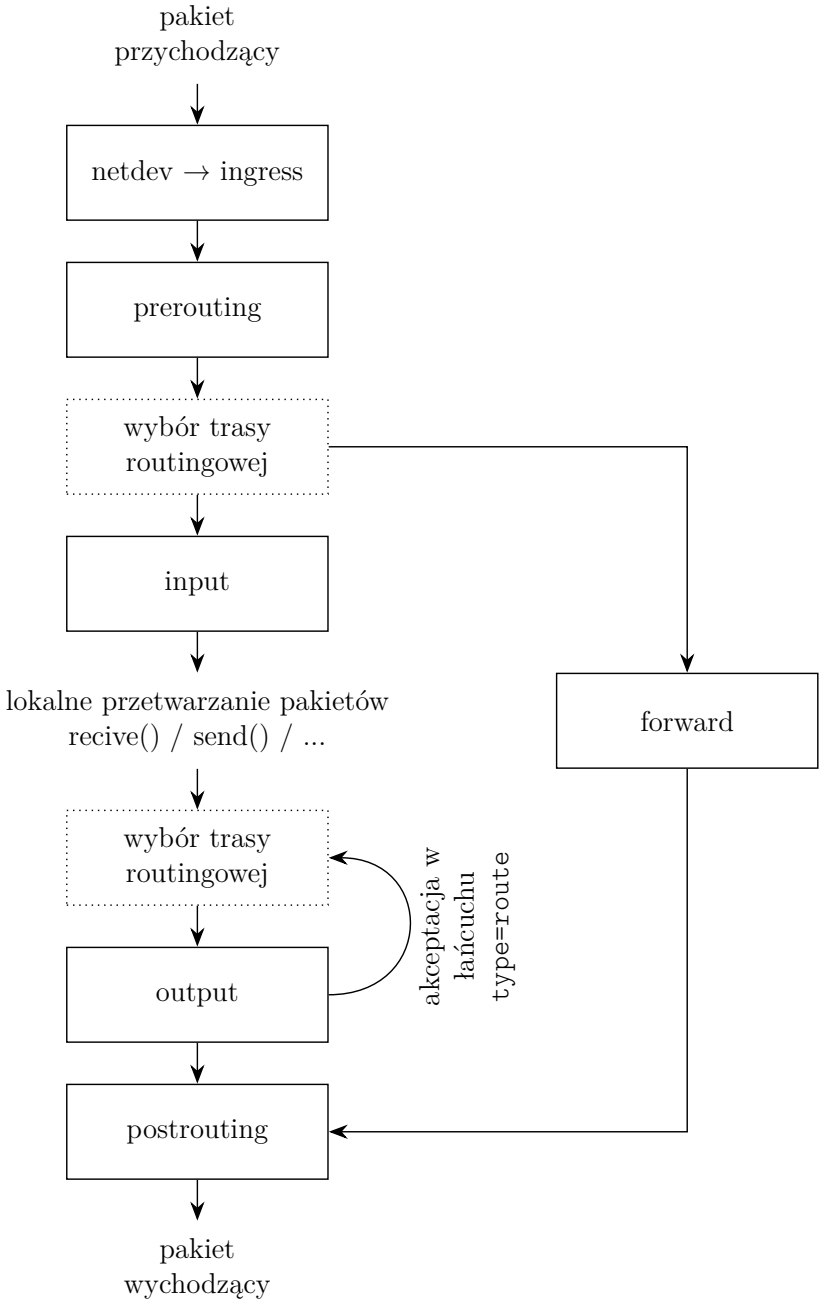
Do konfiguracji filtracji pakietów służy polecenie `nft`. Na starszych systemach `nft` może być niedostępny, wtedy można korzystać z poleceń:

- `iptables`, `ip6tables` konfiguracja filtrów działających na pakietach IP (`iptables` dla IPv4, `ip6tables` dla IPv6), filtracja może odbywać się m.in. w oparciu o źródłowe i docelowe adresy IP, numery portów, protokół warstwy transportowej, interfejsy oraz mechanizm śledzenia połączeń; umożliwia także konfigurację translacji adresów (NAT).
- `ebtables` konfiguracja filtrów działających na poziomie switcha ethernetowego, filtracja może odbywać się m.in. w oparciu o źródłowe i docelowe interfejsy i adresy sprzętowe.
- `arptables` konfiguracja filtrów związanych z protokołem ARP (zamiany adresów IP na adresy sprzętowe)

Konfiguracja filtracji pakietów dokonywana z użyciem poleceń `nft`, `iptables` jest konfiguracją *runtime* i jest tracona po wyłączeniu systemu.

5.4.1 nft (nftables)

Polecenie `nft list ruleset` pozwala na wylistowanie wszystkich reguł.



Rysunek 6: Trasa pakietu przez filtry nftables. Wskazano punkty zaczepień dla łańcuchów reguł.

Tabele, łańcuchy i reguły

- Reguły (`rule`) grupowane są w łańcuchy (`chains`) w ramach których przetwarzane są kolejno (do momentu napotkania reguły kończącej przetwarzanie pakietu).
- Łańcuchy grupowane są w tabele (`table`).
- Każda tabela ma określoną rodzinę obsługiwanych adresów (`family`), mogą to być:
  - `inet` (osobne lub wspólne reguły dla IPv4 i IPv6),
  - `ip` (reguły tylko dla IPv4),
  - `ip6` (reguły tylko dla IPv6),

  - `arp` (reguły dla warstwy L2 przetwarzane przed uruchomieniem procesowania IP),
  - `bridge` (reguły przetwarzane dla pakietów przechodzących przez softwerowy bridge),
  - `netdev` (reguły przetwarzane w momencie wejścia ruchu na urządzenie sieciowe, urządzenie musi być określone dla łańcucha reguł, może być alternatywą dla `tc`).
- Tabel danej rodziny może być wiele, stosowane będą łańcuchu z wszystkich tych tabel (odpowiednio do ich parametrów).
- Tabele dla różnych rodzin mogą mieć taką samą nazwę.

