# Python w Elektronicznej Sieci #9: Sieci komputerowe – konfiguracja i programowanie

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

2020-04-28

## 1 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
  - `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

Innym przydatnym poleceniem jest `tc`, które służy do konfiguracji ustawień kontoli przepływu (np. kolejkowania) na interfejsach sieciowych.

## 1.1 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).

## 1.2 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.

### 1.2.1 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`),
`-n` – 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*.

### 1.2.2 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.

## 1.3 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.

## 1.4 Filtracja pakietów

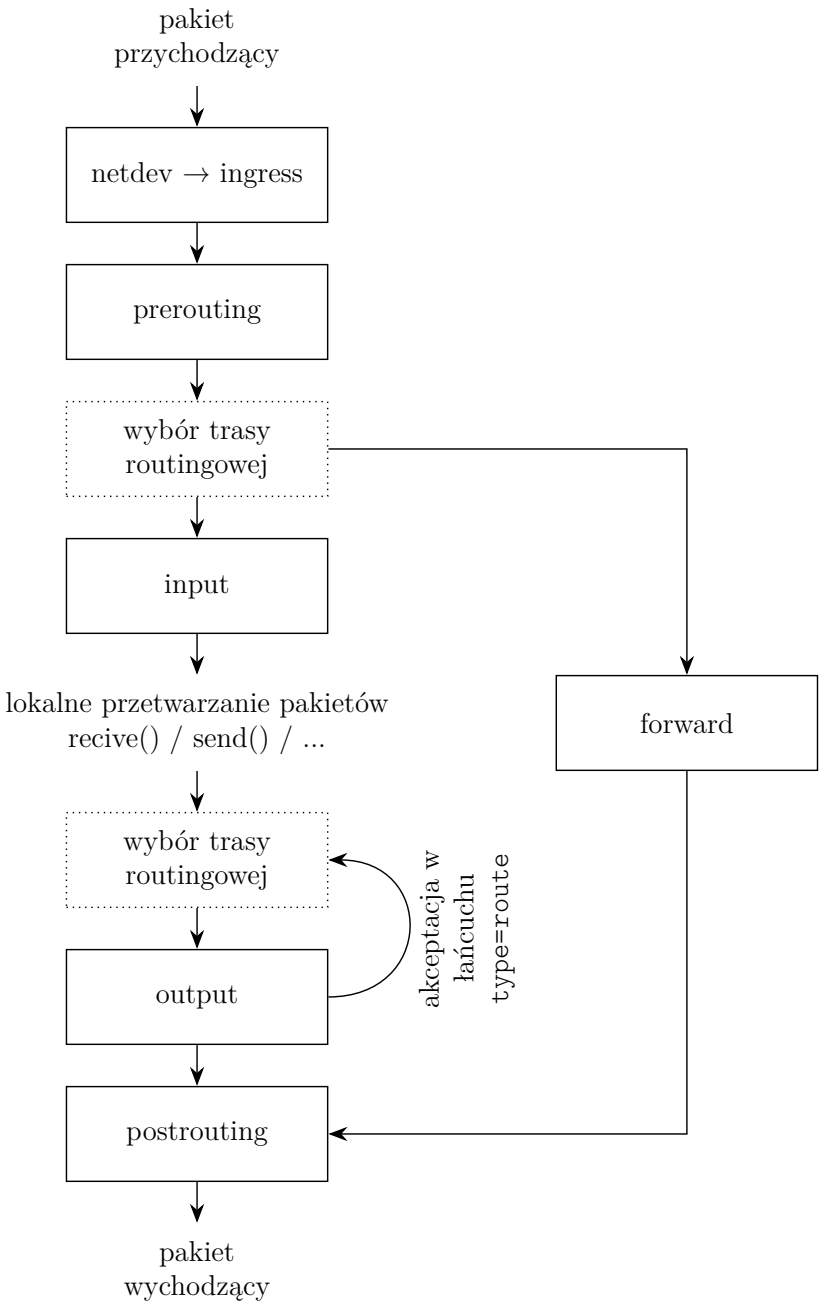
Oprócz wyżej omówionej konfiguracji interfejsów i tras routingowych, często potrzebna jest konfiguracja jądrowych mechanizmów filtracji pakietów. Służy do tego polecenie `nft`.

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)

### 1.4.1 nft (nftables)

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




Rysunek 1: 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 łańcuch 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).

