Agnieszka CHODOREK, Robert CHODOREK

ZASTOSOWANIE EMULATORA NETKIT DO DYDAKTYKI RUTINGU IPV4 I IPV6 NA PRZYKŁADZIE PROTOKOŁÓW RIP I RIPNG

Streszczenie

Niniejszy artykuł jest poświęcony zastosowaniu emulatora sieci komputerowych Netkit do dydaktyki sieci komputerowych w wyższych szkołach technicznych. Jest to oprogramowanie narzędziowe umożliwiające konfigurowanie i testy złożonych sieci bez konieczności użycia specjalizowanego sprzętu sieciowego. Sam emulator ma niskie wymagania sprzętowe. Na przykładzie realizacji ćwiczeń laboratoryjnych z protokołu RIP (w wersji dla IPv4 jak i dla IPv6) przybliżono Czytelnikowi sposób tworzenia topologii testowej i konfigurowania emulowanych ruterów i komputerów końcowych. Przedstawione w artykule możliwości zastosowania emulatora nie ograniczają się tylko do dydaktyki lecz również mogą być wykorzystane do prowadzenia badań naukowych z zakresu protokołów rutingu wewnętrznego.

WSTĘP

Emulator Netkit [1][6] jest bezpłatnym narzędziem programistycznym, publicznie dostępnym pod adresem www.netkit.org, przeznaczonym głównie do dydaktyki w wyższych szkołach technicznych. Obecnie jest on wykorzystywany w co najmniej 15 różnych instytucjach edukacyjnych na całym świecie [5]. Program umożliwia tworzenie maszyn wirtualnych połączonych emulowaną siecią o dostępie rywalizacyjnym (domeną kolizyjną), widzianą przez maszynę wirtualną jako sieć Ethernet.

Maszyna wirtualna, uruchamiana w ramach emulatora Netkit, pracuje pod kontrolą systemu operacyjnego Linux i może pełnić rolę stacji lub rutera. W tym ostatnim przypadku, może korzystać z popularnych pakietów oprogramowania Zebra (obecnie już nierozwijanego) lub Quagga. Pakiet oprogramowania Zebra/Quagga udostępnia szereg protokołów rutingu wewnętrznego, jak RIP, RIPng, OSPF w wersjach 2 i 3, IS-IS, PIM-SM, oraz protokół rutingu zewnętrznego BGP.

Niniejszy artykuł przedstawia możliwości zastosowania emulatora Netkit do dydaktyki rutingu IPv4 i IPv6 na przykładzie protokołów RIP i RIPng. Składa się on z trzech rozdziałów. Rozdział pierwszy poświęcony jest protokołowi RIP i jego odpowiednikowi dla IPv6, RIPng. Rozdział drugi omawia program Netkit i jego zastosowanie jako środowisko testowe w dydaktyce rutingu IPv4 i IPv6. W rozdziale trzecim pokazano przykładowe ćwiczenie laboratoryjne realizowane z wykorzystaniem emulatora Netkit.

1. PROTOKOŁY RIP I RIPNG

Protokół RIP (z ang. Routing Information Protocol) został zdefiniowany w dokumencie RFC 1058 [2], wydanym przez IETF w czerwcu 1988 r i zaktualizowany w listopadzie 1998 do wersji 2 dokumentem RFC 2453 [3]. Obie wersje protokołu RIP są przeznaczone do pracy z protokołem IP w wersji 4 i mogą ze sobą współpracować. Protokół RIP w wersji 1 może być implementowany zarówno w stacjach, jak i ruterach, przy czym stacje działają zawsze w trybie pasywnym, czyli nie wysyłają aktualizacji tablic rutingu.

W wersji 2 protokołu RIP dodano mechanizm uwierzytelnienia komunikatów rutingu. Wersja ta posiada również możliwość rutingu bezklasowego, na podstawie masek podsieci. Ułatwia wykorzystanie wielu ruterów w sieci LAN.

Protokół RIPng (z ang. Routing Information Protocol - new generation) Zdefiniowany dokumentem RFC 2080 [4] w styczniu 1997 roku. Protokół ten jest wersją RIP współpracującą z protokołem IP w wersji 6. Posiada on pełną funkcjonalność protokołu RIPv2.

Protokół RIP działa w oparciu o zasadę wektora odległości. Ruter RIP tworzy tablicę rutingu, w której dla każdej sieci (widzianej jako adres własny sieci i maska) wpisuje odległość, mierzoną liczbą przeskoków (ruterów), przy czym odległość nie może być większa niż 15 (liczba 16 oznacza "nieskończoność"). Powstała w ten sposób lista tras i kosztów ich osiągnięcia jest optymalizowana pod kątem kosztu. Jeżeli pojawią się dwie konkurencyjne trasy to tylko trasa najkrótsza, o najmniejszym koszcie, zostanie zapisana w tablicy rutingu. Pozwala to zarówno na wyeliminowanie dłuższych tras, jak i na uniknięcie zapętleń tablic rutingu.

Komunikaty RIP zawierające aktualne tablice rutingu są okresowo (co pół minuty) rozsyłane do tych wszystkich sąsiadów, którzy są bezpośrednio osiągalni. Na ich podstawie bezpośredni sąsiedzi aktualizują swoje tablice rutingu. Jeżeli przez 180 sekund nie nadejdzie komunikat od sąsiadującego rutera, trasa uznawana jest za nieaktualną. Same komunikaty nie są retransmitowane przez inne rutery. Do transmisji swoich komunikatów RIP wykorzystuje protokół transportowy UDP. Same komunikaty w wersji 1 protokołu RIP są rozsyłane rozgłoszeniowo, dzięki czemu nie są wysyłane poza lokalną podsieć. Wersja 2 protokołu RIP rozsyła swoje komunikaty multikastowo.

2. PROGRAM EMULACYJNY NETKIT JAKO ŚRODOWISKO TESTOWE W DYDAKTYCE RUTINGU IPV4 I IPV6

Emulator Netkit służy do tworzenia i uruchamiania maszyn wirtualnych pracujących pod kontrolą systemu operacyjnego Linux oraz do emulacji połączeń sieciowych między tymi maszynami. Ponieważ maszyny wirtualne uruchamiane są w trybie UML (ang. User Mode Linux), system operacyjny gospodarza (system operacyjny maszyny fizycznej) traktuje system operacyjny gościa (system operacyjny maszyny wirtualnej) jak każdy inny proces użytkownika. Tym samym system gościa korzysta z zasobów sprzętowych i programowych swojego gospodarza (wirtualizacja typu drugiego), co z kolei sprawia, że sam emulator posiada niewielkie wymagania sprzętowo-programowe. Przykładowo, w pracy [6] podano, iż przy 2

GB RAM i procesorze PIV można wygenerować 200 systemów wirtualnych.

