Uniwersytet w Białymstoku

Instytut Informatyki

Albert Denkiewicz

Failover oraz load balancing oparty na IPFilter

Praca dyplomowa napisana
pod kierunkiem
dr. Mariusza Żynela

Białystok 2020

Spis treści

Wstęp **1**

1 IPFilter **2**
1.1 Historia 2
1.2 Architektura 3
1.3 Konfiguracja 7
1.3.1 `ipf.conf` 8
1.3.2 `ipnat.conf` 16
1.3.3 `ippool.conf` 17
1.3.4 Parametry tuningowe 18

2 Failover oraz High Availability **19**
2.1 Failover 19
2.2 Single Point Of Failure 19
2.3 Mean Time Between Failures 20
2.4 Mean Time To Repair 20
2.5 Service Level Agreement 21
2.6 High Availability 22
2.7 Fault Tolerance 23
2.8 Disaster Recovery 23
2.9 Porównanie 24
2.10 Zastosowanie Failover 25

3 Load Balancing **30**
3.1 Strategia dystrybucji obciążenia 30
3.2 Wybór serwera przez Load Balancer 32
3.3 Load balancing w IPFilter 33

4 IP Multipathing **35**
4.1 Budowa IPMP 35
4.2 Przebieg awarii 39
4.3 Przykładowa konfiguracja 40

Podsumowanie **42**

A Gramatyka reguł filtra pakietów **43**

B Parametry tuningowe **45**

C Plik konfiguracyjny `ipf-wan1.conf` **47**

D Plik konfiguracyjny `ipf-wan2.conf` **50**

E Plik konfiguracyjny `ipnat-wan1.conf` **53**

F Plik konfiguracyjny `ipnat-wan2.conf` **54**

Spis rysunków **55**

Bibliografia **56**

Wstęp

Zatrudniając się po raz pierwszy w wielkiej korporacji, człowieka wszystko tam zadziwia. Wielkość przedsięwzięcia, ilość pracowników, czy też zorganizowany czas pracy co do sekundy. Wszyscy pracownicy dają z siebie wszystko pracując na największych obrotach tylko po to, żeby zadowolić szefostwo. Robią to też ich komputery. Muszą przecież cały czas wymieniać się postępami zadań czy też pozyskiwać nowe dane z głównego serwera. Wszystko działa, jest świetnie, lecz zdarza się, że nagle serwer ulega uszkodzeniu, a praca zostaje przerwana. Należy zapobiegać takim sytuacjom stosując wszelakie sposoby, aby zapewnić nieprzerwaną pracę użytkownikom.

Celem tej pracy jest opracowanie oprogramowania, które automatycznie przełączy interfejsy sieciowe w przypadku wykrycia usterki, dając możliwość nieprzerwanej pracy. W realizacji zadania został wykorzystany VirtualBox, na których zainstalowano maszyny testowe oraz odpowiednie podsieci wirtualne. Schemat połączeń sieciowych przedstawiono na rysunku 2.2.

Wraz z Mateuszem Maciejczukiem, który pisze pracę licencjacką [2] na powiązany temat, został opracowany rozdział pierwszy, poświęcony oprogramowaniu IPFilter. W drugim rozdziale dowiemy się o metodach Failover oraz High Availability. Tutaj znajduje się opis opracowanego w ramach pracy programu do przełączania sieci WAN. Trzeci rozdział zawiera informacje na temat Load Balancing, także z wykorzystaniem IPFilter. W ostatnim, czwartym rozdziale, przeczytamy o tym, jak przebiega automatyczne usuwanie skutków awarii łącza sieciowego przy wykorzystaniu IP Multipathing.

W pracy zastosowanych jest dużo angielskich zwrotów bez tłumaczenia ze względu na to, że w środowisku IT rzadko stosuje się ich polskie odpowiedniki.

Rozdział 1

IPFilter

Ten rozdział opracowano wspólnie z Mateuszem Maciejczukiem, który pisze pracę [2] poświęconą również oprogramowaniu IPFilter.

1.1 Historia

IPFilter to oprogramowanie umożliwiające filtrowanie pakietów (ang. packet filtering), a dzięki temu, na konstruowanie zapór sieciowych (ang. firewall) nie tylko na platformie Solaris, ale także na FreeBSD, NetBSD, OpenBSD, AIX, HP-UX, IRIX, Linux oraz wielu innych systemach Unixowych. Wspiera protokoły IPv4 oraz IPv6. Jest filtrem pakietów z pamięcią stanów.

Autorem jest Darren Reed. Projekt od początku funkcjonował jako open source i był dość ściśle związany z platformą Solaris. Niestety, został zaniechany przed kilku laty. Główną przyczyną było wykupienie Sun Microsystems przez Oracle i zamknięcie projektu OpenSolaris.

Na platformie Solaris, do wersji 9, nie było zintegrowanego oprogramowania do filtrowania pakietów. Jedyną alternatywą był IPFilter, ale należało go samodzielnie skompilować i zainstalować. W wersji 9 pojawił się firewall pod nazwą SunScreen, zaimplementowany przez inżynierów z Sun Microsystems. Architektura SunScreen przypominała współczesne IPTables z Linuxa, choć nie tak rozbudowane. Był to moduł jądra systemu z interaktywnym interfejsem do wprowadzania reguł. Przy bardziej rozbudowanych konfiguracjach żonglowanie zaklęciami, aby dodać lub zmodyfikować regułę dawało w kość. Trudna była także diagnostyka. Wielu administratorów nie instalowało w ogóle SunScreen, tylko kompilowało i instalowało IPFilter.