Kierowanie ruchu do reguł

- Ruch do łańcucha może być kierowany jawnie przez regułę w innym łańcuchu lub automatycznie w oparciu o parametry danego łańcucha: typ (`type`), punkt zaczepienia (`hook`) i priorytet (`priority`).
- Pasujące łańcuchy (o tym samym punkcie zaczepienia) będą przetwarzane kolejno wg priorytetów do momentu napotkania reguły kończącej przetwarzanie pakietu w którymś z tych łańcuchów (lub przetworzenia wszystkich reguł).
- Podstawowym typem łańcucha jest `filter`. Dodatkowo mogą być użyte typy:
  - `nat` – translacja adresów sieciowych w oparciu o śledzenie połączenie (`conntrack`), reguły przetwarzają tylko pierwszy pakiet połączenia, pozostałe przetwarza utworzony wpis `conntrack`, typ może być użyty jedynie w łańcuchach tabel związanych z protokołami IP (inet, ip, ip6) z wyjątkiem łańcucha `forward`
  - `route` – zaakceptowanie w takim powoduje wyszukanie nowej trasy routingowej, typ może być użyty jedynie w łańcuchach wyjściowych (zaczepionych w `output`) tabel związanych z protokołami IP (inet, ip, ip6)
- Dostępne punkty zaczepienia reguł zależą od rodziny:
  - dla `inet`, `ip`, `ip6` i `bridge` są to: prerouting input forward output postrouting
  - dla `arp` są to: input output
  - dla `netdev` są to: ingress
- Priorytet jest określany swobodnie i może być wartością ujemny lub dodatnią. Warto mieć świadomość iż śledzenie pakietów (`conntrack`) na wejściu ma priorytet -200 (jest robione przed większością innych reguł) a na wyjściu 300 (jest robione po większości innych reguł).

Konstrukcja poleceń

Polecenia nft można konstruować na kilka sposobów. Możemy dla każdego elementu tworzonego firewalla wywoływać polecenie nft – np poniższe polecenie utworzy tablicę ABC, w niej łańcuch XYZ i w nim jedną dwie reguły (akceptację ruchu wchodzącego interfejsem ”eth0” i eth1):

```
nft 'add table ip ABC';  nft 'add chain ip ABC XYZ'
nft 'add rule  ip ABC XYZ iifname "eth0" accept'
nft 'add rule  ip ABC XYZ iifname "eth1" accept'
```

Możemy też kilka poleceń nft podać w jenym uruchomieniu komendy nft, rozdzielając je średnikiem[3]:

```
nft 'add table ip ABC; add chain ip ABC XYZ
     add rule ip ABC XYZ iifname "eth0" accept
     add rule ip ABC XYZ iifname "eth1" accept'
```

3. Zauważ że popzrednio średnik był poza cudzysłowem (aby bash zinterpretował go jako koniec pierwszej komendy), a teraz musi być wewnątrz cudzyłowia lub być zabezpieczony w inny sposób (aby bash nie zinterpretował go jako koniec komendy).

Zauważ że w pierwszej linii mamy dwa polecenia nft rozdielone średnikiem, a w kolejne nie są już nim rozdzielane. Wynika to z tego że w składni nft – podobnie jak w bashu – średnik może zostać zastąpiony nową linią.

W obu tych wypadkach dodając kolejną regułem musimy każdorazowo powtarać określenie jej lokalizacji (typ tablicy, jej nazwę i nazwę łańcucha). Aby tego uniknąć można zastoswać zapis z blokami wydzielanymi przy pomocy nawiasów klamrowych:

```
nft 'table ip ABC { chain XYZ { iifname "eth0" accept; iifname "eth1" accept; }; };'
```

Zauważ średniki po każdym z wewnętrznych poleceń i po klamerkach kończących bardziej zewnętrzne polecenia. W przypadku zapisu wieloliniowego, gdyby występował tam znak nowej linii mogłyby być one pominięte.

Wszystkie powyższe zapisy generują identyczny układ reguł firewalla. Zapis ten można jeszcze bardziej skompresować, ale uzyskamy wtedy też bardziej skompresowaną regułę firewalla:

```
nft 'table ip ABC { chain XYZ { iifname {"eth0", "eth1"} accept; }; };'
```

Pliki konfiguracyjne

```
#!/usr/sbin/nft -f
flush ruleset

table inet filter {
        chain INPUT {
                type filter hook input priority 0; policy drop;

                # lo and established / invalid connections
                iifname "lo" accept
                ct state {established, related} accept
                ct state invalid reject

                # icmp, igmp
                meta l4proto icmp icmp type timestamp-request reject
                meta l4proto {icmp, ipv6-icmp, igmp} accept

                # ssh
                ip  saddr 10.40.0.0 tcp dport ssh accept
                ip6 saddr {
                        2001:db8:0:a17::123,
                        2001:db8:0:1313::/64
                } tcp dport ssh accept

                # reject all other packets with ICMP error
                reject
        }
}
```

Zauważ że zamiast powtarzać regułę dla każdego adresu:

```
        ip6 saddr 2001:db8:0:a17::123 tcp dport ssh accept
        ip6 saddr 2001:db8:0:1313::/64 tcp dport ssh accept
```

możemy podać zbiór parametrów (np. adresów) w klamerkach w ramach jednej reguły (tak jak pokazano powyżej). Możliwe jest także definiowanie zbiorów adresów (`set`) i odwoływanie się do nich z użyciem `@nazwa`.

Podobnie jak przy wpisywaniu poleceń nftables bespośrednio w argumentach polecenia nft, także w plikach konfiguracyjnych możemy zapisywać je zarówno w „notacji klamerkowej" (jak powyżej) jak i ciągu kolejnych poleceń - np.:

```
#!/usr/sbin/nft -f
add table ip filter
add chain ip filter POST {type nat hook postrouting priority 100; policy accept;}
add rule ip filter POST oifname "ens4" ip saddr 10.40.0.0/24 snat to 213.135.50.250
```

Co tworzy maskowanie adresów IP z 10.40.0.0/24 na 213.135.50.250 dla ruchu wychodzącego interfejsem ens4 i jest równoważne:

```
#!/usr/sbin/nft -f
table ip filter {
        chain POST {
                type nat hook postrouting priority 100; policy accept;
                oifname "ens4" ip saddr 10.40.0.0/24 snat to 213.135.50.250
        }
}
```

Od wersji 0.9.2 nftables możliwe jest też tworzenie wspólnych reguł dla udp i tcp w następujący sposób:
`add rule inet filter INPUT meta l4proto {tcp, udp} th dport domain`

Zadanie 5.4.1

Napisz polecenia które włączą przekazywanie pakietów (routing) pomiędzy interfejsami `eth3` i `eth4`, ale nie zezwolą na przekazywanie pakietów innymi interfejsami (w tym pakietów inny interfejs ↔ `eth3` / `eth4`).
Wskazówka: skorzystaj z reguł filtracji pakietów