Tak zbudowana wirtualna sieć komputerowa opiera się na pojęciu domeny kolizyjnej. Domenę kolizyjną należy rozumieć jako rodzaj wirtualnego koncentratora ruchu, do którego podłączane są poszczególne systemy wirtualne, zarówno końcowe (stacje), jak i pośredniczące (rutery). Do komunikacji w warstwie sieciowej wykorzystywany jest protokół IP w wersji obsługiwanej przez system gospodarza. W praktyce oznacza to, że jest możliwa zarówno praca z IP w wersji 4 (IPv4), jak i z IP w wersji 6 (IPv6). Domyślną wersją protokołu IP jest wersja 4. Podobnie, jak w przypadku maszyn fizycznych, adresy IP przydzielane są interfejsom maszyny wirtualnej.

```
pc1:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 6e:5f:98:37:0c:07
          inet addr:80.26.1.5  Bcast:80.26.1.255  Mask:255.255.255.0
          inet6 addr: fe80::6c5f:98ff:fe37:c07/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:167 errors:0 dropped:0 overruns:0 frame:0
          TX packets:30 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11312 (11.0 KiB)  TX bytes:2372 (2.3 KiB)
          Interrupt:5

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:756 (756.0 B)  TX bytes:756 (756.0 B)
```

Rys. 1. *Interfejs eth0 komputera pc1 z rysunku 2a*

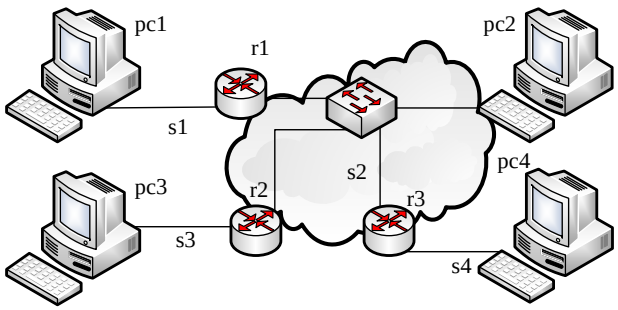
Nadawanie interfejsowi adresu IP w wersji 4 odbywa się z wykorzystaniem polecenia `ifconfig`. Do nadania interfejsowi adresu IP w wersji 6 można wykorzystać polecenie IPv6 z ustawioną opcją `address`. Interfejs otrzymuje również automatycznie nadany adres miejscowy IPv6 i maskę dla adresacji miejscowej (na rysunku 1[1] są to `fe80::6c5f:98ff:fe37:c07/64`). Adres miejscowy tworzony jest na bazie adresu karty sieciowej. Adres widoczny na rysunku 1 powstał z adresu karty sieciowej `6e:5f:98:37:0c:07` poprzez dodanie `fffe` w środku adresu i negację siódmego bitu (w pierwszym bajcie). Jak widać na rysunku 1, zamiast `6e` (`0110 1110`), widocznego w adresie karty sieciowej, jest `6c` (`0110 1100`). Na rysunku 1 widać również m.in. adres rozgłoszeniowy IP w wersji 4 (`80.26.1.255`) oraz adresy własne interfejsów (loopback), `127.0.0.1` dla IP w wersji 4 i `::1/128` dla IP w wersji 6.

Zarządzanie eksperymentem emulacyjnym może mieć charakter indywidualny lub zbiorczy [1][6]. W zarządzaniu indywidualnym, takie czynności jak powoływanie, wstępne konfigurowanie oraz usuwanie nowych instancji systemów końcowych i pośredniczących realizowane są z poziomu konsoli. Jest to praca tzw. w trybie "v", z v-poleceniami, czyli poleceniami rozpoczynającymi się na literę v jak "virtual [machine]" (z ang.: [maszyna] wirtualna). W zarządzaniu zbiorczym nie zarządza się pojedynczą maszyną wirtualną lecz wirtualnym laboratorium jako całością. Powoływanie, wstępne konfigurowanie oraz usuwanie laboratorium realizowane jest z poziomu konsoli Netkit. Jest to praca tzw. w trybie "l", z l-poleceniami, czyli poleceniami rozpoczynającymi się na literę l jak "laboratory" (z ang.: laboratorium.).

Opis laboratorium zarządzanego w trybie zbiorczym znajduje się w pliku lab.conf (przykład takiego pliku znajduje się na rysunku 2b). Plik lab.conf zawiera opis maszyn wirtualnych zarządzanych w ramach laboratorium oraz topologię sieci łączącej te maszyny. Opis maszyn wirtualnych zawiera ich nazwy (etykiety maszyn) oraz konfigurację sprzętową. Jeżeli nie podano inaczej, przyjmowane są domyślne ustawienia konfiguracyjne, przechowywane w pliku netkit.conf. Możliwa jest również automatyczna konfiguracja maszyn wchodzących w skład laboratorium. Pliki konfigurujące maszyn mają rozszerzenie startup, a ich nazwy są nazwami (etykietami) maszyn wirtualnych. Skrypt zawarty w pliku konfiguracyjnym *.startup jest uruchamiany na maszynie wirtualnej wskazanej nazwą pliku, tuż po rozpoczęciu pracy tej maszyny. Szczególnym przypadkiem pliku *.startup jest plik shared.startup, zawierający skrypt wspólny dla wszystkich maszyn wirtualnego laboratorium.