W kolejnej, 10 wersji Solaris pojawił się IPFilter. Były to ciekawe czasy (2003-2005), gdy grupa użytkowników Solaris odwiodła Sun Microsystems od rezygnacji z platformy x86, a firma zaczęła dużo agresywniej inwestować w swój system operacyjny. W tej wersji pojawiło się całe mnóstwo nowatorskich rozwiązań: ZFS, D-Trace, Branded Zones, SMF, FM, Trusted Extensions, JDS by wymienić tylko najciekawsze, które dopiero teraz trafiają na

inne systemy, dzięki udostępnieniu kodu źródłowego jako OpenSolaris w 2008 roku. Wtedy Darren Reed podpisał kontrakt z Sun Microsystems. Pojawiły się konkretne motywacje finansowe, IPFilter oficjalnie stał się integralną częścią systemu Solaris i zyskał trwałe wsparcie ze strony świetnych inżynierów z Sun Microsystems.

IPFilter powstał na Solaris, a w zasadzie na SunOS, bo wówczas tak nazywał się system tworzony przez Sun Microsystems. Pierwsza wersja 1.0 została wypuszczona 22 kwietnia 1993 roku.

W Solaris 10 znalazła się wersja 4.1.9 w 2004. To jest ostatnia wersja IPFilter, jaka została zintegrowana z systemem Solaris. Przez wiele lat istnienia i wpierania Solaris 10 była ona wielokrotnie poprawiana i dostosowywana do nowszych wersji jądra systemu. IPFilter ewoluował dwutorowo: firma Sun Microsystem podtrzymywała wersję 4.1.9, natomiast niezależnie Darren Reed wypuszczał nowsze, udoskonalone wersje, często zawierające poprawki firmy Sun. Spadkobiercy Solaris 10, czyli Oracle Solaris 11, Illumos, OpenIndiana, SmartOS i pozostałe dalej używają wersji 4.1.9.

Wersja 5.1, która ujrzała światło dzienne 9 maja 2010 roku, była całkowitym przepisaniem kodu z licznymi zmianami, aczkolwiek wsteczna kompatybilność konfiguracji została zachowana.

Ostatnia wersja IPFilter, jaką można znaleźć, jest 5.1.2. Wyszła 22 czerwca 2012 roku i została włączona do FreeBSD. Jedynie w źródłach tego systemu można tę wersję znaleźć.

W tej chwili pierwotna strona IPFilter i związana z nim lista dyskusyjna nie działają. Utworzona przez autora IPFilter strona na `sourceforge.net` zawiera źródła wersji 5.0.5 oraz 4.1.32. Jej ostatnia aktualizacja była wykonana w 2013 roku. Na znajdującym się tam forum czasem ktoś coś zamieści, ale są to pojedyncze posty pozostające bez odzewu. Darren Reed nie odpisuje na wysłane wiadomości e-mail. Aktywni zostali chyba wyłącznie twórcy FreeBSD konserwujący wersję 5.1.2. Tak obecnie wygląda sytuacja z IPFilter.

Narzuca się w tym momencie pytanie, po co inwestować w IPFilter i wiązać z tym oprogramowaniem swoje rozwiązania? Odpowiedź jest prosta: tak długo, jak chcemy używać Solaris, jesteśmy skazani na IPFilter, bo to jedyny sensowny firewall dla tej platformy.

1.2 Architektura

IPFilter działa zgodnie z ogólną teorią filtra pakietów warstwy 3 i 4 modelu OSI i nie różni się zbytnio od innych implementacji jak IPTables, IPFW, czy PF. Różnice polegają na podejściu do sposobu konfiguracji i oczywiście składni reguł oraz na wewnętrznej realizacji filtra.

Z poziomu userland możemy manipulować stosem TCP/IP, ale w dość ograniczony sposób. Programy narzędziowe takie jak `ifconfig`, `route`, czy `ndd` mogą w istotny sposób wpływać na przepływ pakietów przez system,

ale filtrowanie pakietów i translacja adresów wymagają dostępu do pakietów TCP/IP w ich surowej postaci, dostępnej jedynie na poziomie zarezerwowanym dla jądra systemu. Dlatego najważniejszą częścią IPFilter jest moduł jądra, zwany `ipf`, który odrabia całą czarną robotę. IPFilter można skompilować razem z jądrem, jako jego integralną część, ale robi się to w bardzo wyjątkowych sytuacjach, na ogół, na platformach z jądrem monolitycznym.

Firewall powinien mieć nieograniczony dostęp do pakietów TCP/IP i przetwarzać je możliwie sprawnie, nie wpływając na ogólną przepustowość sieci w istotny sposób. To jest drugi powód, aby umieścić go w jądrze systemu.

Poniżej zamieszczony jest fragment listy załadowanych modułów jądra w Solaris 10, uzyskanej poleceniem `modinfo`. Jak widać, pod ID 161 znajduje się moduł `ipf`.

```
 Id         Loadaddr   Size Info Rev Module Name
154 ffffffffefd2a000  33790   65   1  e1000g (Intel PRO/1000 Ethernet 5.2.27)
155 ffffffffef26f898    dc0    -   1  mac_ether (Ethernet MAC plugin 1.1)
157 ffffffffefa9a000   1d48    6   1  openeepr (OPENPROM/NVRAM Driver v1.20)
158 ffffffffefd5e000  a0930    8   1  zfs (ZFS filesystem version 15)
158 ffffffffefd5e000  a0930  181   1  zfs (ZFS storage pool)
159 ffffffffefcd4300    f80   24   1  pts (Slave Stream Pseudo Terminal dr)
160 ffffffffefcf2000  1d2e0  202   1  mpt_sas (MPTSAS HBA Driver 00.00.00.16)
161 ffffffffefdf6000  9b4d0  165   1  ipf (IP Filter: v5.1.2)
162 ffffffffef228748    b28   21   1  log (streams log driver)
163 ffffffffefa9c000   1b48  154   1  cryptoadm (Cryptographic Administrative In)
165 ffffffffefa9e000   16d0  185   1  power (power button driver v1.17)
166 ffffffffefc2c220    f30   90   1  kstat (kernel statistics driver 1.26)
167 ffffffffefd13000   5ab0   88   1  devinfo (DEVINFO Driver 1.71)
169 ffffffffefcab6b8    b28  104   1  objmgr (Object Manager 1.27)
170 ffffffffef9f3440    f38  113   1  xsvc (xserver svc)
```

