Linux i sieci: Sieci komputerowe

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

2019-07-06

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).

1.1 Struktura warstwowa

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)

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).

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) zapisywany 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. Np. 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.

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.

1.3.2 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 *

a1  = IPv6Address("2001:0db8::17:15")
aa1 = int(a1)
print("adress IPv6 jest 128 bitową liczbą całkowitą np.: " + str(a1) + " == " + hex(aa1))

n0  = IPv6Network("::/112");
m1  = n0.netmask
mm1 = int(m1)
p1  = n0.prefixlen

print("Maska podsieci IPv6 jest 128 bitową liczbą całkowitą np.: " + str(m1) + " == " + hex(mm1))
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(p1))
print("jest to ilość bitów o wartości 1 w masce, czyli im większy prefix tym mniejsza sieć.")

n1  = IPv6Network("2001:0db8::17:15/112", strict=False);
```

```
nn1 = int(n1.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(mm1 & aa1) + " == " + str(n1) + " == " + hex(nn1))

# 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(n1, IPv6Address("2001:0db8::17:ab13"))
sprawdzSiec(n1, IPv6Address("2001:0db8::13:a"))
```

Zadanie 1.3.1

Przeanalizuj powyższy kod programu. Zapoznaj się z dokumentacją modułu ipaddress[a] i pamiętając, że adres IPv4 jest 32 bitową liczbą, zmodyfikuj ten program aby zamiast na adresach IPv6 działał na IPv4.

a. Można ją wyświetlić uruchamiając po jego zaimportowaniu: `help('ipaddress')`

1.4 Routing

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)

2 Komunikacja TCP/IP

W oparciu o protokół IP działają protokoły warstwy transportowej takie jak UDP, TCP, czy też (mniej znane protokoły czasu rzeczywistego, transmisji strumieniowych): RTP, RTCP i SCTP. 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 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).

Dodatkowo 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.

2.1 Polularne 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. Domeny posiadają budowę hierarchiczną / drzewiastą (precyzja rośnie od prawej do lewej, a kolejne poziomy oddzielane są kropkami). 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
- 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

3 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).

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.

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 obecnie szczególnie istotnej roli.

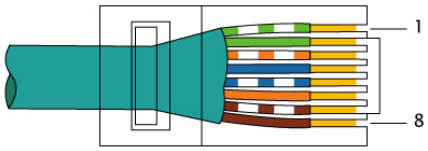
3.1 Kable

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[1].

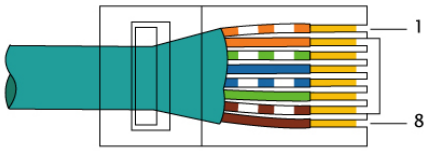
EIA/TIA 568A



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.

4 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`

1. 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 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
  - `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 bridgowego o nazwie INTERFACE
  - `ip link set SLAVE master INTERFACE` – włączenie interfejsu SLAVE w skład bridgowego INTERFACE
  - `ip link set SLAVE nomaster` - wyłaczenie interfejsu SLAVE z bridgowego
- 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
- 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

Oprócz wyżej omówionej konfiguracji interfejsów i tras routingowych, często potrzebna jest konfiguracja jądrowych mechanizmów filtracji pakietów. Służą do tego komendy:

- `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)

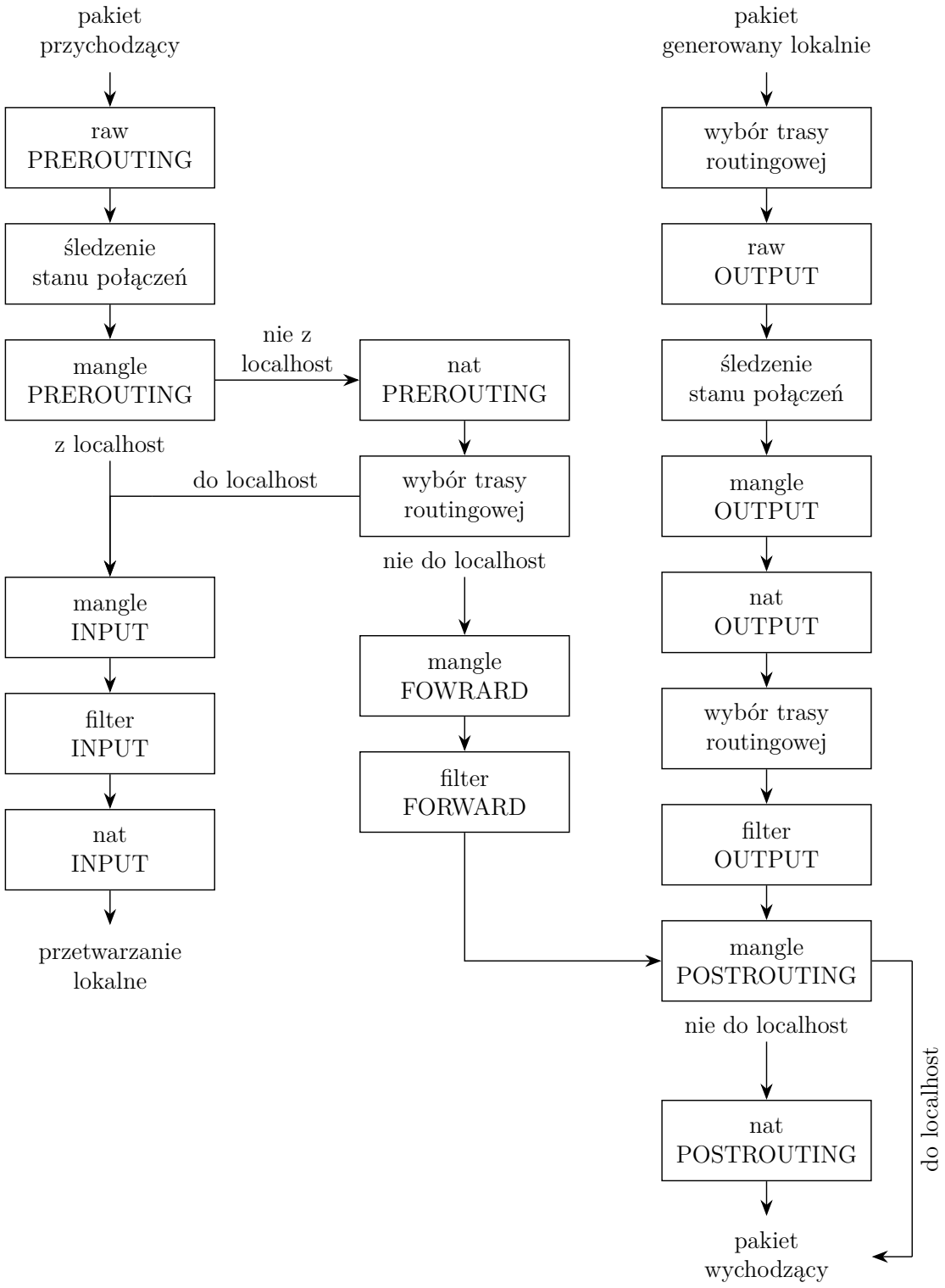
Ponadto konieczne może być dokonywanie pewnych ustawień poprzez jądrowe systemy plików `/proc` i `/sys`. Najczęstszym przypadkiem jest włączenie przekazywania pakietów pomiędzy interfejsami poprzez:

```
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)

4.1 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
- `-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 Programowanie usług sieciowych

5.0.1 wysyłanie danych po UDP

```
import socket, sys

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

dstAddrInfo = socket.getaddrinfo(sys.argv[1], sys.argv[2])
dstAddrInfo = dstAddrInfo[0]
sfd = socket.socket(dstAddrInfo[0], socket.SOCK_DGRAM)

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

5.0.2 odbiór danych po UDP

```
import socket, sys

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

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());
```

5.0.3 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 odbiór
rdfd, _, _ = select.select([sfd], [], [], 13.0)
if sfd in rdfd:
        d = sfd.recv(4096)
        print(d.decode())

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

5.0.4 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)

# funkcja zajmująca się odbieraniem połączeń i ich obsługą
def acceptConn(sfd):
        global childNum

        #  odebranie połączenia
        sfd_c, sAddr = sfd.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()
                return

        # aby móc obsługiwać wiele połączeń rozgałęziamy proces
        pid = os.fork()
        if pid == 0:
                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 not data:
                                        print("koniec połączenia od:", sAddr)
                                        break
                                print("odebrano od", sAddr, ":", data.decode());
                                sfd_c.send(data)
                        else:
                                print("timeout połączenia od:", sAddr)
                                break
                # zamykanie połączenia
                sfd_c.shutdown(socket.SHUT_RDWR)
                sfd_c.close()
```

```
                sys.exit()
        else:
                childNum += 1

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

6 Zadania domowe

Zadanie 6.0.1

Ustal czy host o adresie IPv4 192.168.65.20 należy do sieci 192.168.33.15/19.

Zadanie 6.0.2

Ustal czy host o adresie IPv6 2001:6a0:0:21::60:2 należy do sieci 2001:6a0:0:10::/58.

Zadanie 6.0.3

Ustal adresy serwerów DNS posiadających informację o domenie *gov*. Podaj polecenie którego użyłeś.

Zadanie 6.0.4

Polecenie `ip r` pokazało następującą tablicę routingu:

```
default via 192.168.29.2 dev eth0.2
192.168.29.192/27 dev eth0.2  proto kernel  scope link  src 192.168.29.193
172.16.16.0/27 via 172.16.18.2 dev tun5
172.16.16.48/28 dev wlan0  proto kernel  scope link  src 172.16.16.49
172.16.18.0/30 dev tun5  proto kernel  scope link  src 172.16.18.1
192.168.29.0/24 dev eth0  proto kernel  scope link  src 192.168.29.1
```

Ustal trasę (urządzenie którym zostanie wysłany pakiet oraz jeżeli jest potrzebny to adres routera do którego będzie przesyłany) dla następujacych adresów IP:

- 8.8.8.8
- 192.168.29.202
- 172.16.16.15

Zadanie 6.0.5

Na zajęciach omawialiśmy serwer usługi ”echo” (odsyłającej odebrane dane do nadawcy). Powyżej znajduje się przykładowy kod wysyłający dane po UDP (”klient UDP”) i odbierający dane po UDP (”serwer UDP”). W oparciu o te informacje napisz program realizujący funkcję serwera echo z użyciem UDP.
Jeżeli nie lubisz Pythona program może być w C.

Zadanie 6.0.6

Zapoznaj się z RFC1924 i napisz program konweryjący adresy IPv6 pomiędzy notacją dwukropkową a noatacją base-85 zgodną z tą specyfikacją.