Opis topologii sieci łączącej maszyny wirtualne wiąże się z koniecznością zdefiniowania nazw (etykiet) sieci wirtualnych (domen kolizyjnych). Jest to realizowane równocześnie ze wskazaniem interfejsów maszyn podłączonych do tychże sieci. Format zapisu jest taki sam, jak w przypadku każdego innego parametru. Podawany jest identyfikator maszyny wirtualnej, po czym w nawiasie kwadratowym podawana jest nazwa parametru, a po znaku równości wartość tego parametru (rys. 2b). Podczas tworzenia opisu topologii sieci, parametrem jest numer N interfejsu ethN danej maszyny

(a)



(b)

```
pc1[0]=s1
pc2[0]=s2
pc3[0]=s3
pc4[0]=s4

r1[0]=s1
r1[1]=s2
r2[0]=s2
r2[1]=s3
r3[0]=s2
r3[1]=s4
```

Rys. 2. *Przykładowe środowisko testowe: a) topologia sieci testowej, b) zawartość pliku konfiguracyjnego emulatora Netkit (plik `lab.conf`) opisującego sieć zobrazowaną na rysunku 2a*

[1] Polecenie ifconfig oznacza adresy IPv4 etykietą inet, a adresy IPv6 etykietą inet6.

wirtualnej. Jako wartość parametru podawana jest nazwa sieci wirtualnej, do której podłączony jest interfejs ethN.

(a)

```
!
hostname R1_ripngd
password root
enable password root
!
router ripng
```

(b)

```
r1:~# usr/lib/quagga/ripngd -d
```

(c)

```
r1:~# sysctl -w net.ipv6.conf.all.forwarding=1
net.ipv6.conf.all.forwarding = 1
r1:~#
```

Rys. 3. *Uruchamianie rutingu IPv4 i IPv6: a) zawartość pliku konfiguracyjnego ripngd.conf, b) uruchamianie modułów Zebry i RIP, c) włączenie rutingu IPv6*

Na rysunku 2a przedstawiony został przykład środowiska testowego, składającego się z czterech sieci wirtualnych (s1, s2, s3 i s4) oraz siedmiu maszyn wirtualnych, z których cztery (pc1, pc2, pc3 i pc4) pełnią rolę stacji podłączonych do sieci, a trzy (r1, r2, r3) ruterów. Każda z sieci wirtualnych (domen kolizyjnych) składa się z wirtualnego koncentratora, do którego podłączone są odpowiednie interfejsy ruterów i stacji roboczych. Na rysunku 2a pokazano budowę wewnętrzną sieci s2 (zaznaczonej schematycznie w postaci chmury). Na schemacie widoczny jest wirtualny koncentrator, do którego podłączone są interfejsy komputera pc2 oraz interfejsy ruterów r1, r2 i r3. W przypadku sieci s1, s3, i s3, które są wirtualnymi łączami punkt-punkt, budowę wewnętrzną sieci pominięto, aby nie zaciemniać rysunku. Identyfikatory interfejsów zostały doprecyzowane w pliku konfiguracyjnym laboratorium (rys. 2b). I tak, komputer pc2 oraz rutery r2 i r3 łączą się z siecią s2 przez swoje interfejsy eth0, a ruter r1 przez interfejs eth1. Parametry konfiguracyjne poszczególnych sieci wirtualnych (adresy własne i maski dla IPv4 i IPv6) oraz parametry konfiguracyjne stacji i ruterów znajdują się, odpowiednio, w tabelach tab. 1, tab. 2 i tab. 3.

Aby uruchomić ruter na maszynie wirtualnej, należy przygotować pliki startowe Zebry (plik zebra.conf) i RIP lub RIPng (odpowiednio, plik ripd.conf lub plik ripngd.conf). Pliki te należy umieścić w katalogu etc/quagga każdego rutera. Przykładowa zawatość pliku ripngd.conf dla rutera R1 z rysunku 2a została zamieszczona na rysunku 3a. W kolejnym kroku należy uruchomić moduł Zebry i obsługę protokołu RIP w trybie demona (opcja -d na rysunku 3b).

3. WIZUALIZACJA PRACY PROTOKOŁÓW RIP I RIPNG Z WYKORZYSTANIEM EMULATORA NETKIT

Emulator Netkit może zostać wykorzystany do zobrazowania pracy protokołów rutingu. Zostało to wykorzystane, między innymi, podczas opracowywania ćwiczeń laboratoryjnych prowadzonych na Politechnice Świętokrzyskiej w ramach przedmiotu "Wybrane zagadnienia routingu". W ramach ćwiczeń, Studenci przygotowują konfigurację złożonej sieci teleinformatycznej. Topologia i parametry konfiguracyjne sieci są podawane przez prowadzącego i zależą, między innymi, od stopnia zaawansowania grupy studenckiej. Przykład prostego środowiska testowego został podany na rysunku 2 i w tabelach 1, 2, 3. Aby ćwiczenie było interesujące, liczba ruterów sieci testowej nie powinna być mniejsza od 3, zaś dla każdej ścieżki całkowita liczba ruterów nie powinna przekraczać 15. Zespoły studenckie samodzielnie opracowują pliki konfiguracyjne emulatora Netkit pracującego w trybie "l".

Podstawowe ćwiczenie laboratoryjne ma na celu zapoznanie studentów z możliwościami konfigurowania protokołów rutingu wewnętrznego RIP i RIPng oraz obserwację pracy protokołu RIP. W pierwszym kroku studenci łączą się z modułem zebra za pomocą usługi telnet (rys. 4a: polecenie telnet z parametrami localhost oraz zebra). Nazwa zebra jest tu predefiniowanym numerem portu dla obsługi modułu Zebra. Logowanie odbywa się z wykorzystaniem hasła podanego wcześniej w pliku konfiguracyjnym zebra.conf. Po zalogowaniu studenci zapoznają się z dostępnymi poleceniami

Tab. 1. *Przykładowe parametry konfiguracyjne sieci o topologii przedstawionej na rysunku 2a*

identyfikator sieci IPv4 lub IPv6	protokół IP w wersji 4		protokół IP w wersji 6	
	adres własny sieci	maska sieci	adres własny sieci	maska sieci
s1	80.26.1.0	/24	3ffe:1:1:1::	/64
s2	80.26.2.0	/24	3ffe:1:1:2::	/64
s3	80.26.3.0	/24	3ffe:1:1:3::	/64
s4	80.26.4.0	/24	3ffe:1:1:4::	/64

Tab. 2. *Przykładowe parametry konfiguracyjne stacji pokazanych na rysunku 2a*

identyfikator stacji	identyfikator sieci	adres IPv4 stacji	adres IPv6 stacji
pc1	s1	80.26.1.5	3ffe:1:1:1::5
pc2	s2	80.26.2.7	3ffe:1:1:2::7
pc3	s3	80.26.3.18	3ffe:1:1:3::18
pc4	s4	80.26.4.34	3ffe:1:1:4::34