5.4.2 iptables

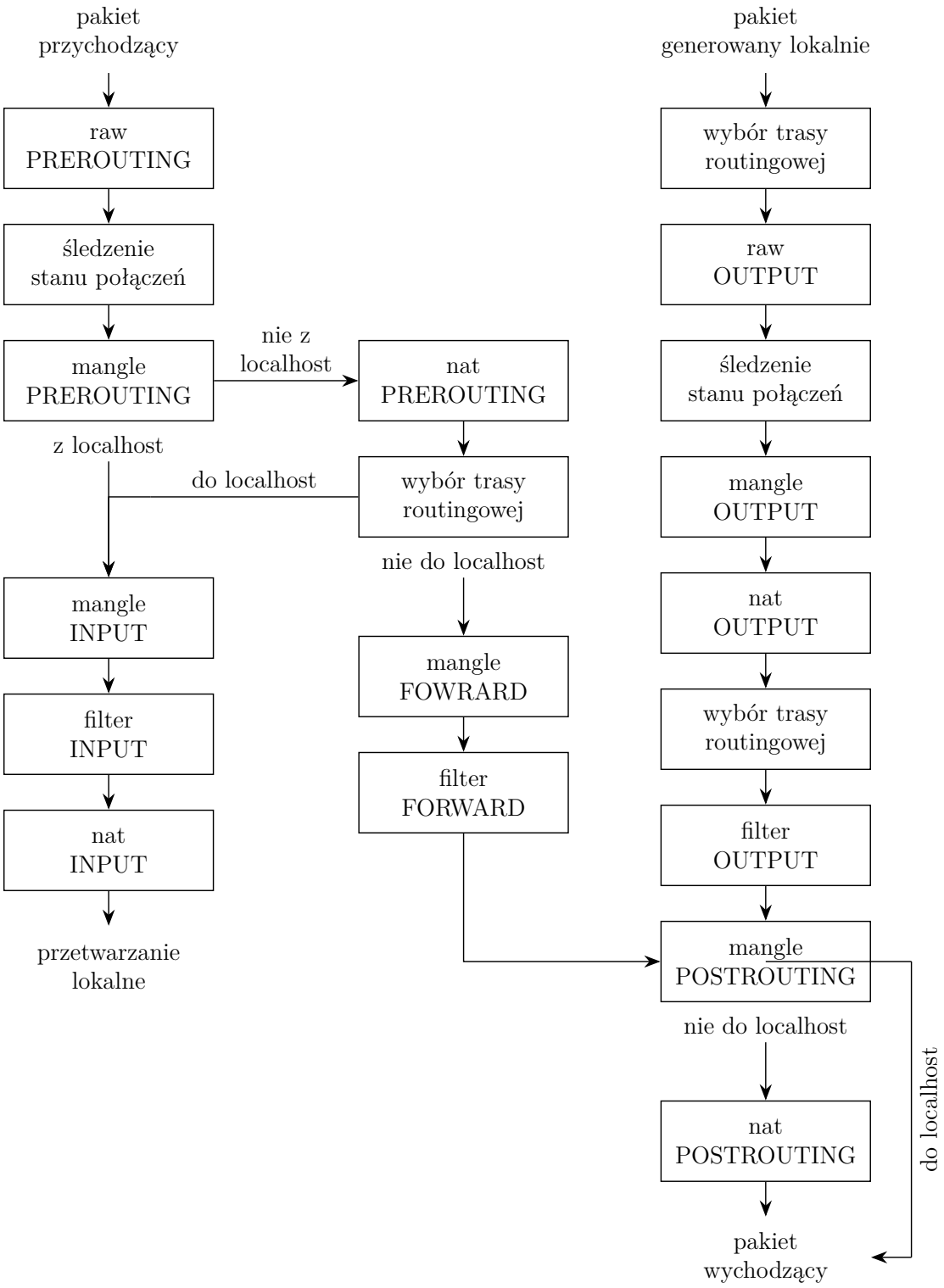
Iptables wykorzystuje kilka tablic reguł (najistotniejszymi są `filter` i `nat`). Tablica może zostać określona przy pomocy opcji `-t`, jeżeli nie użyto tej opcji operacje będą wykonywane na tablicy `filter`. Zależności pomiędzy poszczególnymi łańcuchami i tablicami przedstawia (uproszczony) diagram przejścia pakietu przez mechanizm iptables:

W każdej z tablic występuje kilka różnych łańcuchów reguł. Każdy łańcuch posiada akcję domyślną, która może zostać ustawiona komendą `iptables [-t TABLICA] -P ŁAŃCUCH AKCJA`. Reguły do wskazanego łańcucha (w wskazanej tablicy) mogą być dodawane/usuwane z użyciem komend:

- `iptables [-t TABLICA] -A|-D ŁAŃCUCH REGUŁA` – dodanie (`-A`) lub usunięcie (`-D`) reguły
- `iptables [-t TABLICA] -I ŁAŃCUCH POZYCJA REGUŁA` – wstawienie reguły na wskazaną pozycję
- `iptables [-t TABLICA] -F ŁAŃCUCH` – usuniecie wszystkich reguł z łańcucha

Reguły składają się ze zbioru dopasowań (filtrów) w postaci opcji do komendy iptables oraz akcji podawanej po opcji `-j`, do najistotniejszych filtrów należą:

- `-s ADRES` – pasuje gdy adres źródłowy w pakiecie zgadza się z podaną siecią IP (lub pojedynczym adresem)
- `-d ADRES` – pasuje gdy adres docelowy w pakiecie zgadza się z podaną siecią IP (lub pojedynczym adresem)
- `-p PROTOKÓŁ --dport PORT` – pasuje gdy pakiet zawiera w sobie pakiet wskazanego protokołu (np. tcp, udp) i adresowany jest na wskazany numer portu



Rysunek 7: Trasa pakietu przez filtry iptables. Wskazano punkty zaczepień nazwy łańcuchów.

- `-i INTERFEJS` – pasuje gdy pakiet przyszedł wskazanym interfejsem sieciowym
- `-o INTERFEJS` – pasuje gdy pakiet wychodzi wskazanym interfejsem sieciowym

Najistotniejszymi akcjami jest ACCEPT (zaakceptowanie/przepuszczenie pakietu przez łańcuch), REJECT (odrzucenie pakietu z wygenerowaniem komunikatu błędu poprzez ICMP), DROP (zapomnienie o pakiecie / ciche zignorowanie) oraz LOG (zapisanie informacji do logu).