W IPFilter można wyróżnić 3 następujące podsystemy:

1. podsystem filtra pakietów,
2. podsystem translacji adresów,
3. podsystem puli adresów.

Filtr pakietów analizuje pakiety TCP/IP, porównując je do reguł umieszczonych na wewnętrznej liście. Jeśli pakiet pasuje do danej reguły, wykonywana jest jedna z dwóch możliwych akcji: pakiet jest przepuszczany lub blokowany. Dodatkowo może być przekazana informacja do monitora, przepuszczony pakiet może zostać zduplikowany albo przekierowany na inny interfejs niż ten, wyznaczony na podstawie tablicy routingu. Ostatnia akcja wykorzystywana jest do realizacji policy routing.

Jak sugeruje nazwa, zadaniem podsystemu translacji adresów jest zmiana adresu IP i portu TCP w pakiecie zgodnie z regułami translacji umieszczonymi na wewnętrznej liście. Są tutaj dwie klasy reguł: jedna odpowiada SNAT, druga DNAT.

Podsystem puli adresów pozwala na gromadzenie wielu adresów sieciowych pod wybraną nazwą. Jeśli na przykład mamy 5 adresów IP, z który blokujemy wszelki ruch, możemy je umieścić w jednej puli i zamiast pisać 5, niemal identycznych reguł, różniących się jedynie adresem źródłowym pakietu, wystarczy napisać jedną regułę, używając nazwy puli w miejscu adresu.

Poza modułem `ipf` w skład oprogramowania IPFilter w wersji 5.1.2 (podobnie dla wersji wcześniejszych od 4.0) wchodzą następujące programy pozwalające kontrolować jego zachowanie:

`ipf` Interfejs użytkownika do podsystemu filtra pakietów. Modyfikuje i zarządza wewnętrzną listą reguł w jądrze systemu. Czyta plik konfiguracyjny, parsuje znajdujące się tam reguły i umieszcza je na wewnętrznej liście reguł. Przy jego pomocy można także usuwać reguły, włączyć i wyłączyć filtrowanie, podejrzeć lub zmodyfikować wewnętrzne zmienne sterujące filtrowaniem pakietów oraz translacją adresów.

`ipfs` Zapisuje i odtwarza tablicę stanów filtra oraz NAT.

`ipfstat` Raportuje statystyki filtra pakietów.

`ipftest` Testuje reguły filtra bez konieczności ładowania ich do wewnętrznej listy w jądrze systemu. Pracuje na plikach utworzonych przez różne sniffery sieciowe jak `libpcap`, `snoop`, `tcpdump` i dla każdej reguły zwraca: `pass`, `block` lub `nomatch`.

`ipmon` Monitoruje pakiety przeznaczone do umieszczenia w logach. Przetwarza pakiety z surowej postaci do postaci czytelnej dla człowieka i wynik umieszcza w pliku lub w logu systemowym `syslog`.