Tab. 3. *Przykładowe parametry konfiguracyjne ruterów pokazanych na rysunku 2a*

identyfikator rutera	adres IP interfejsu eth0		adres IP interfejsu eth1	
	IPv4	IPv6	IPv4	IPv6
r1	80.26.1.1	3ffe:1:1:1::1	80.26.2.1	3ffe:1:1:2::1
r2	80.26.2.2	3ffe:1:1:2::2	80.26.3.1	3ffe:1:1:3::1
r3	80.26.2.3	3ffe:1:1:2::3	80.26.4.1	3ffe:1:1:4::1

pakietu Zebra, ze szczególnym uwzględnieniem polecenia show. Tę część ćwiczenia kończy wyświetlenie tablicy rutingu w stylu "uproszczonego Cisco" (rys. 4b). Studenci kończą pracę z usługą telnet za pomocą polecenia exit.

(a)

```
r1:~# telnet localhost zebra
Trying 127.0.0.1...
Connected to r1.
Escape character is '^]'.

Hello, this is Quagga (version 0.99.10).
Copyright 1996-2005 Kunihiro Ishiguro, et al.


User Access Verification

Password:
R1> enable
Password:
R1#
```

(b)

```
R1# show ip route
Codes: K - kernel route, C - connected, S - static, R -
       I - ISIS, B - BGP, > - selected route, * - FIB ro

C>* 80.26.1.0/24 is directly connected, eth0
C>* 80.26.2.0/24 is directly connected, eth1
C>* 127.0.0.0/8 is directly connected, lo
R1#
```

Rys. 4. *Konfigurowanie modułu Zebra (protokół IPv4): a) łączenie się z modułem zebra, b) wynik wykonania polecenia* `show ip route` *przez ruter R1*

Tab. 4. *Oznaczenia kodowe źródeł informacji rutingowej - moduł Zebra*

Kod	Legenda (w jęz. angielskim)	Opis
K	kernel route	trasy utworzone przez jądro Linuxa
C	connected	sieci miejscowe rutera
S	static	trasy statyczne
R	RIP (RIPng)	trasy pozyskane przez protokół RIP (RIPng)
O	OSPF	trasy pozyskane przez protokół OSPF
I	ISIS	trasy pozyskane przez protokół ISIS
B	BGP	trasy pozyskane przez protokół BGP

Protokoły RIP i RIPng są pierwszymi protokołami rutingu poznawanymi przez studentów na zajęciach z przedmiotu "Wybrane zagadnienia routingu", w związku z czym podczas realizacji ćwiczeń laboratoryjnych duży nacisk jest kładziony na poznawanie reprezentacji tablic rutingu, gromadzących informacje o trasach prowadzących do poszczególnych sieci. Analiza zawartości tablic rutingu prowadzona jest po skonfigurowaniu każdego rutera. Obserwowane są zarówno tablice rutingu nowopodłączonego rutera, jak i wpływ wprowadzenia do sieci nowego rutera na tablice rutingu pozostałych węzłów.

Przykładowa tablica rutingu, wyświetlona w wyniku wykonania polecenia `show ip route` przez ruter R1, została zamieszczona na rysunku 4b. Każdy wpis do tablicy rozpoczyna się polem znaczników, z których pierwszy jest jednoliterowym kodem informującym, z jakiego źródła ruter pozyskał informację o danej trasie. Objaśnienia oznaczeń kodowych źródeł informacji rutingowej zostały zamieszczone w tablicy 4.

Kolejny znacznik, znak ">" (znak większości), wskazuje na aktualnie wybraną trasę (ang. *selected route*) prowadzącą do danej sieci. Jeżeli do danej sieci prowadzi kilka tras alternatywnych, znacznik wskazuje, która z nich jest aktualnie wykorzystywana. Jeżeli dana trasa alternatywna nie jest aktualnie wykorzystywana, drugi znacznik jest znacznikiem pustym, a na ekranie wyświetlony zostanie znak spacji.

Znacznik "*" (gwiazdka) oznacza trasę pozyskaną z tablicy FIB (ang. *FIB route*). Tablica FIB (ang. *Forwarding Information Base*) jest tablicą rutingu utrzymywaną przez jądro systemu operacyjnego Linux. Poszczególne protokoły rutingu utrzymują własne tablice rutingu, z których wpisy są eksportowane do tablicy rutingu modułu Zebra, zwanej RIB (ang. *Routing Information Base*). Jest to zbiorcza tablica demona rutingu zebra. FIB zawiera trasy, na podstawie których jądro systemu operacyjnego Linux ma podjąć decyzję o rutingu, czyli tylko najlepsze trasy. W RIB natomiast mogą być przechowywane informacje o wszystkich trasach, łącznie z trasami alternatywnymi. W praktyce, najnowsze wpisy z tablic rutingu protokołów są przekazywane do tablicy RIB, natomiast zebra eksportuje z RIB do FIB tylko najlepsze trasy. Na FIB dodatkowo można wpływać narzędziami systemowymi, natomiast na zawartość RIB mają wpływ jedynie protokoły rutingu, ruting statyczny i, ogólnie, konfiguracja demona rutingu zebra.

Legenda wyjaśniająca znaczenie poszczególnych znaczników jest wyświetlana bezpośrednio przed informacjami pozyskanymi z tablicy rutingu (na rys. 4b legenda jest widoczna tylko częściowo).

Bezpośrednio po trzech znacznikach (źródła, używalności, FIB) wyświetlany jest adres bazowy sieci docelowej, wraz z maską podawaną w formacie skróconym. W następnej kolejności podawana jest informacja, do którego interfejsu rutera jest podłączona sieć docelowa, ewentualnie przez który (słowo kluczowe: *via*) interfejs jest ona osiągalna oraz czas, jaki upłynął od chwili dodania wpisu). Jak widać na rysunku 4b, na obecnym, wstępnym etapie realizacji ćwiczenia laboratoryjnego tablica rutera R1 zawiera jedynie wpisy tras prowadzących do sieci miejscowych (znacznik "C"), czyli sieci bezpośrednio podłączonych do rutera. Są to:

- trasa do sieci s1 (adres własny: 80.26.1.0), podłączonej bezpośrednio do rutera R1 przez interfejs eth0,
- trasa do sieci s2 (adres własny: 80.26.2.0), podłączonej bezpośrednio do rutera R1 przez interfejs eth1,
- trasa do sieci wirtualnej utworzonej na bazie adresu zwrotnego stacji (adres własny: 127.0.0.0), osiągalnej przez interfejs zwrotny (ang. *loopback, lo*).