Przykład konfiguracji iptables:

```
# polityki domyślne
```

```
iptables -P INPUT DROP
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT

# interfejs lokalny oraz połączenia nawiązane
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A INPUT -m state --state INVALID -j REJECT

# SSH
iptables -A INPUT -p tcp --dport ssh -s 0.0.0.0/0 -j sshguard
iptables -A INPUT -p tcp --dport ssh -s 0.0.0.0/0 -j ACCEPT

## ICMP
iptables -A INPUT -p icmp -j ACCEPT

## RESZTA
iptables -A INPUT -j REJECT
```

Do wyświetlenia wszystkich reguł można użyć komendy `iptables-save` / `ip6tables-save`. Generuje ona skrypt który może zostać wczytany przy pomocy `iptables-restore` / `ip6tables-restore`.

5.5 Sieci bezprzewodowe

Linux może być także klientem sieci bezprzewodowych WiFi - do ich konfiguracji przydać mogą się następujące narzędzia:

- `iwconfig` – podstawowe operacje na interfejsie bezprzewodowym
- `iwlist` – listowanie ”widocznych” sieci i informacji o nich
- `wpa_supplicant` – łączenie się z sieciami zabezpieczonymi WPA

Linux może być nie tylko klientem takich sieci, ale pełnić w nich też funkcję access pointu, do tego zadania pomocne mogą być narzędzia takie jak:

- `hostapd` – uruchomienie access pointa na bazie linuxa i karty wifi
- `dnsmasq` – serwer DHCP oraz serwer maskujący DNS
  (nie tylko dla sieci bezprzewodowych, ale często używany z hostapd)

oraz oczywiście odpowiednia konfiguracja routingu i przekazywania pakietów.

6 Programowanie usług sieciowych

6.1 wysyłanie danych po UDP

```
import socket, sys

if len(sys.argv) != 3:
  print("USAGE: " + sys.argv[0] + " dstHost dstPort", file=sys.stderr)
  exit(1)

# pobieramy informacje o adresach na które rozwija się podana nazwa hosta / usługi
# dzięki tej funkcji jako dstHost możemy podać zarówno nazwę domenową jak i numer IP
# (w którejś z standardowych notacji) hosta z którym chcemy się połączyć
```

```
# a jako dstPort numer portu lub nazwę usługi z /etc/services
dstAddrInfo = socket.getaddrinfo(sys.argv[1], sys.argv[2], type=socket.SOCK_DGRAM)

# funkcja ta nam zwraca listę dostępnych możliwości połączenia (np. nazwa domenowa
# może rozwijać się na wiele różnych adresów), przekazując do getaddrinfo
# opcjonalny argument type zawęziliśmy ta listę do połączeń UDP (SOCK_DGRAM)

# moglibyśmy próbować kolejnych adresów w pętli, ale w tym prostym przykładzie
# próbujemy użyć jedynie pierwszego ze zwróconych adresów
dstAddrInfo = dstAddrInfo[0]

# otwieramy gniazdo
sfd = socket.socket(dstAddrInfo[0], socket.SOCK_DGRAM)

# wysyłamy dane
sfd.sendto("Ala ma kota".encode(), dstAddrInfo[4])
```

6.2 odbiór danych po UDP

```
import socket, sys

if len(sys.argv) != 2:
  print("USAGE: " + sys.argv[0] + " listenPort", file=sys.stderr)
  exit(1)

# otwieramy gniazdo
sfd = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)

# ustawiamy opcję gniazda pozwalającą na jednoczesną obsługę IPv4 i IPv6
sfd.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_V6ONLY, 0)

# ustawiamy adres i port na którym słuchamy
# adres zerowy oznacza słuchanie na wszystkich adresach IP danego hosta
sfd.bind(('::', int(sys.argv[1])))

while True:
  # czekamy na dane, gdy je otrzymamy to odbieramy
  data, sAddr, = sfd.recvfrom(4096)
  # i wypisujemy co i od kogo dostaliśmy
  print("odebrano od", sAddr, ":", data.decode());
```

6.3 klient TCP

```
import socket, select, sys

if len(sys.argv) != 3:
  print("USAGE: " + sys.argv[0] + " dstHost dstPort", file=sys.stderr)
  exit(1);
```

```
# struktura zawierająca adres na który wysyłamy
dstAddrInfo = socket.getaddrinfo(sys.argv[1], sys.argv[2], proto=socket.IPPROTO_TCP)

# mogliśmy uzyskać kilka adresów, więc próbujemy używać kolejnych do skutku
for aiIter in dstAddrInfo:
  try:
    print("try connect to:", aiIter[4])
    # utworzenie gniazda sieciowego ... SOCK_STREAM oznacza TCP
    sfd = socket.socket(aiIter[0], socket.SOCK_STREAM)
    # połączenie ze wskazanym adresem
    sfd.connect(aiIter[4])
  except:
    # jeżeli się nie udało ... zamykamy gniazdo
    if sfd:
      sfd.close()
    sfd = None
    # i próbujemy następny adres
    continue
  break;

if sfd == None:
  print("Can't connect", file=sys.stderr)
  exit(1);

# wysyłanie
sfd.sendall("Ala ma Kota\n".encode())

# czekanie na dane i odbiór danych
while True:
  rdfd, _, _ = select.select([sfd], [], [], 13.0)
  if sfd in rdfd:
    d = sfd.recv(4096)
    d = d.decode()
    print(d, end="")

    # odbiór pustego pakietu lub pakietu zawierającego
    # jedynie pustą linię kończy działanie
    if d == "" or d == "\n" or d == "\r\n":
      break
  else:
    # timeout kończy działanie
    break

# zamykanie połączenia
sfd.shutdown(socket.SHUT_RDWR)
sfd.close()
```

Kod do pobrania: https://bitbucket.org/OpCode-eu-org/opcode-vip/raw/master/vademecum/code-src/sieciowe/TCP_-_klient.py