### 1.4.2 iptables

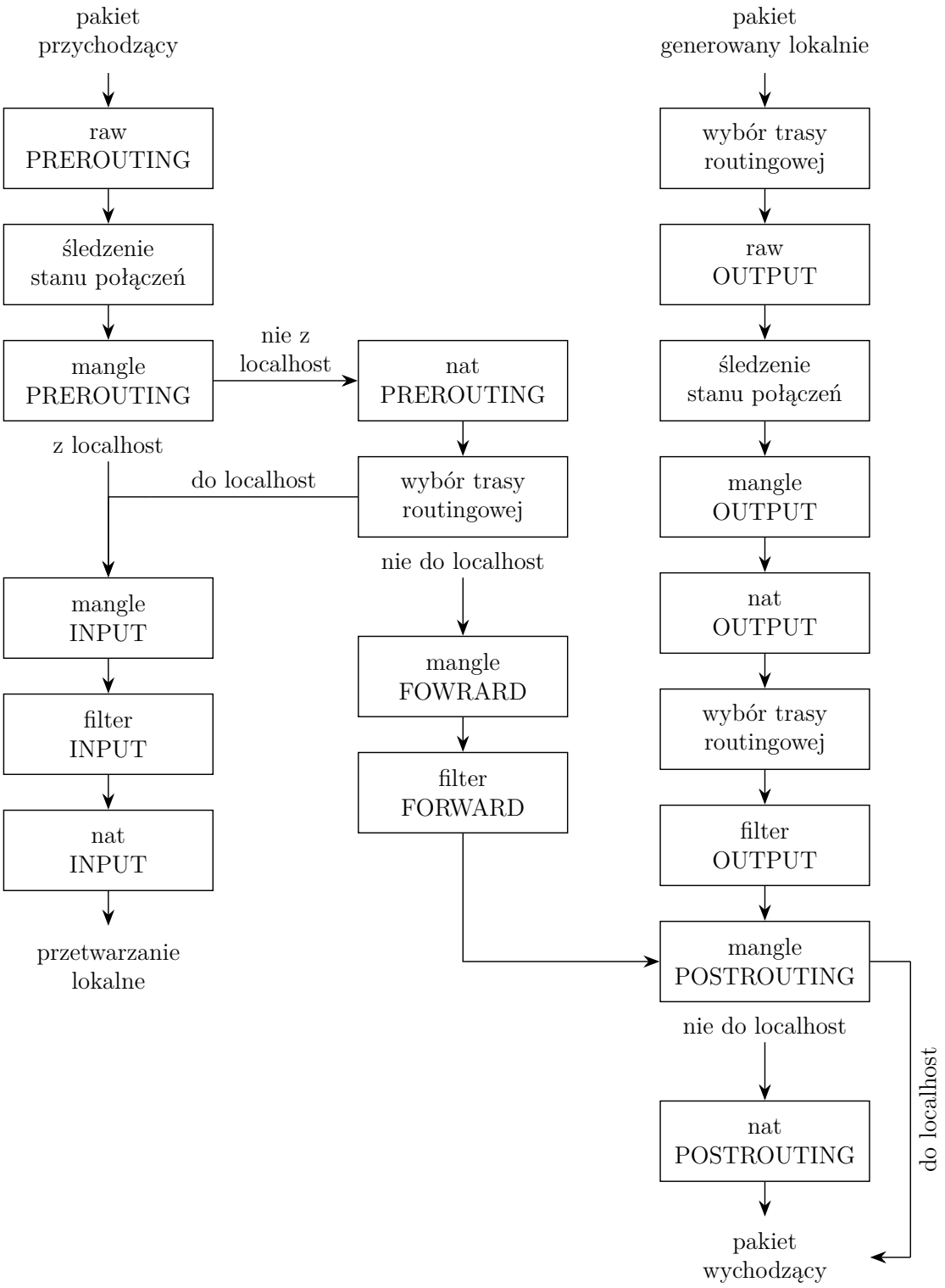
Iptables wykorzystuje kilka tablic reguł (najistotniejszymi są `filter` i `nat`). Tablica może zostać określona przy pomocy opcji `-t`, jeżeli nie użyto tej opcji operacje będą wykonywane na tablicy `filter`. Zależności pomiędzy poszczególnymi łańcuchami i tablicami przedstawia (uproszczony) diagram 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
- `-i INTERFEJS` – pasuje gdy pakiet przyszedł wskazanym interfejsem sieciowym




Rysunek 2: Trasa pakietu przez filtry iptables. Wskazano punkty zaczepień nazwy łańcuchów.

- `-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`.

# 2 Programowanie usług sieciowych

## 2.1 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])
```

## 2.2 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());
```

## 2.3 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()
```

## 2.4 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)
```

# 3 Zadania

**Zadanie 3.0.1**

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

**Zadanie 3.0.2**

Powyżej 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 program realizujący funkcję serwera echo z użyciem UDP.
*Jeżeli nie lubisz Pythona program może być w C lub C++.*

**Zadanie 3.0.3**

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 3.0.4**

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?

# 4 Zadania dodatkowe

**Zadanie 4.0.1**

Zapoznaj się z RFC1924 i napisz program konwertujący adresy IPv6 pomiędzy notacją dwukropkową a notacją base-85 zgodną z tą specyfikacją.
*Wskazówka: do odczytu adresu w standardowej notacji dwukropkowej możesz użyć narzędzi z modułu* ***ipaddress***

# 5 Praca domowa

## 5.1 Instrukcja wysyłania rozwiązań

Rozwiązania zadań domowych należy przesłać na adres ciekawi.pracownia@icm.edu.pl wpisując jako temat wiadomości `g2.x PD9`, gdzie `x` to numer grupy, np. `g2.1 PD9` dla grupy nr. 1, itd. Zadania domowe są nie obowiązkowe, jednak zachęcamy do ich robienia i wysyłania rozwiązań (nawet niekompletnych).

Termin nadsyłania zdań domowych to 2020-05-05 godzina $16^{00}$. Jeżeli wysłałeś rozwiązania w terminie, ale nie były one w 100% poprawne i dostałeś od sprawdzającego możliwość wysłania poprawki masz na to dodatkowe 4 dni.

Na ten adres można także nadsyłać ewentualne pytania do zadań (zarówno domowych jak i innych zamieszczonych w skrypcie), w tym wypadku także prosimy o umieszczenie w temacie wiadomości `g2.x`, gdzie `x` to numer grupy.

## 5.2 Zadania domowe

**Zadanie 5.2.1 — 1 pkt**

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

**Zadanie 5.2.2 — 1 pkt**

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

**Zadanie 5.2.3 — 2 pkt**

Napisz polecenia które włączą przekazywanie pakietów pomiędzy `eth3` i `eth4`, ale nie zezwolą na przekazywanie pakietów innymi interfejsami

**Zadanie 5.2.4 — 2 pkt**

Napisz serwer TCP, który oczekuje że klient wyśle mu liczbę (z zakresu od 2 do 10), 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:

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