Trasy te są aktualnie wykorzystywane (znacznik ">") i w tablicy FIB (znacznik "*"). Wpisy o trasach prowadzących do sieci bezpośrednio podłączonych do rutera są ważne bezterminowo.

(a)

```
R1_ripd# configure terminal
R1_ripd(config)#
```

(b)

```
R1_ripd(config)#
R1_ripd(config)# router rip
R1_ripd(config-router)#
R1_ripd(config-router)#
```

(c)

```
R1_ripd(config-router)# network 80.26.1.0/24
R1_ripd(config-router)# network 80.26.2.0/24
R1_ripd(config-router)# exit
```

(d)

```
R1_ripd(config-router)# version 2
R1_ripd(config-router)#
```

Rys. 5. *Konfigurowanie RIP: a) wynik polecenia* `configure terminal`*, b) wynik polecenia* `router rip`*, c) polecenia* `network` *i* `exit`*, d) wynik polecenia* `version 2`

W drugim kroku studenci łączą się poprzez telnet z demonem protokołu RIP (polecenie telnet z parametrami `localhost` oraz

ripd), podając hasło zapisane w pliku konfiguracyjnym ripd.conf. Po wydaniu polecenia configure terminal i zwróceniu uwagi na zmianę kontekstu (rys. 5a), studenci wchodzą do konfiguracji demona protokołu RIP (rys. 5b). Następnie dodają do tablicy rutingu wszystkie sieci miejscowe i kończą pracę z usługą telnet (rys. 5c). Sieciami miejscowymi rutera R1 z rys. 2a są sieci s1 (80.26.1.0/24) i s2 (80.26.2.0/24). Zmiana wersji protokołu RIP z domyślnej RIPv1 na RIPv2 realizowana jest z poziomu konfiguracji demona ripd, za pomocą polecenia version (rys. 5d).

(a)

```
R1(config-if)# ipv6 nd prefix 3ffe:1:1:1::/64
```

(b)

```
R1(config-if)# no ipv6 nd suppress-ra
R1(config-if)# ipv6 nd ra-interval 10
```

(c)

```
R1(config-if)# ipv6 address 3ffe:1:1:1::5/64
```

Rys. 6. *Konfigurowanie modułu Zebra (protokół IPv6): a) prefiks sieci i maska, b) załączenie protokołu ND, c) adres interfejsu*

W przypadku IPv6 ćwiczenie przebiega w dwóch podobnych krokach. Studenci powinni zwrócić uwagę na podobieństwa i różnice w konfigurowaniu modułu Zebra dla IPv6 oraz na to, że rozgłaszanie prefiksów sieci można wykonać z poziomu modułu Zebra. Po zalogowaniu do zebry (z wykorzystaniem usługi telnet) należy wejść do konfiguracji interfejsu eth0 (polecenie interface z parametrem eth0). Tu sieci nadawany jest prefiks i maska (dla rutera R1 i sieci s1 będzie to 3ffe:1:1:1::/64 - rys. 6a) oraz następuje załączenie protokołu sygnalizacyjnego ND (ang. *Neighbour Discovery*). Na rysunku 6b widoczne jest załączenie nadawania ND i ustawienie czasu rozgłaszania na 10 sekund. Jeśli zachodzi taka potrzeba, z tego poziomu można nadać też własny adres interfejsu (rys. 6c).

W dalszej kolejności, studenci zapoznają się z dostępnymi poleceniami pakietu Zebra, w tym z poleceniem show oraz wyświetlają tablicę rutingu (polecenie show ipv6 route). Studenci kończą pracę z usługą telnet wydając polecenie exit.

```
R2_ripd(config)# router ripng
R2_ripd(config-router)# network
R2_ripd(config-router)# network eth0
R2_ripd(config-router)# network eth1
```

Rys. 7. *Konfigurowanie RIPng (ruter R2)*

W drugim kroku studenci przechodzą do fazy konfigurowania demona protokołu RIPng. Jest ona bardzo zbliżona do omawianego wcześniej konfigurowania protokołu RIP. Po połączeniu się przez telnet z demonem RIPng i wejściu do konfiguracji demona protokołu RIP studenci dodają do tablicy rutingu sieci miejscowe (rys. 7) i kończą pracę z usługą telnet. Należy zwrócić uwagę na specyfikę sposobu dodawania sieci miejscowej do tablic rutingu RIP i RIPng.

Po zakończeniu konfigurowania każdego z ruterów studenci obserwują zachowanie rutera, wyświetlając co kilka-kilkanaście sekund zawartość tablic rutingu. Protokół RIP co około 30 sekund rozsyła zawartość swojej tablicy rutingu do ruterów bezpośrednio z nim sąsiadujących, które na tej podstawie aktualizują zawartość własnych tablic rutingu. Można zatem zaobserwować tablicę rutingu zawierającą jedynie wpisy o sieciach miejscowych rutera (rys. 8a), a po chwili tablicę uzupełnioną o wpis pochodzący z tablicy rutingu nowouruchomionego sąsiada (rys. 8b). Uruchomienia kolejnych ruterów skutkują kolejnymi aktualizacjami tablic rutingu (rys. 8c). Obserwacje takie można wykonać zarówno dla protokołu RIP (rys. 8a,b,c), jak i dla protokołu RIPng (rys. 8d,e). Obserwacja ta jest jednocześnie weryfikacją poprawności skonfigurowania maszyn wirtualnych laboratorium Netkit.

Jest specyfiką modułu Zebra, że szczegóły wyświetlanych wpisów do tablicy rutingu różnią się minimalnie, w zależności od kontekstu. Na rysunku 4b zobrazowano wyświetlenie tablicy w kontekście połączenia z terminalem Zebra. Rysunek 8, z kolei, pokazuje tablicę wyświetlaną w kontekście połączenia z demonem protokołu RIP (rys. 8a,b,c) lub RIPng (rys. 8d,e).