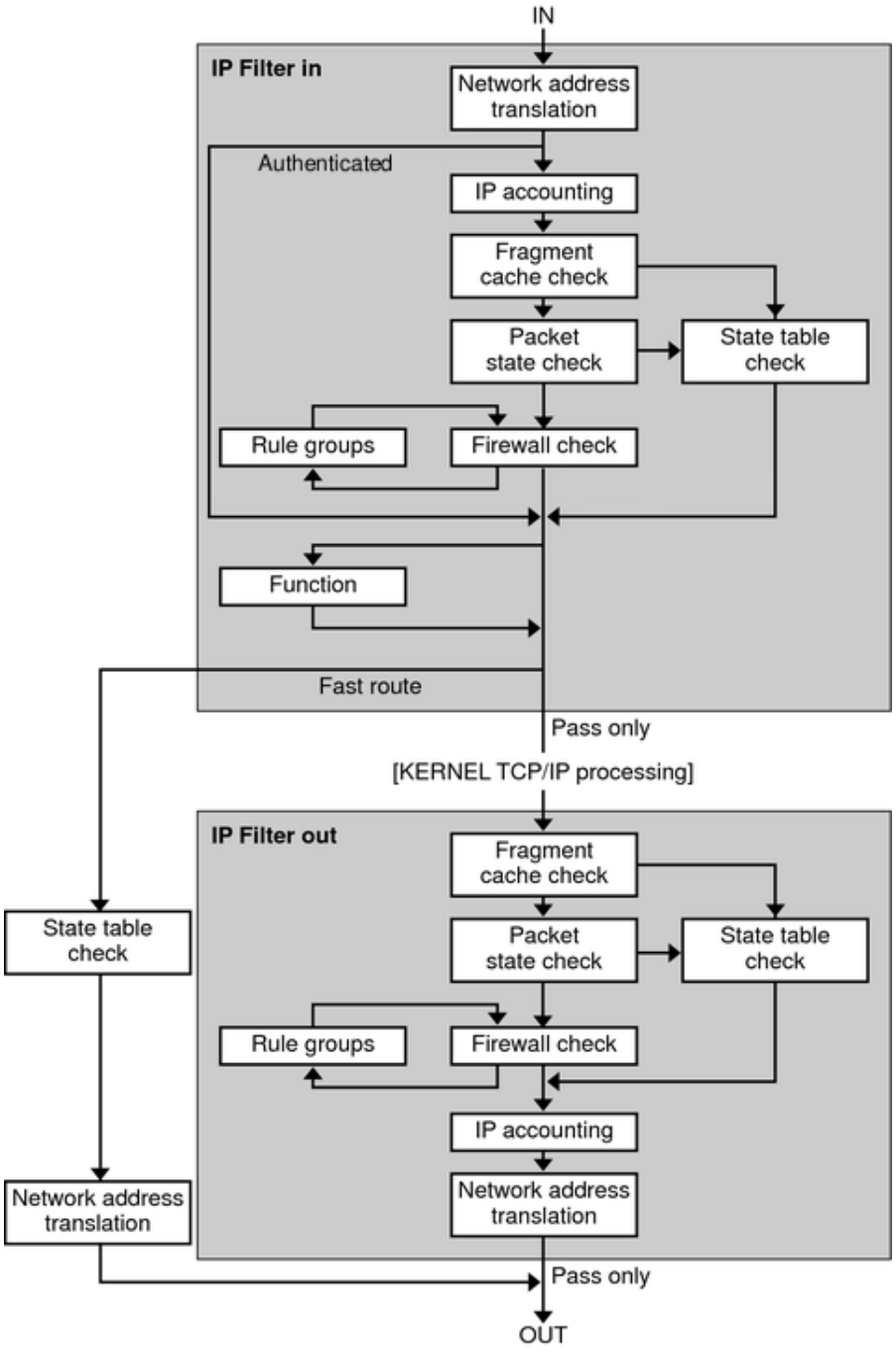
`ipnat` Interfejs użytkownika do podsystemu translacji adresów. Modyfikuje i zarządza wewnętrzną listą reguł translacji w jądrze systemu. Czyta plik konfiguracyjny, parsuje znajdujące się tam reguły translacji i umieszcza je na wewnętrznej liście reguł translacji. Przy jego pomocy można także usuwać reguły, obejrzeć aktualną listę reguł oraz statystyki NAT.

`ippool` Interfejs użytkownika do podsystemu pul adresów. Czyta plik konfiguracyjny i ładuje pule adresów do jądra systemu. Umożliwia także inspekcję pul w jądrze, dodawanie i usuwanie pojedynczych adresów.

`ipresend` Wysyła pakiety TCP/IP przechwycone wcześniej snifferem sieciowym. Pozwala testować i diagnozować sieć TCP/IP.

`ipsend` Generuje pakiet TCP/IP i wysyła go poprzez wskazany interfejs sieciowy. Pozwala testować i diagnozować sieć TCP/IP.

`iptest` Wysyła serię pakietów TCP/IP w celu testowania stosu TCP/IP, filtra pakietów lub NAT.



Rysunek 1.1: Diagram przepływu pakietów przez jądro systemu i IPFilter.

W przepływie pakietu TCP/IP przez system z IPFilter (rys. 1.1) wyróżniamy następujące, istotne etapy przetwarzania:

Network Address Translation Zamiana prywatnego, źródłowego adresu IP na adres publiczny albo mapowanie wielu prywatnych, źródłowych adresów na jeden adres publiczny w przypadku SNAT. W przypadku DNAT zamiana publicznego adresu docelowego na adres prywatny w sieci lokalnej.

IP accounting Gromadzenie danych statystycznych na potrzeby raportowania użycia filtra pakietów i translatora adresów.

Fragment cache check Jeśli pakiet w kolejce jest fragmentem, a poprzedni fragment został przepuszczony, to ten fragment jest również przepuszczany z pominięciem tablicy stanów i sprawdzania reguł.

Packet state check Jeśli reguła zawiera `keep state`, to wszystkie pakiety w określonej sesji są automatycznie przepuszczane albo blokowane, w zależności od tego, czy ta reguła przepuszcza, czy blokuje pakiety.

Firewall check Sprawdzenie czy dany pakiet ma zostać przepuszczony do dalszego przetwarzania w jądrze systemu lub do wyjścia przez interfejs do sieci.

Groups Grupy pozwalają pisać reguły w sposób przypominający drzewo.

Function Akcja jaka ma być podjęta. Możliwe akcje to przepuszczenie, zablokowanie lub wysłanie komunikatu ICMP.

Fast-route Specjalne reguły pozwalają przekazać pakiet bezpośrednio na podany interfejs wyjściowy z pominięciem stosu TCP/IP w jądrze systemu, który wyznacza trasę pakietu na podstawie tablicy routingu i powoduje pomniejszenie TTL pakietu.

Authenticated Pakiety, które zostały poddane autentykacji, przechodzą przez filtr tylko raz, aby uniknąć wielokrotnego przetwarzania.

1.3 Konfiguracja

Wdrożenie filtra pakietów wymaga precyzyjnego sformułowania polityki, określenia kto ma dostęp do jakich zasobów i usług. Na podstawie takiej polityki opracowuje się plan reguł filtra i translacji adresów. Ostatni etap, to zapisanie planu reguł zgodnie ze składnią wdrażanego systemu, czy oprogramowania.