6.4 serwer TCP

```
import socket, select, signal, sys, os

MAX_CHILD = 5
QUERY_SIZE = 3
TIMEOUT = 13
BUF_SIZE = 4096

if len(sys.argv) != 2:
  print("USAGE: " + sys.argv[0] + " listenPort", file=sys.stderr)
  exit(1);

# obsługa sygnału o zakończeniu potomka
childNum = 0
def onChildEnd(s, f):
  print("odebrano sygnał o śmierci potomka")
  global childNum
  childNum -= 1
  os.waitpid(-1, os.WNOHANG);
signal.signal(signal.SIGCHLD, onChildEnd)

# utworzenie gniazd sieciowych ... SOCK_STREAM oznacza TCP
sfd_v4 = socket.socket(socket.AF_INET,  socket.SOCK_STREAM)
sfd_v6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)

# ustawienie opcji gniazda ... IPV6_V6ONLY=1 wyłącza korzystanie
# z tego samego socketu dla IPv4 i IPv6
sfd_v6.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_V6ONLY, 1)

# przypisanie adresów ...
# '0.0.0.0' oznacza dowolny adres IPv4 (czyli to samo co INADDR_ANY)
# '::' oznacza dowolny adres IPv6 (czyli to samo co in6addr_any)
sfd_v4.bind(('0.0.0.0', int(sys.argv[1])))
sfd_v6.bind(('::',      int(sys.argv[1])))

# określenie gniazd jako używanych do odbioru połączeń przychodzących
# (długość kolejki połączeń ustawiona na wartość QUERY_SIZE)
sfd_v4.listen(QUERY_SIZE)
sfd_v6.listen(QUERY_SIZE)

# czekanie na połączenia z użyciem select() w nieskończonej pętli
while True:
  sfd, _, _ = select.select([sfd_v4, sfd_v6], [], [])
  for fd in sfd:
    #  odebranie połączenia
    sfd_c, sAddr = fd.accept()

    # weryfikacja ilości potomków
    if childNum >= MAX_CHILD:
      print("za dużo potomków - odrzucam połączenie od:", sAddr);
      sfd_c.send("Internal Server Error\r\n".encode())
```

```
      sfd_c.close()
      break

    # aby móc obsługiwać wiele połączeń rozgałęziamy proces
    pid = os.fork()
    if pid > 0:
      # rodzic - zwiększamy licznik potomków
      childNum += 1
    else:
      # potomek - obsługa danego połączenia
      print("połączenie od:", sAddr)
      while True:
        # czekanie na dane z timeout'em
        # aby zabezpieczyć się przed atakiem DoS
        rd, _, _ = select.select([sfd_c], [], [], TIMEOUT)
        if sfd_c in rd:
          data = sfd_c.recv(BUF_SIZE)
          if data:
            print("odebrano od", sAddr, ":", data.decode());
            sfd_c.send(data)
          else:
            print("koniec połączenia od:", sAddr)
            break
        else:
          print("timeout połączenia od:", sAddr)
          break
      # zamykanie połączenia
      sfd_c.shutdown(socket.SHUT_RDWR)
      sfd_c.close()
      sys.exit()
```

Kod do pobrania: https://bitbucket.org/OpCode-eu-org/opcode-vip/raw/master/vademecum/code-src/sieciowe/TCP_-_serwer.py

Zadanie 6.0.1

Napisz (w Pythonie lub C/C++) serwer TCP, który oczekuje że klient wyśle mu liczbę (z zakresu od 2 do 9), w odpowiedzi na którą serwer odeśle do tego klienta trójkąt z gwiazdek odpowiedniej wielkości. Na przykład dla żądania klienta w postaci ”3” powinien to być:

```
*
**
***
```

Zadanie 6.0.2

W skrypcie znajdują się przykładowe kody wysyłający dane po UDP (”klient UDP”) i odbierający dane po UDP (”serwer UDP”) oraz kod serwera usługi ”echo” (odsyłającej odebrane dane do nadawcy) w wariancie TCP, którą omawialiśmy na wykładzie.
W oparciu o te informacje napisz (w Pythonie lub C/C++) program realizujący funkcję serwera echo z użyciem UDP.

Zadanie 6.0.3

Uruchom dwie instancje serwera echo korzystającego z protokołu UDP.
Zastanów się co by się stało jeżeli jeden z tych serwerów dostałby pakiet pochodzący od drugiego z nich?
Korzystając z pakietu *scapy* oraz posiadając prawa root'a możemy przy pomocy Pythona wysyłać dowolnie spreparowane pakiety IP:

```
from scapy.all import IP, IPv6, UDP, send

send(IPv6(src=sIP, dst=dIP) / UDP(sport=sPort, dport=dPort) / "ABC ... XYZ")
# powyższa funkcja utworzy (a następnie wyśle):
#  → pakiet IPv6 od sIP do dIP
#    (adresy podajemy jako napisy),
#  → w którym jest pakiet UDP z portem źródłowym sPort i docelowym dport
#    (porty podajemy jako wartości numeryczne)
#  → w którym są dane "ABC ... XYZ"

# jeżeli zamiast IPv6() użyjemy IP() będziemy używać pakietu IPv4

# możemy też zaimportować inne funkcjonalności z modułu scapy
# (np. ICMP, TCP, ...) i używać ich do budowy naszych pakietów
```

Modyfikując powyższy kod spróbuj wysłać sfałszowany pakiet adresowany do jednego z serwerów, który jako adres nadawcy ma podany drugi z serwerów.
Scapy nie jest elementem biblioteki standardowej pythona – konieczne może być zainstalowanie pakietu ***python3-scapy*** *albo zainstalowanie go poprzez managera modułów pythonowych „pip": **pip3 install scapy-python3**.*

Zadanie 6.0.4

Zobacz co się stanie jeżeli w sfałszowanym pakiecie podasz ten sam serwer jako nadawcę i odbiorcę. Usługa UDP-echo była kiedyś powszechnie stosowaną usługą diagnostyczną umożliwiająca testowanie połączenia sieciowego. Do tej pory ma nawet przyznany standardowy numer portu (7). Jak myślisz dlaczego usługa UDP-echo nie jest już powszechnie dostępną na każdym komputerze podłączonym do Internetu?