Rysunek 8a jest odpowiednikiem rysunku 4b dla kontekstu połączenia z RIP. Jak widać, lista dopuszczalnych oznaczeń kodowych źródeł informacji rutingowej jest teraz krótsza, brak w nich kodów związanych z jądrem Linuxa i protokołem ISIS.

Nie ma też znacznika wyboru ostatecznej trasy spośród tras alternatywnych (znak ">"). Protokół RIP ustala zawsze tylko jedną trasę, optymalną (minimalną) pod względem liczby przeskoków (liczby węzłów pośredniczących). Informację o trasie, przekazywane przez protokół do tablicy FIB, należy traktować jedynie jako propozycję. RIP proponuje taką trasę, a czy moduł Zebra z tej propozycji skorzysta zależy od preferencji, jakie nadamy dla tego źródła informacji rutingowej.

Ponieważ nie ma eksportu informacji z FIB do protokołów rutingu, znacznik pozyskania z FIB (znak "*") również nie występuje. Wśród tras widocznych na rysunku 8a występują tylko dwie z trzech tras widocznych na rysunku 4b. Są to trasy prowadzące do sieci miejscowych s1 i s2. Trzecia trasa, czyli trasa do sieci wirtualnej zbudowanej na bazie interfejsu zwrotnego loopback, nie jest obserwowalna na poziomie protokołu rutingu, a jednokierunkowy eksport z RIB do FIB sprawia, że informacja o tej trasie nie może zostać pozyskana.

Tab. 5. *Oznaczenia kodowe sposobu pozyskania informacji rutingowej - kody szczegółowe dla protokołów RIP i RIPng*

kod	sposób pozyskania informacji o trasie
(n)	w normalny sposób, tj. z innego rutera RIP (RIPng)
(s)	przez redystrybucję z rutingu statycznego
(d)	trasa domyślna
(r)	przez redystrybucję z innego protokołu rutingu
(i)	z konfiguracji interfejsu
(a/S)	trasa zagregowana lub trasa nierozgłaszana (tylko RIPng)

W tablicy wyświetlanej w kontekście połączenia z demonem protokołu RIP nie ma znaczników znacznikach używalności i FIB. Dochodzą natomiast kody szczegółowe (ang. *sub-codes*, dosł. podkody), rozszerzające znacznik źródła informacji rutingowej. Informują one, w jaki sposób protokół RIP pozyskał informację o trasie. Kodem szczegółowym jest oznaczenie literowe ujęte w nawiasy okrągłe (tab. 5). Litera n (z ang. *normal*, dosł. normalny) wskazuje, że informacja o trasie została pozyskana w normalny sposób, czyli przez wymianę informacji z protokołem RIP z innego rutera. Litera s (z ang. *static*, dosł. statyczny) oznacza, że informacja o trasie została pozyskana przez redystrybucję tras z rutingu statycznego. Litera d (z ang. *default*, dosł. domyślny) symbolizuje trasę domyślną, czyli trasę domyślnie ustawioną w ruterze. Trasy domyślne zazwyczaj ustawiane są w ruterach końcowych w danej sieci, tj. w takich ruterach, za którymi nie ma już innych ruterów. Litera r (z ang. *redistribute*, dosł. redystrybucja) świadczy o tym, że informacja o trasie została pozyskana na drodze redystrybucji z innego protokołu rutingu. Litera i (z ang. *interface*, dosł. interfejs) oznacza, że informacja o trasie została pozyskana z konfiguracji interfejsu.

(a)

```
R1_ripd(config)# exit
R1_ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

     Network         Next Hop         Metric From            Tag Time
C(i) 80.26.1.0/24    0.0.0.0               1 self              0
C(i) 80.26.2.0/24    0.0.0.0               1 self              0
R1_ripd#
```

(b)

```
R1_ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

     Network         Next Hop         Metric From            Tag Time
C(i) 80.26.1.0/24    0.0.0.0               1 self              0
C(i) 80.26.2.0/24    0.0.0.0               1 self              0
R(n) 80.26.3.0/24    80.26.2.2             2 80.26.2.2         0 02:57
R1_ripd#
```

(c)

```
R1_ripd# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

     Network         Next Hop         Metric From            Tag Time
C(i) 80.26.1.0/24    0.0.0.0               1 self              0
C(i) 80.26.2.0/24    0.0.0.0               1 self              0
R(n) 80.26.3.0/24    80.26.2.2             2 80.26.2.2         0 02:31
R(n) 80.26.4.0/24    80.26.2.3             2 80.26.2.3         0 02:52
R1_ripd#
```

(d)

```
R1_ripd# show ipv6 ripng
Codes: R - RIPng, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface, (a/S) - aggregated/Suppressed

   Network      Next Hop                    Via     Metric Tag Time
C(i) 3ffe:1:1:1::/64
                  ::                       self         1   0
C(i) 3ffe:1:1:2::/64
                  ::                       self         1   0
R1_ripd#
```

(e)

```
R1_ripd# show ipv6 ripng
Codes: R - RIPng, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface, (a/S) - aggregated/Suppressed

   Network      Next Hop                    Via     Metric Tag Time
C(i) 3ffe:1:1:1::/64
                  ::                       self         1   0
C(i) 3ffe:1:1:2::/64
                  ::                       self         1   0
R(n) 3ffe:1:1:3::/64
                  fe80::3840:eeff:fe31:9ecd eth1        2   0  02:49
R(n) 3ffe:1:1:4::/64
                  fe80::ec97:f2ff:feab:470c eth1        2   0  02:51
R1_ripd#
```

Rys. 8. *Obserwacja pracy protokołów RIP i RIPng: a) tablica rutingu rutera R1 dla IPv4 (przed uruchomieniem rutera R2 i R3), b) tablica rutingu rutera R1 dla IPv4 (po kilkudziesięciu sekundach od uruchomienia rutera R2), c) ostateczna tablica rutingu rutera R1 dla IPv4, d) tablica rutingu rutera R1 dla IPv6 (przed uruchomieniem rutera R2 i R3), e) ostateczna tablica rutingu rutera R1 dla IPv6*

Tablica rutingu RIB protokołu RIP przedstawiana jest za pomocą siedmiu kolumn. Zawierają one, w kolejności (rys. 8a, b, c):
- oznaczenie kodowe źródła informacji rutingowej oraz sposobu jej pozyskania,
- sieć docelową (ang. *network*), czyli adres własny sieci docelowej i jej maskę w formacie skróconym,
- adres IP następnego rutera na trasie (dosł. następnego przeskoku, ang. *next hop*),
- koszt (metrykę, ang. *metric*) danej trasy, zależny od liczby ruterów pośredniczących (przeskoków),
- informację, od kogo (ang. *from*) pozyskana została trasa,
- znacznik trasy (ang. *tag*),
- czas (ang. *time*), jaki upłynął od otrzymania ostatniej aktualizacji wpisu od innego rutera.