Domyślnie, do konfiguracji IPFiltra używane są trzy następujące pliki:

- `/etc/ipf/ipf.conf`,

- `/etc/ipf/ipnat.conf`,
- `/etc/ipf/ippool.conf`.

Każdy z nich, odpowiada jednemu z trzech podsystemów: filtra, translatora adresów i puli adresów.

1.3.1 `ipf.conf`

W pliku `/etc/ipf/ipf.conf` znajdują się reguły opisujące zachowanie filtra pakietów. Plik ten jest czytany przez program `ipf`, który ładuje je do pamięci wewnętrznej modułu jądra. Jest to domyślna, standardowa lokalizacja tego pliku, ale można ją zmienić i podać pełną ścieżkę używając opcji `-f`. Źródłem reguł może być także standardowe wejście, gdy jako ścieżkę do pliku podamy znak `-`. Może to być użyteczne na przykład wtedy, gdy potrzebujemy usunąć wszystkie reguły wejściowe:

```
ipfstat -i | ipf -rf -
```

Program `ipf` dopisuje reguły do listy wewnętrznej kolejno, tak jak zostały umieszczone w pliku konfiguracyjnym. Dopisanie sekwencji `@<num>` przed regułą powoduje wstawienie tej reguły na pozycji `<num>` aktualnej listy wewnętrznej. Jest to szczególnie wygodne podczas testowania aktywnego zestawu reguł.

Reguły sprawdzane są w takiej kolejności, w jakiej znajdują się na liście wewnętrznej. Jeden pakiet może pasować do wielu reguł. Ostatnia z nich jest decydująca, chyba że zastosowano opcję `quick`.

Z punktu widzenia zapory sieciowej istotne są dwa rodzaje reguł: te, które blokują i odrzucają pakiety (reguły typu `block`) oraz te, które przepuszczają pakiety (reguły typu `pass`). Obok decyzji co zrobić z danym pakietem znajdują się deklaracje określające warunki podjęcia tej decyzji i sposobu jej egzekucji. Najprostsze, poprawne reguły, jakie można sformułować, mogłyby wyglądać następująco:

```
block in all
pass out all
```

ale w tej postaci są zupełnie bezużyteczne, chyba że chcemy całkowicie odciąć się od sieci.

Każda reguła filtra składa się z co najmniej trzech następujących komponentów:

1. słowo kluczowe decyzji (`pass`, `block`, itp.),
2. kierunek pakietu (`in` albo `out`),
3. wzorzec adresu lub słowo kluczowe `all`.

Długie linie w pliku konfiguracyjnym można łamać excplicite używając znaku \, albo implicite pisząc jedną regułę w kilku wierszach. Komentarze zaczynają się od znaku `#` i kończą znakiem końca linii.

Dalej opisywane są zasady jak konstruować poprawne reguły. Jakie są dobre i złe praktyki, co należy uznać za bezpieczne, a co nie, wykracza jednak poza zakres tego opisu. Ograniczyliśmy się także do opisu tylko tych konstrukcji i deklaracji, które wykorzystujemy dalej w przykładach i testach, pomijając wiele pozostałych. Gramatyka pliku `ipf.conf` znajduje się w dodatku A. Tekst opracowano na podstawie [3].

Słowa kluczowe filtra

Pierwsze słowo kluczowe każdej reguły określa, co zostanie zrobione, gdy pakiet zostanie dopasowany do tej reguły. Możliwe są następujące słowa kluczowe:

`pass` Pakiet zostaje oznaczony jako przepuszczony w kierunku jakim płynie.

`block` Pakiet zostaje oznaczony jako zablokowany. Zablokowane pakiety płynące w kierunku `in` nie pojawią się w stosie TCP/IP jądra systemu, natomiast pakiety płynące w kierunku `out` nigdy nie wyjdą poza interfejs i nie pojawią się w na kablu.

`log` W wyniku tej reguły tworzony jest rekord informacyjny dla programu `ipmon`, który zapisuje go w określonym dzienniku systemowym. Reguły tego typu nie wpływają na to, czy ostatecznie pakiet ma być przepuszczony, czy zablokowany.

`count` Reguły tego typu pozwalają administratorowi zliczać pakiety i ilość przesyłanych bajtów. Dla ruchu wchodzącego (inbound) reguły takie są aplikowane po translacji adresów i filtrze, natomiast dla ruchu wychodzącego (outbound) są aplikowane przed translacją adresów, zanim pakiet zostanie odrzucony. Dlatego reguły te nie powinny być traktowane jako rzetelny wskaźnik.

`auth` Pakiety dopasowane taką regułą są kolejkowane do przetworzenia w programie z warstwy użytkowej systemu. Program ten zwraca werdykt co dalej zrobić z pakiem, przepuścić, czy zablokować. Jeśli kolejka się zapełni, pakiety zostaną odrzucone.

`call` Reguły takie dają dostęp do dodatkowych funkcji wbudowanych w IPFilter, które pozwalają podejmować bardziej skomplikowane akcje, aby podjąć ostateczną decyzję.

Dopasowanie interfejsu sieciowego

W systemach wyposażonych w kilka interfejsów sieciowych może być konieczne określenie, którego z tych interfejsów dotyczy dana reguła. Przyjęta polityka filtra pakietów może wymagać rozróżniania pakietów według interfejsu, na jakim się znajduje.

W niektórych maszynach obecność interfejsu sieciowego może zmieniać się dynamicznie. Na przykład programowe interfejsy sieci komutowanych opartych o Point-to-Point Protocol pojawiają się w momencie nawiązania połączenia na porcie szeregowym. Aby umożliwić działanie filtra na takich systemach, interfejs występujący w regułach może nie być obecny w systemie. Trzeba uważać, bo może to prowadzić do cichych błędów, gdy nazwa interfejsu została przekręcona podczas wpisywania.