7 Wykład wideo[4]

- *Podstawy sieci komputerowych* – http://video.opcode.eu.org/10.01-sieci-podstawy.mkv
- *Internet Protocol* – http://video.opcode.eu.org/10.02-sieci_ip.mkv
- *Adresacja i routing IP* – http://video.opcode.eu.org/10.03-routing_ip.mkv
- *TCP, UDP, ICMP i DNS* – http://video.opcode.eu.org/10.04-tcp_udp_dns_icmp.mkv
- *Usługi sieciowe i tunele* – http://video.opcode.eu.org/10.05-uslugi_tunele.mkv
- *Ethernet* – http://video.opcode.eu.org/11.01-ethernet.mkv
- *Standardy sieciowe* – http://video.opcode.eu.org/11.02-standardy.mkv
- *Konfiguracja sieci* – http://video.opcode.eu.org/11.03-konfiguracja.mkv

4. Filmy posiadają napisy wgrane do kontenera multimedialnego jako osobny strumień – napisy mogą być włączone lub wyłączone w odtwarzaczu. W wielu filmach dużo dzieje się "na dole ekranu", dlatego polecamy odtwarzać filmy z napisami umieszczonymi poniżej filmu, np. przy pomocy polecenia: `vlc --video-filter='croppadd{paddbottom=120}' --sub-margin=-10 PLIK.mkv`

- *Programowanie usług sieciowych* – http://video.opcode.eu.org/11.04-programowanie.mkv

8 Literatura dodatkowa

- *SSH jako VPN* (http://blog.opcode.eu.org/2020/06/09/ssh_jako_vpn.html) – opis konfiguracji tuneli SSH
- *Linux - podręcznik administratora sieci* (http://www.interklasa.pl/portal/index/subjectpages/informatyka/linuxadm.pdf)
- *Introduction to TCP/IP* (https://www.coursera.org/learn/tcpip/home/welcome) – kurs na coursera.org

9 Rozwiązania zadań

Treści zadań zamieszczone zostały w odpowiednich rozdziałach skryptu.

Poniżej zamieszczone są przykładowe rozwiązania „głównych” zadań z tego skryptu wraz z komentarzami. Wiemy że zajrzenie do nich już przy pierwszej trudności jest kuszące, mimo to rekomendujemy przynajmniej podjąć uczciwą, co najmniej kilkunastominutową na każde z zadań, próbę rozwiązania tych zadania bez zaglądania do odpowiedzi.

Pamiętaj!: Samodzielne rozwiązanie problemu (wraz z wszystkimi trudnościami po drodze i popełnionymi błędami) jest dużo bardziej kształcące od nawet wielokrotnego przepisania gotowego rozwiązania, jednak nawet jednokrotne przepisanie rozwiązania jest bardziej kształcące od wielokrotnego przekopiowania go.

Rozwiązanie zadania 3.0.1

```
$ ping -c4 cieksi.icm.edu.pl
PING wwww2.icm.edu.pl (213.135.59.55) 56(84) bytes of data.
64 bytes from wwww2.icm.edu.pl (213.135.59.55): icmp_seq=1 ttl=60 time=3.91 ms
64 bytes from wwww2.icm.edu.pl (213.135.59.55): icmp_seq=2 ttl=60 time=3.63 ms
64 bytes from wwww2.icm.edu.pl (213.135.59.55): icmp_seq=3 ttl=60 time=2.94 ms
64 bytes from wwww2.icm.edu.pl (213.135.59.55): icmp_seq=4 ttl=60 time=5.03 ms

--- wwww2.icm.edu.pl ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 6ms
rtt min/avg/max/mdev = 2.942/3.875/5.025/0.752 ms
```

Polecenie na standardowym wyjściu wypisuje m.in.:

- odpytywany adres IP i wynik odwrotnego dns'u dla niego
- ilość danych wysyłanych (może być użyteczne do testowania problemów z MTU)
- dla każdego zapytania, które dostało odpowiedź:

Rozwiązanie zadania 1.3.1

tak — zakres sieci 2001:06a0:0000:0000:0000:0000:0000:0000 - 2001:06a0:0000:003f:ffff:ffff:ffff:ffff

Rozwiązanie zadania 1.4.1

Adres 2001:db8:abc:21:123::ff3 zawiera się w zakresie sieci następujących sieci wystepujących w tablicy routingu:

- 2001:db8::/32
- 2001:db8:abc:21::/64
- default (0::/0)

Najdłuższy prefix spośród tych sieci ma 2001:db8:abc:21::/64 (64 bity), zatem zostanie użyty wpis w tablicy routingowej dla tej sieci.
W efekcie pakiet zostanie skierowany do routera 2001:db8:ff:21::1 poprzez interfejs eth2.

- ilość przesłanych otrzymanych w pakiecie
- serwer który odpowiedział (rev-dns i ip)
- numer kolejny zapytania/odpowiedzi
- TTL pakietu z odpowiedzią (wskazuje na ilość routerów przez które przeszedł pakiet)
- czas potrzebny na uzyskanie odpowiedzi

- podsumowujące statystyki związane z ilością zagubionych pakietów i czasami odpowiedzi

Zamiast informacji o poszczególnych odpowiedziach polecenie ping może nic nie wypisywać (brak odpowiedzi):

```
$ ping -c 1 1.5.2.1
PING 1.5.2.1 (1.5.2.1) 56(84) bytes of data.

--- 1.5.2.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

lub podawać komunikat błędu wraz z jego nadawcą (jeżeli taki został zwrócony):

```
$ ping -c1 192.168.6.55
PING 192.168.6.55 (192.168.6.55) 56(84) bytes of data.
From 192.168.6.3 icmp_seq=1 Destination Host Unreachable

--- 192.168.6.55 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Kod powrotu polecenie ping informuje o tym czy host jest osiągalny czy nie, dzięki czemu polecenie to może zostać użyte np. w bash'owym warunku if:

```
if ping -c 1 10.0.1.1 >& /dev/null; then
    echo "10.0.1.1 jest dostępny"
else
    echo "10.0.1.1 nie jest dostępny"
fi
```

Rozwiązanie zadania 3.0.2

Należy skorzystać z polecenia traceroute, tracepath, mtr lub podobnych.
Wynik polecenia host może wyglądać następująco (ale nie będzie nic dziwnego jeżeli otrzymasz inny – trasa routingowa zależy od miejsca z którego się łączysz, może też zmienić się z czasem)

```
traceroute www.example.org
traceroute to www.example.org (93.184.216.34), 30 hops max, 60 byte packets
 1  fundbox.home (192.168.6.1)  0.570ms  9.648ms  9.624 ms
 2  192.0.0.1 (192.0.0.1)  7.981ms  8.007ms  8.082 ms
 3  195.117.0.225 (195.117.0.225)  8.120ms  8.189ms  8.223 ms
 4  poz-r1.tpnet.pl (194.204.175.94)  18.474ms  18.512ms  18.547 ms
 5  hbg-b1-link.telia.net (213.248.96.144)  23.234ms  23.268ms  23.304 ms
 6  hbg-bb4-link.telia.net (213.155.135.86)  110.655ms hbg-bb3-link.telia.net
    (213.155.135.80)  104.215ms hbg-bb4-link.telia.net (213.155.135.86)  104.056
    ms
 7  * * ldn-bb4-link.telia.net (62.115.122.161)  101.809 ms
 8  * * *
 9  nyk-b6-link.telia.net (80.91.254.36)  106.471ms nyk-b6-link.telia.net
    (62.115.125.63)  104.446ms  104.562 ms
10  edgecast-ic-317660-nyk-b5.c.telia.net (62.115.147.201)  109.616ms  109.669ms
    112.148 ms
11  152.195.69.131 (152.195.69.131)  106.924ms 152.195.68.141 (152.195.68.141)
    109.509ms 152.195.69.139 (152.195.69.139)  106.711 ms
12  93.184.216.34 (93.184.216.34)  103.418ms  108.218ms  106.420 ms
13  93.184.216.34 (93.184.216.34)  106.454ms  106.587ms  110.858 ms
```