Na rysunku 8a jest widoczna tablica rutingu protokołu RIP zaobserwowana na ruterze R1 niedługo po zakończeniu czynności konfiguracyjnych. Tablica zawiera dwie trasy, do sieci s1 (adres własny: `80.26.1.0`) i do sieci s2 (adres własny: `80.26.2.0`), obie prowadzące do sieci miejscowych rutera R1 (kod "C") i pozyskane z konfiguracji interfejsu (kod "`(i)`"). Ponieważ zarówno s1, jak i s2 jest bezpośrednio osiągalna z rutera R1, adres IP następnego rutera na trasie został wyzerowany (`0.0.0.0`). Metryka takiej trasy jest równa 1 (sieci miejscowe są oddzielone od rutera R1 tylko własnym ruterem brzegowym, czyli samym ruterem R1) i jest to najmniejsza możliwa metryka dla protokołu RIP. Informacja, od kogo została pozyskana trasa wskazuje na wpis własny (ang. *self*). Ponieważ wpisy o trasach do sieci podłączonych bezpośrednio są ważne bezterminowo, dla obu tych wpisów ostatnia kolumna (czas mierzony od ostatniej aktualizacji) pozostaje niewypełniona.

Dołączenie do sieci prawidłowo skonfigurowanego rutera R2, a następnie rutera R3, skutkować będzie uzupełnieniem tablicy rutingu rutera R1 o wpisy pozyskane na drodze wymiany komunikatów między, kolejno, R1 a R2 (rys. 8b) oraz R1 a R3 (rys. 8c). Rutery R2 i R3 wprowadziły do tablicy rutingu rutera R1 informacje o swoich sieciach miejscowych:
- sieci s3 (adres własny: `80.26.3.0`), bezpośrednio dołączonej do rutera R2,
- sieci s4 (adres własny: `80.26.4.0`), bezpośrednio dołączonej do rutera R3.

Obie te trasy zostały pozyskane przez protokół RIP (kod źródła "R") na drodze normalnej wymiany informacji rutingowej (kod "`(n)`"), czyli poprzez wymianę tablic rutingu z sąsiadującymi ruterami. W kolumnie zawierającej informacje o następnym ruterze na trasie jest podany adres IP interfejsu eth0 rutera R2 (`80.26.2.2`, rys. 8b,c) oraz adres IP interfejsu eth0 rutera R3 (`80.26.2.3`, rys. 8c). Adresy te występują ponownie w piątej kolumnie, informującej od kogo (z jakiego adresu IP) została pozyskana trasa. Ponieważ wpisy o trasach do sieci s2 i s3 zostały pozyskane ze źródeł zewnętrznych, kolumna piąta zamiast słowa kluczowego *self* zawiera adres interfejsu rutera, który nadesłał tablicę rutingu będącą źródłem wpisu.

W przypadku tras prowadzących od rutera R1 do sieci s3 lub sieci s4, metryka trasy jest równa 2.

Ostatnia kolumna zawiera czas, jaki upłynął od chwili dokonania wpisu (lub jego ostatniej aktualizacji) do chwili wykonania polecenia `show ip rip`. Jak widać na rysunkach 8b i 8c, czas ten zliczany jest "do tyłu", od wartości 3:00 (trzy minuty zero sekund) do zera. Z chwilą wyzerowania timera, trasa zostanie uznana za nieaktualną. Ostatnia kolumna ma zatem charakter "czasu życia" trasy.

Warto zauważyć, że tablica rutingu protokołu RIP nie zawiera jawnego wskazania interfejsu, na który będą wysyłane rutowane datagramy IP, a jedynie adres IP interfejsu następnego rutera na trasie. Interfejs ten wskazywany jest w sposób niejawny, pośredni. Jest to interfejs, który znajduje się w tej samej sieci, co interfejs następnego rutera.

Rysunek 8d,e przedstawia tablicę rutingu rutera R1 pracującego z wersją 6 protokołu IP i protokołem RIPng. Na rysunku 8d widoczna jest tablica RIPng obserwowana przed uruchomieniem ruterów R2 i R3 (jest to odpowiednik rysunku 8a), natomiast na rysunku 8e zobrazowana została ostateczna wersja tablicy rutingu, zaobserwowana po uruchomieniu wszystkich ruterów sieci testowej (odpowiednik rysunku 8c). Jak widać na rysunkach, tablice rutingu RIPng i RIP nieco różnią się między sobą, poczynając od czysto formalnej zmiany znaczenia kodu źródła informacji rutingowej RIP (kod "R" oznacza teraz protokół RIPng zamiast RIP - tab. 4), poprzez wprowadzenie nowego kodu szczegółowego (kod "`(a/S)`" – tab. 5), na strukturze reprezentacji tablicy rutingu kończąc.

Dodatkowy kod szczegółowy `(a/S)` (z ang. *aggregated/Suppressed*, dosł. zagregowany/wytłumiony) informuje o dokonanych agregacjach tras lub o trasach nierozgłaszanych.

Tablica wyświetlana w wyniku wykonania polecenia `show ipv6 ripng` ma siedem kolumn, jednakże (w porównaniu do tablicy RIP) jedna kolumna została usunięta, a jedna dodana:
- usunięto kolumnę zawierającą informację od kogo została pozyskana dana trasa,
- dodano kolumnę informującą, przez (ang. *via*) który interfejs należy wysyłać rutowane datagramy IP.

