Linux i Python w Elektronicznej Sieci #11: Sieci komputerowe – ethernet, konfiguracja i programowanie usług sieciowych

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

2023-06-25

1 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 obecnie szczególnie istotnej roli.

1.1 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 w

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 · 8 – 1500 · 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 · 8 – 1500 · 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.

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.

1.2 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.

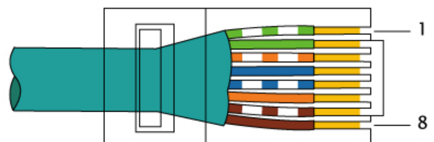
1.3 Kable

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[1].

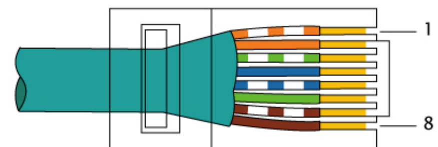
EIA/TIA 568A



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.

2 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

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 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 2.0.1

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

Zadanie 2.0.2

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

2.1 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).

2.2 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.

2.2.1 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*.

2.2.2 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.

2.3 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/.

2.4 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).

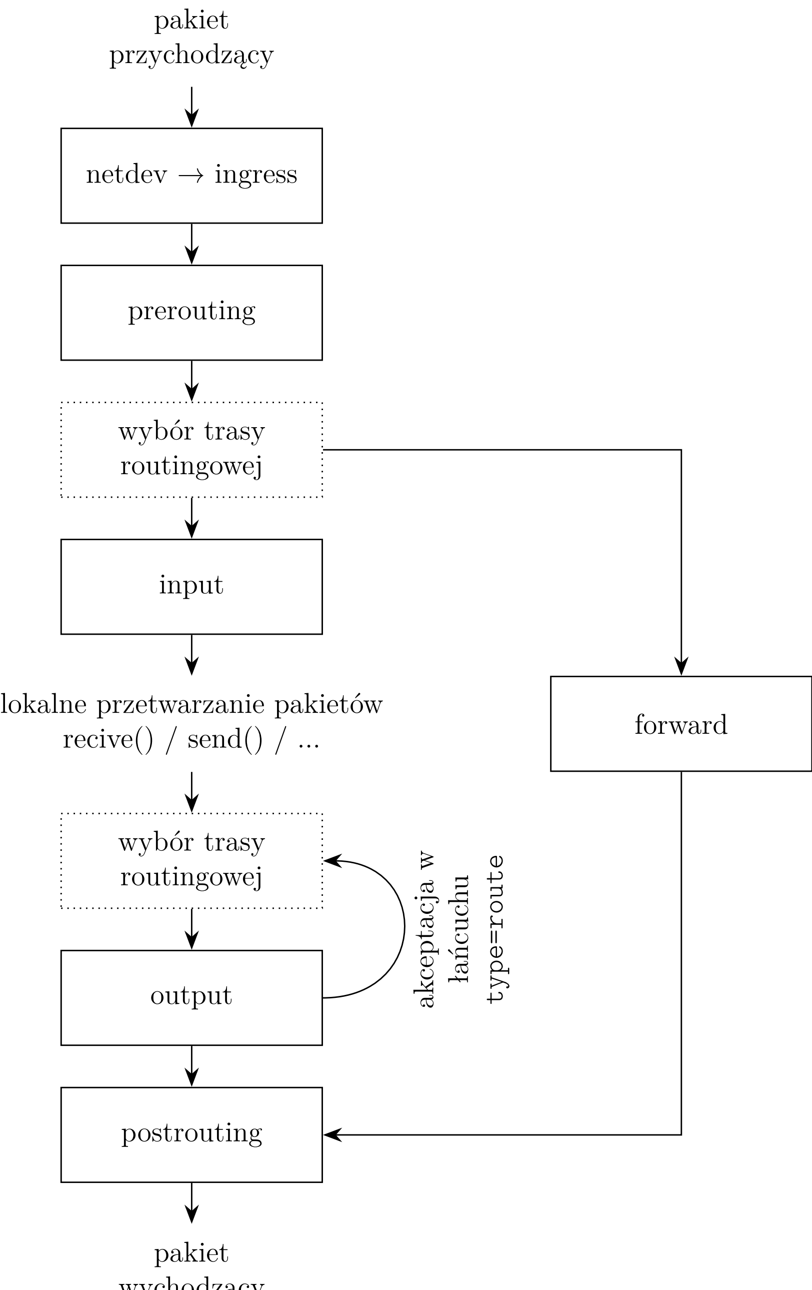
Do konfiguracji filtracji pakietów służy polecenie `nft`. Na starszych systemach `nft` może być niedostępny, wtedy można korzystać z poleceń:

- `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.

2.4.1 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 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[2]:

```
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'
```

2. 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 rozdzielone 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 zastosować 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 2.4.1

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

2.5 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.

3 Programowanie usług sieciowych

3.1 wysyłanie danych po UDP

```python
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])
```

3.2 odbiór danych po UDP

```python
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());
```

3.3 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***

3.4 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 3.0.1

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 3.0.2

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 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 Wykład wideo[3]

- *Ethernet* – http://video.opcode.eu.org/11.01.mkv

3. 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`

- *Standardy sieciowe* – http://video.opcode.eu.org/11.02.mkv
- *Konfiguracja sieci* – http://video.opcode.eu.org/11.03.mkv
- *Programowanie usług sieciowych* – http://video.opcode.eu.org/11.04.mkv

5 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
- *Vademecum informatyki praktycznej* (http://vip.opcode.eu.org/) - zbiór materiałów na temat elektroniki i programowania, zawierający także dość rozbudowaną listę literatury dodatkowej.
- *Linux i Python w Elektronicznej Sieci* (https://ciekawi.icm.edu.pl/lpes) - strona domowa kursu *LPES*, zawierająca nagrania i skrypty do innych wykładów, skrypty ćwiczeniowe, itd.
- *OpCode.eu.org* (http://vip.opcode.eu.org/) - strona internetowa autora kursu *LPES*, zawierająca różne materiały z szeroko rozumianej inżynierii komputerowej i elektronicznej (część materiałów pokrywa się z zawartością skryptów z tego kursu, ale nie wszystkie)

6 Zadania

Poniższe zadania znajdują się także w odpowiednich rozdziałach skryptu. Zostały jednak zamieszczone zbiorczo także w tym miejscu dla wygody czytelnika.

Zadanie 2.0.1

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

Zadanie 2.0.2

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

Zadanie 2.4.1

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

Zadanie 3.0.1

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 3.0.2

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 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?

7 Rozwiązania zadań

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 2.0.1

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

Rozwiązanie zadania 2.0.2

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

Rozwiązanie zadania 2.4.1

```
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 3.0.1

```
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 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());

                        # zamist 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
                         ↪  używając 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 zmianę w stosunku co do przykładowego kodu ze skryptu.

Rozwiązanie zadania 3.0.2

```
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());
    sfd.sendto(data, sAddr)
```

Rozwiązanie zadania 3.0.3

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:

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

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

Rozwiązanie zadania 3.0.4

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

© Matematyka dla Ciekawych Świata, 2017-2023.
© Robert Ryszard Paciorek <rrp@opcode.eu.org>, 2003-2023.
Wykorzystano grafiki należące do domeny publicznej.
Kopiowanie, modyfikowanie i redystrybucja dozwolone pod warunkiem zachowania informacji o autorach.