Poniższe reguły dotyczą interfejsów odpowiednio `e1000g0` oraz `e1000g1`:

```
block in on e1000g0 all
pass out on e1000g1 all
```

Dopasowanie adresu

Najprostszy i najbardziej podstawowy sposób dopasowania w regułach filtra, to określenie adresu IP oraz portu TCP albo UDP. W nagłówku warstwy drugiej modelu TCP/IP mamy dwa adresy IP: źródłowy (source) oraz docelowy (destination). Konstruując regułę, adres źródłowy podajemy, poprzedzając go słówkiem `from`, a docelowy słówkiem `to`.

Adresy IP podajemy zgodnie z notacją CIDR, gdzie znak `/` oddziela adres IP od liczby określającej długość maski w bitach. W ten sposób możemy określiść całą podsieć. Pojedynczy host określamy, wpisując `/32` lub nie wpisując długości maski wcale. Słówko `any` oznacza dowolny adres IP. Na przykład:

```
block in on e1000g0 from 192.168.0.0/24 to any
pass out on e1000g1 from any to 172.16.0.10
```

Nie jest możliwe podanie zakresu adresów, który nie da się wyrazić za pomocą notacji CIDR, czyli nie jest podsiecią.

Ogólnie rzecz biorąc, zamiast adresów IP można używać adresów domenowych DNS. Należy jednak uważać, bo jeśli jednej nazwie odpowiada więcej niż jeden adres IP, to brany jest tylko pierwszy z nich. Natomiast w przypadku, gdy pobranie adresu z DNS trwa długo, albo jest blokowane przez inną część reguł, załadowanie i sprawdzenie reguły z adresem DNS może być opóźnione, jeśli w ogóle nie zakończy się błędem.

W sytuacji, gdy aplikujemy tę samą regułę do wielu adresów IP, zamiast tworzyć osobne reguły, można skonstruować jedną regułę, która zamiast konkretnego adresu IP używa tak zwanej puli adresów. Na przykład:

```
pass in on e1000g0 from pool/trusted to 172.16.0.10
```

Pule adresów tworzone są programem `ippool`.

W systemach o wielu interfejsach, albo gdy adresy interfejsów są dynamiczne, (na przykład uzyskiwane są przez DHCP) wygodne jest używanie nazw interfejsów zamiast ich adresów. W regułach skojarzonych z konkretnym interfejsem, poprzez użycie słówka `on` albo `via`, zamiast adresu IP możemy użyć nazwy interfejsu. Na przykład, gdy podstawowy adres interfejsu `e1000g0` to `192.168.0.1`, a maska to `255.255.255.0`, wówczas następujące reguły są równoważne:

```
pass in on e1000g0 from any to 192.168.0.1/32
pass in on e1000g0 from any to e1000g0/0
```

Warto tu zwrócić uwagę na odstępstwo od zasady, że sufiks `/32` oznacza konkretnego hosta w notacji CIDR. Podanie takiego sufiksu i całkowite opuszczenie sufiksu po nazwie interfejsu nie daje spodziewanych rezultatów.

Dodatkowo słówko `netmasked` pomaga zastosować dynamicznie przydzielaną maskę, tak aby zaadresować całą podsieć:

```
pass in on e1000g0 from any to 192.168.1.0/24
pass in on e1000g0 from any to e1000g0/netmasked
```

W regułach, które nie są skojarzone z żadnym interfejsem, nazwę interfejsu podajemy w nawiasach okrągłych. Poniższe reguły są równoważne:

```
pass in from any to 192.168.0.1/32
pass in from any to (e1000g0)/0
pass in from any to (e1000g0)
```

Dopasowanie protokołu

Aby móc dopasować pakiet według portu TCP/UDP musimy najpierw określić protokół pakietu. Robi się to za pomocą słówka `proto`, po którym podajemy nazwę protokołu albo jego numer (z pliku `/etc/protocols`). Na przykład:

```
block in on e1000g0 proto tcp from 192.168.0.0/24 to any
pass out on e1000g1 proto udp from any to 172.16.0.10
pass in on bge0 proto icmp from any to 192.168.0.0/16
```

Dopasowanie portu TCP/UDP

Gdy określimy w regule protokół, możemy wskazać numer portu, jaki ma być dopasowany. Ponieważ numery portów używane są inaczej niż adresy IP, dopasowując port, możemy użyć jednej z następujących relacji logicznych w ich całkiem naturalnym sensie: `< x`, `<= x`, `> x`, `>= x`, `= x`, `!= x`, gdzie `x` to numer portu. Dodatkowo można stosować zakresy portów:

`x <> y` numer portu jest mniejszy niż `x` i większy niż `y`,

`x >< y` numer portu jest większy niż `x` i mniejszy niż `y` oraz

`x:y` numer portu jest większy lub równy `x` i mniejszy lub równy `y`.

Na przykład:

```
block in on e1000g0 proto tcp from any port >= 1024 to any port < 1024
pass in on e1000g1 proto tcp from any to 172.16.10 port = 22
block out proto udp from any to 10.1.1.1 port = 135
pass in proto udp from 1.1.1.1 port = 53 to 10.1.1.1 port = 53
pass in proto tcp from 127.0.0.0/8 to any port = 6000:6009
```

Rozszerzone dopasowanie

W nagłówkach pakietów TCP znajdują się flagi, które wskazują na to, czy dany pakiet to nawiązanie, kontynuacja lub zakończenie połączenia itp. W połączeniu z dopasowaniem według portów, dopasowanie według flag TCP daje możliwość konstruowania bardzo precyzyjnych reguł.