Podobnie jak w przypadku polecenia ping wypisywana jest informacja o odpytywanym adresie IP (i wynik odwrotnego dns'u dla niego), ilość wysłanych danych. Wypisywane są kolejne routery (ip i rev-dns) wraz z czasami odpowiedzi (domyślnie traceroute na każdym poziomie robi 3 zapytania). Gwiazdki oznaczają brak odpowiedzi. Jeżeli od pewnego momentu wystepują same gwiazdki oznacza

to że wraz z dalszym zwiększaniem TTL nie dostajemy już kolejnych komunikatów o jego przekroczeniu. Może to wynikać z osiągnięcia hostu docelowego, który ma zablokowany testowany port bez wysyłania komunikatu o niedostępności portu lub błędu routera który nie przekazuje poprawnie komunikatów o zbyt małym TTL.

Rozwiązanie zadania 3.0.3

Należy użyć na przykład polecenia `host www.bitbucket.org` lub poleceń `dig AAAA www.bitbucket.org` ; `dig A www.bitbucket.org`.
Wynik polecenia host może wyglądać następująco (ale nie będzie nic dziwnego jeżeli otrzymasz inne adresy – dane w DNS ulegają zmianom, niekiedy nawet dynamicznie wprowadzanym).

```
www.bitbucket.org is an alias for bitbucket.org.
bitbucket.org has address 104.192.141.1
bitbucket.org has IPv6 address 2406:da00:ff00::22c5:2ef4
bitbucket.org has IPv6 address 2406:da00:ff00::6b17:d1f5
bitbucket.org has IPv6 address 2406:da00:ff00::22e9:9f55
bitbucket.org has IPv6 address 2406:da00:ff00::34cc:ea4a
bitbucket.org mail is handled by 1 aspmx.l.google.com.
bitbucket.org mail is handled by 5 alt1.aspmx.l.google.com.
```

Warto zauważyć że zostaliśmy poinformowani także o tym że www.bitbucket.org jest aliasem (CNAME) na bitbucket.org.
Zwracanie wielu IP jest jednym ze sposobów rozkładania obciążenia i zapewnienia redundancji. Innymi rozwiązaniami jest udzielanie różnych odpowiedzi różnym kliemntom (tak robi np. www.google.com):

```
$ host www.google.com
www.google.com has address 172.217.20.164
www.google.com has IPv6 address 2a00:1450:401b:802::2004
```

z innego hosta:

```
$ host www.google.com
www.google.com has address 216.58.206.4
www.google.com has IPv6 address 2a00:1450:4001:821::2004
```

Jeszcze innym jest używanie mechanizmów routingowych takich jak anycast.

Rozwiązanie zadania 3.0.4

Należy uruchomić w osobnych oknach terminala kolejno polecenia:

1. `tcpdump -i lo -A port 5555` - podsłuchujemy komunikację używającą portu 5555
2. `nc -lp 5555` - uruchamiamy nasłuch na porcie 5555
3. `nc localhost 5555` - łączymy się na port 5555

Dane wpisywane w jednym z uruchomionych netcat'ów będą wypisywane w drugim i odwrotnie. Wszystkie te dane (wraz z informacjami zawartymi w nagłówkach IP i TCP) będą wyświetlane w okienku w którym działa tcpdump.
Oczywiście możemy użyć innego numeru portu, itd.

Rozwiązanie zadania 3.0.5

```
netcat www.opcode.eu.org 80
GET / HTTP/1.1
Host: www.opcode.eu.org

HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Sun, 26 Apr 2020 09:12:09 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 11490
Last-Modified: Wed, 08 Apr 2020 17:36:32 GMT
Connection: keep-alive
ETag: "5e8e0ba0-2ce2"
Accept-Ranges: bytes
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl" lang="pl">
<head>
                <title>OpCode.eu.org - strona główna</title>
```

[...]

Jeżeli użyjemy innego nagłówka `Host:` w zapytaniu możemy dostać np. komunikat o przekierowaniu:

```
netcat www.opcode.eu.org 80
GET / HTTP/1.1
Host: opcode.eu.org

HTTP/1.1 301 Moved Permanently
Server: nginx/1.14.2
Date: Tue, 21 Apr 2020 17:58:44 GMT
Content-Type: text/html
Content-Length: 185
Connection: keep-alive
Location: http://www.opcode.eu.org/

<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx/1.14.2</center>
</body>
</html>
```

Przeglądarki WWW nie wyświetlają tych komunikatów tylko automatycznie przechodzą pod wskazany w nagłówku `Location: adres`.

Rozwiązanie zadania 3.0.6

Wynika to z stosowania w większości protokołów sieciowych (w tym w HTTP) jako znaku linii sekwencji dwu bajtowej \n\r (nowa linia, powrót karetki). Serwer WWW obsługujący domenę opcode.eu.org postępuje według filozofii nakazującej liberalne podejście do danych otrzymywanych i restrykcyjne do generowanych przez siebie (wysyłanych) i poprawnie interpretuje formalnie błędne żądanie zawierające znaki nowej linii w postaci \n. Serwer obsługujący www.icm.edu.pl nie jest już tak liberalny i wymaga znaków nowej linii w postaci \n\r, dlatego do netcata musimy dodać opcję -C aby konwertował wprowadzone znaki nowej linii na taką postać przed wysłaniem.