W ten sposób, oprócz adresu następnego rutera na trasie datagramu, dodano jeszcze jawne wskazanie, przez który interfejs należy datagramy do tego następnego rutera wysłać. W IPv6 wiele adresów IP może być przypisanych do jednego interfejsu i wysłanie datagramów do jakiegoś sąsiedniego rutera może nie być tak oczywiste, jak w przypadku IPv4. Tym bardziej, że komunikaty RIPng są wprawdzie wysyłane na adres multikastowy `ff02::09`, ale adres węzła źródłowego jest adresem miejscowym z puli adresowej `fe80::`, czyli adresem lokalnym dla łącza (ang. *link-local*), a zatem nierutowalnym. Jest to zgodne z ogólną filozofią protokołów RIP/RIPng (komunikaty RIP/RIPng) są przekazywane tylko w sieci miejscowej i nie mogą być retransmitowane przez inne rutery.

Aby namacalnie pokazać studentom problem z adresami nierutowalnymi, można skorzystać z programu ping uruchamianego na jednym z ruterów. W przypadku adresów globalnych do program ping wystarczy podać adres docelowego systemu. W przypadku adresów nierutowalnych należy dodatkowo podać w opcjach programu ping interfejs, przez który wysyłamy datagramy IP.

Rysunek 8d jest odpowiednikiem rysunku 8a dla protokołu RIPng. Pomijając zmianę adresacji z 32-bitowej na 128-bitową i drobne różnice w zawartości kolumn, tablice rutingu RIP i RIPng widoczne na rysunkach, odpowiednio, 8a i 8d przechowują tę samą informację. Tablica zawiera dwie trasy, prowadzące do sieci s1 oraz s2, będących sieciami miejscowymi rutera R1 (kod "C"), a informacja o trasach została pozyskana z konfiguracji interfejsu (kod "`(i)`"). Jako, że obie sieci są bezpośrednio osiągalne z rutera R1, adres IP następnego rutera na trasie został wyzerowany (`::`). Informacja, że trasa została wprowadzona jako wpis własny (ang. *self*) rutera R1 znalazła się w kolumnie *Via* (a nie w, usuniętej w RIPng, kolumnie *From*). Metryka każdej trasy jest równa 1. Wpisy o trasach do sieci podłączonych bezpośrednio są bezterminowe, zatem kolumna czasu nie została wypełniona.

Podobnie rzecz się ma z parą rysunków 8c i 8e, na których przedstawiających tablice rutingu protokołów, odpowiednio, RIP i RIPng w chwili, gdy tablica rutingu rutera R1 została już uzupełniona o wpisy pochodzące z tablic ruterów R2 i R3. W przypadku wpisów pozyskanych od innych ruterów RIPng na uwagę zasługują

głównie adresy IP następnego rutera na trasie. Adresy te są pozyskiwane z pola adresu źródłowego datagramu IP przenoszącego tablicę rutingu i są to adresy tworzone na bazie adresu karty sieciowej, adresy lokalne dla łącza, nieprzenoszone poza sieć miejscową. Ponieważ dla wszystkich wpisów prefiks sieci wynosi `fe80::`, nie da się rozróżnić, na który interfejs rutera R1 (eth0 czy eth1) należy kierować datagram IP przesyłany do sieci nie będącej siecią podłączoną bezpośrednio do R1. Stąd konieczność jawnego wskazania interfejsu w kolumnie *via*.

PODSUMOWANIE

Netkit jest niewymagającym, interesującym pod względem możliwości emulacyjnych, rozwiązaniem przeznaczonym do wspomagania dydaktyki sieci komputerowych w wyższych szkołach technicznych. W niniejszym artykule, na przykładzie dwóch ćwiczeń laboratoryjnych z przedmiotu "Wybrane zagadnienia routingu", wykazano, że program Netkit bardzo dobrze nadaje się do realizacji ćwiczeń laboratoryjnych obejmujących konfigurowanie i testowanie protokołów rutingu wewnętrznego zawartych w pakiecie Zebra/Quagga, współpracujących zarówno z wersją 4, jak i wersją 6 protokołu IP.

Ćwiczenia laboratoryjne, których ramy zostały przedstawione w niniejszej publikacji, można łatwo rozszerzać o kolejne obserwacje, zarówno tablicy rutingu RIP (RIPng), jak i obserwację tablicy na poziomie modułu Zebra. Możliwe jest również przechwycenie przekazywanych wiadomości rutingowych za pomocą programu tcpdump, a następnie analiza ich zawartością (dzięki programowi Wireshark).

BIBLIOGRAFIA

1. Chodorek A., Chodorek R., *Możliwości zastosowania emulatora Netkit w badaniach naukowych i dydaktyce*. Logistyka 2014, nr 6.
2. Hedrick C.L., *Routing Information Protocol*. RFC 1058, June 1988.
3. Malkin G., *RIP Version 2*. RFC 2453, November 1998.
4. Malkin G., Minnear R., *RIPng for IPv6*. RFC 2080, January 1997.
5. Pizzonia M., Rimondini M., *Netkit: network emulation for education*. Software: Practice and Experience, 2014.
6. Rimondini M., *Emulation of computer networks with Netkit*. Department of Computer science and Automatization of the University of Rome, Italy 2007.

APPLICABILITY OF THE NETKIT EMULATOR TO TEACHING ROUTING BY THE EXAMPLE OF RIP AND RIPNG PROTOCOLS

Abstract

This article is devoted to the use of computer network emulator Netkit to teaching computer networks in higher technical schools. It is a software tool to configure and test complex networks without the need of usage of specialized networking equipment. The emulator has low hardware requirements. By the example of laboratory classes of the RIP protocol (in version for IPv4 and IPv6) was brought closer to the reader how to create a test topology and setting up emulated routers and hosts. The possibility of using the Netkit emulator is not limited only to teaching but also can be used to conduct scientific research in the field of internal routing protocols.

Autorzy:
dr inż. **Agnieszka Chodorek** – Politechnika Świętokrzyska, Wydział Elektrotechniki, Automatyki i Informatyki, Katedra Systemów Informatycznych; 25-314 Kielce; al. Tysiąclecia Państwa Polskiego 7. E-mail: a.chodorek@tu.kielce.pl
dr inż. **Robert Chodorek** – AGH Akademia Górniczo-Hutnicza, Wydział Informatyki, Elektroniki i Telekomunikacji, Katedra Telekomunikacji; 30-059 Kraków; Al. A. Mickiewicza 30. E-mail: chodorek@agh.edu.pl