Najbardziej podstawowe flagi TCP to:

`S SYN` Ustawiana jest jako jedyna w pierwszym pakiecie inicjującym połączenie ze strony klienta. W odpowiedzi serwer odsyła pakiet, z flagami `SYN` i `ACK` (tak zwany three-way handshake).

`A ACK` Ustawiana jest w pakiecie potwierdzającym otrzymanie danych.

`F FIN` Ustawiana, gdy jeden z uczestników połączenia kończy je.

`R RST` Ustawiana wyłącznie w pakiecie odpowiedzi na pakiet skierowany na port, który nie jest używany.

Gdy chcemy przepuścić wyłącznie pakiety inicjujące połączenie, możemy sformułować to, używając słówka `flags` w następujący sposób:

```
pass in on e1000g0 proto tcp from any to e1000g0/0 port = 80 flags S
```

Tak naprawdę, dopiero gdy zastosujemy pamięć stanów, filtrowanie na podstawie flag TCP pokaże swoją moc.

Filtrowanie na podstawie innych parametrów nie tylko nagłówka IP, możliwe jest nie tylko w protokołach TCP i UDP, ale także w ICMP. Słówko `icmp-type` pozwala precyzyjnie określać jakie komunikaty ICMP blokujemy lub przepuszczamy na podstawie typu ICMP. Na przykład żądanie echa to typ 8 (`echo`), natomiast odpowiedź to typ 0 (`echorep`). Poniżej wpuszczamy żądania echa i wypuszczamy odpowiedzi:

```
pass in on e1000g0 proto icmp from any to e1000g0/0 icmp-type echo
pass out on e1000g1 proto icmp from e1000g1/0 to any icmp-type echorep
```

Filtrowanie z pamięcią stanu

Filtrowanie z pamięcią stanu polega na tym, że IPFilter zapamiętuje pewne informacje z jednego bądź kilku pakietów, które widział i jest w stanie je później zaaplikować do pakietów, jakie pojawią się w przyszłości.

W różnych warstwach transportowych TCP/IP wygląda to inaczej. W TCP pierwszy pakiet inicjujący połączenie (z ustawioną flagą SYN) posiada dość informacji, aby na ich podstawie przepuścić kolejne pakiety z tego połączenia. IPFilter korzysta z portu TCP, flag, rozmiaru okienka oraz numerów sekwencyjnych, aby dopasować pakiety. W UDP wyłącznie numer portu może być wykorzystany. W ICMP typy komunikatów razem z identyfikatorem mogą być użyte. Dla pozostałych protokołów pozostaje jedynie adres IP i numer protokołu.

Pamięć stanu w regule włączamy za pomocą `keep state`. Pozwala to między innymi zredukować ilość reguł. Na przykład 4 reguły:

```
pass in on e1000g0 proto tcp from any to any port = 22
pass out on e1000g1 proto tcp from any to any port = 22
pass in on e1000g1 proto tcp from any port = 22 to any
pass out on e1000g0 proto tcp from any port = 22 to any
```

mogą być zastąpione jedną:

```
pass in on e1000g0 proto tcp from any to any port = 22 flags S keep state
```

Gdy używamy pamięci stanów dla protokołu TCP warto dodawać opcję `flags S`, aby tworzyć nowy stan wyłącznie podczas inicjacji połączenia, na samym jego początku, nie w środku połączenia, gdy mamy mniej informacji.

Policy routing

Zapory sieciowe z uwagi na swoje przeznaczenie, często lokowane są na styku kilku sieci o różnych parametrach. Często pożądane jest, aby przepływ pakietów był inny niż ten przewidziany tablicą routingu w jądrze systemu. Decyzje o zmianie trasy pakietu mogą być podejmowane na podstawie nie tylko adresu docelowego, co jest naturalne, ale także na podstawie adresu źródłowego albo portów. Dlatego też czasem policy routing zwane jest source based routing.

IPFilter wspiera policy routing i pozwala określić następny przeskok (next hop) w konstruowanych regułach. Przeskok deklaruje się podając nazwę interfejsu, który ma być użyty do wysłania pakietu oraz adres IP rutera bezpośrednio przyłączonego do danego interfejsu. Na przykład, gdy ruter przyłączony do interfejsu `e1000g1` ma adres `10.0.1.1` i chcemy, aby pakiety pochodzące z sieci `192.168.0.0/24`, pojawiające się na interfejsie `e1000g0`, wyszły do tego routera, to instruujemy IPFilter w następujący sposób:

```
pass out quick on e1000g0 to e1000g1:10.0.1.1 \
    from 192.168.0.0/24 to any keep state
```

Pakiet wychodzący z rutera można zduplikować, na przykład w celach diagnostycznych, używając słówka `dup-to`:

```
pass out quick on e1000g0 to e1000g1:10.0.1.1 dup-to e1000g2:172.16.0.1 \
    from 192.168.0.0/24 to any keep state
```

Rozważmy przykładową sytuację, gdy mamy dwóch dostawców usług sieciowych ISP. Router jednego dostawcy podłączony jest do interfejsu `e1000g0` i to jest trasa domyślna pakietów. Router drugiego podłączony jest do interfejsu `e1000g1`. Aby odpowiedzi na pakiety wchodzące przez `e1000g1` wychodziły przez `e1000g1` pomoże konstrukcja oparta o `reply-to`:

```
pass in on e1000g1 reply-to e1000g1:10.0.1.1 \
    proto tcp from any to 10.0.1.10/32 port = 22 flags S keep state
```

Odsyłanie informacji o błędzie

W sytuacji, gdy pakiet jest odrzucany przez reguły filtra, normalnie nie jest wysyłana żadna informacja zwrotna o tym fakcie. Z jednej strony nie chcemy generować dodatkowego ruchu w sieci, gdy blokujemy próby ataku. Z drugiej strony host próbujący połączyć się z nieużywanym portem TCP albo UDP, będzie ponawiał próby, sądząc że pakiet został zgubiony gdzieś po drodze.