Rozwiązanie zadania 3.0.7

```
ping -c2 liceali$ci.icm.edu.pl
PING www2.icm.edu.pl (213.135.59.55) 56(84) bytes of data.
64 bytes from www2.icm.edu.pl (213.135.59.55): icmp_seq=1 ttl=60 time=4.20 ms
64 bytes from www2.icm.edu.pl (213.135.59.55): icmp_seq=2 ttl=60 time=3.26 ms

--- www2.icm.edu.pl ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 3.260/3.732/4.204/0.472 ms
```

Jak widać działa. Standard kodowania dowolnych znaków Unicode, tak aby mogły być użyte w nazwach domenowych został opracowany w 2003 roku (Punycode, RFC3492), jest dość powszechnie zaimplementowany. Możemy użyć go także w pythonowej metodzie encode (np. `print( "ź$łw".encode("punycode") )`).
Ciężko powiedzieć dlaczego (przynajmniej w Polsce) znaki z poza ASCII są tak rzadko stosowane w nazwach domenowych.
Ciekawostka: kodowanie znaków nie ASCII w nazwie domenowej jest określone jednoznacznie, natomiast w dalszej części adresu URL niezbyt - zasadniczo zależy od konfiguracji serwera, ale standardem defacto jest tutaj UTF8).

Rozwiązanie zadania 5.0.1

```
ip addr add 172.33.13.113/24 dev eth5
```

Rozwiązanie zadania 5.0.2

```
ip route add 10.13.0.0/16 via 172.33.13.13
```

Rozwiązanie zadania 5.4.1

```
for f in /proc/sys/net/ipv*/conf/*/forwarding; do echo 1 > $f; done

nft add table ip filter
nft add chain ip filter FORWARD '{ type filter hook forward priority 0; }'
nft add rule  ip filter FORWARD iifname "eth3" oifname "eth4" accept
nft add rule  ip filter FORWARD iifname "eth4" oifname "eth3" accept
nft add rule  ip filter FORWARD reject
```

Rozwiązanie zadania 6.0.1

```
import socket, select, signal, sys, os

MAX_CHILD = 5
QUERY_SIZE = 3
TIMEOUT = 13
BUF_SIZE = 4096

if len(sys.argv) != 2:
    print("USAGE: " + sys.argv[0] + " listenPort", file=sys.stderr)
    exit(1);

# obsługa sygnału o zakończeniu potomka
childNum = 0
def onChildEnd(s, f):
    print("odebrano sygnał o śmierci potomka")
    global childNum
    childNum -= 1
    os.waitpid(-1, os.WNOHANG);
signal.signal(signal.SIGCHLD, onChildEnd)

# utworzenie gniazd sieciowych ... SOCK_STREAM oznacza TCP
sfd_v4 = socket.socket(socket.AF_INET,  socket.SOCK_STREAM)
sfd_v6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)

# ustawienie opcji gniazda ... IPV6_V6ONLY=1 wyłącza korzystanie
# z tego samego socketu dla IPv4 i IPv6
sfd_v6.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_V6ONLY, 1)

# przypisanie adresów ...
# '0.0.0.0' oznacza dowolny adres IPv4 (czyli to samo co INADDR_ANY)
# '::' oznacza dowolny adres IPv6 (czyli to samo co in6addr_any)
sfd_v4.bind(('0.0.0.0', int(sys.argv[1])))
sfd_v6.bind(('::',      int(sys.argv[1])))

# określenie gniazd jako używanych do odbioru połączeń przychodzących
# (długość kolejki połączeń ustawiona na wartość QUERY_SIZE)
sfd_v4.listen(QUERY_SIZE)
sfd_v6.listen(QUERY_SIZE)

# czekanie na połączenia z użyciem select() w nieskończonej pętli
while True:
    sfd, _, _ = select.select([sfd_v4, sfd_v6], [], [])
    for fd in sfd:
```

```
                #  odebranie potączenia
                sfd_c, sAddr = fd.accept()

                # weryfikacja ilości potomków
                if childNum >= MAX_CHILD:
                    print("za dużo potomków - odrzucam połączenie od:", sAddr);
                    sfd_c.send("Internal Server Error\r\n".encode())
                    sfd_c.close()
                    break

                # aby móc obsługiwać wiele połączeń rozgałęziamy proces
                pid = os.fork()
                if pid > 0:
                    # rodzic - zwiększamy liczniik potomków
                    childNum += 1
                else:
                    # potomek - obsługa danego połączenia
                    print("połączenie od:", sAddr)
                    while True:
                        # czekanie na dane z timeout'em
                        # aby zabezpieczyć się przed atakiem DoS
                        rd, _, _ = select.select([sfd_c], [], [], TIMEOUT)
                        if sfd_c in rd:
                            data = sfd_c.recv(BUF_SIZE)
                            if data:
                                print("odebrano od", sAddr, ":", data.decode());

                                # zamiast odsyłać odebrane dane poprzez sfd_c.send(data)
                                try:
                                    liczba = int(data.decode())
                                except:
                                    liczba = 0
                                if liczba < 2 or liczba > 10:
                                    sfd_c.send("Błędne polecenie - podaj liczbę >=2 i <=10.\n".encode())
                                else:
                                    trojkat = ""
                                    for i in range(1, liczba+1):
                                        trojkat += "*"*i+"\n"
                                    sfd_c.send(trojkat.encode())
                                # generujemy na ich podstawie trójkąt i go odsyłamy
                                # uzywajac sfd_c.send
                            else:
                                print("koniec połączenia od:", sAddr)
                                break
                        else:
                            print("timeout połączenia od:", sAddr)
                            break
                # zamykanie połączenia
                sfd_c.shutdown(socket.SHUT_RDWR)
                sfd_c.close()
                sys.exit()
```

Kolorowaniem kodu wyróżniono zmiane w stosunku co do przykładowego kodu ze skryptu.

Rozwiązanie zadania 6.0.2

```
import socket, sys

if len(sys.argv) != 2:
    print("USAGE: " + sys.argv[0] + " listenPort", file=sys.stderr)
    exit(1);
```

Po prostu sprawdź ... a się przekonasz że „auto-DoS" też jest możliwy.

Rozwiązanie zadania 6.0.4

```
send(IPv6(src="::1", dst="::1") / UDP(sport=8888, dport=9999) / "ABC ... XYZ")

import sys
from scapy.all import IP, IPv6, UDP, send
```

Potrzebne będą trzy okna terminala. W dwóch uruchamiamy napisaną przez nas echo usługę UDP, na przykład: `python3 echo.py 9999` i `python3 echo.py 8888`. W trzecim uruchamiamy z prawami root'a następujący kod:

Rozwiązanie zadania 6.0.3

```
sfd = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
sfd.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_V6ONLY, 0)
sfd.bind(('::', int(sys.argv[1])))

while True:
    data, sAddr, = sfd.recvfrom(4096)
    print("odebrano od", sAddr, ":", data.decode());
    sfd.sendto(data, sAddr)
```