W przypadku TCP można odesłać pakiet z flagą `RST`, aby uniknąć dalszych prób połączenia na dany port:

```
block return-rst in proto tcp from 10.0.0.0/8 to any
```

W przypadku pozostałych protokołów (także i TCP) można odesłać pakiet ICMP typu 3, czyli Destination Unreachable:

```
block return-icmp in proto tcp from 10.0.0.0/8 to any
```

Dodatkowo przy `return-icmp` można określić kod ICMP informujący dokładnie o rodzaju problemu. Może to być jeden z następujących kodów:

- `filter-prohib`,
- `net-prohib`,
- `host-unk`,
- `host-unr`,
- `net-unk`,
- `net-unr`,
- `port-unr`,

- `proto-unr`.

Ich nazwy są dość sugestywne. Który kod wybierzemy, zależy jaki efekt chcemy uzyskać i w jakim stopniu chcemy zdradzać istnienie zapory sieciowej na drodze do hosta docelowego.

```
block return-icmp(port-unr) in proto tcp from 10.0.0.0/8 to any
```

Powyżej wygenerowany pakiet ICMP jako adres źródłowy zawierać będzie adres interfejsu użytego do wysłania tego pakietu. Gdy chcemy ukryć istnienie zapory sieciowej i udawać, że odpowiada host docelowy, używamy następującej konstrukcji:

```
block return-icmp-as-des(port-unr) in proto tcp from 10.0.0.0/8 to any
```

Raportowanie

Są dwa sposoby raportowania informacji o przepływających przez IPFilter pakietach. Można użyć specjalnie do tego stworzonej reguły typy `log` albo do reguł pozostałych typów dodać specjalne słówko `log`. W ten drugi sposób można jednocześnie przepuścić lub zablokować pakiet i zapisać informację o tym w dzienniku:

```
pass in log quick proto tcp from any to any port = 22
```

Gdy stosujemy pamieć stanów, to do dziennika trafią informacje o wszystkich pakietach z danego połączenia. Może być tego trochę za dużo, a wystarczy informacja tylko o pierwszym pakiecie:

```
pass in log first quick proto tcp from any to any port = 22 \
    flags S keep state
```

Normalnie IPFilter zapisuje w dzienniku wyłącznie informacje z nagłówka pakietu. Gdy potrzebujemy zapisać dodatkowe informacje zawierające przesyłane dane, stosujemy słówko `body`:

```
block in log body proto icmp all
```

Pozwala to dodatkowo zapisać w dzienniku do 128 bajtów danych przesyłanych w pakiecie.

1.3.2 ipnat.conf

Plik /etc/ipf/ipnat.conf zawiera reguły opisujące zachowanie podsystemu translacji adresów. Do listy wewnętrznej reguły ładowane są za pomocą programu ipnat. Parametr -f, podobnie jak w ipf, pozwala wskazać inny plik lub standardowe wejście zamiast domyślnego pliku. Poniższy opis został opracowany na podstawie dokumentacji [3]. Przedstawione tutaj zostały tylko wybrane konstrukcje.

Reguły NAT zbudowane są w ten sposób, że najpierw występuje słowo kluczowe określające rodzaj mapowania adresów, za nim znajdują się deklaracje pozwalające dopasować pakiet, dalej jest symbol ->, a za nim deklaracje mówiące co ma być wpisane do pakietu.

Translacja adresów źródłowych

Ten rodzaj translacji w terminologii IPFiltra określa się jako *mapowanie*. W mapowaniu zarówno adres IP, jak i port mogą zostać zmienione. Najprostsza tego typu reguła może wyglądać następująco:

```
map e1000g0 0/0 -> 0/32
```

Mówi ona, aby przemapować źródłowe adresy IP wszystkich pakietów wychodzących (outbound) przez interfejs e1000g0, na adres IP, jaki w danym momencie ma ten interfejs. Wartość 0/0 po lewej stronie pasuje do wszystkich pakietów, natomiast 0/32 oznacza bieżący adres interfejsu e1000g0.

Gdy potrzebujemy przemapować tylko część ruchu wychodzącego przez e1000g0, na przykład z lokalnej podsieci 192.168.0.0/24 na konkretny adres zewnętrzny 10.0.0.10, możemy to wykonać tak:

```
map e1000g0 192.168.0.0/24 -> 10.0.0.10/32
```

To proste przemapowanie samego tylko adresu źródłowego szybko doprowadzi do problemów w protokołach TCP i UDP. Ponieważ numery portów nie są mapowane, więc gdy kilka hostów z sieci lokalnej będzie nawiązywać połączenia z tym samym źródłowym portem, to po mapowaniu pojawi się wiele takich samych par adres/port. Rozwiązuje się to, stosując dyrektywę portmap po prawej stronie:

```
map e1000g0 192.168.0.0/24 -> 10.0.0.10/32 portmap tcp/udp 10000:30000
```

W ten sposób źródłowe numery portów TCP/UDP będą mapowane na zakres [10000, 30000].

Przy niektórych usługach translacja adresów musi być wykonana w dość przemyślany sposób. Dotyczy to między innymi protokołu FTP, w którym do komunikacji używane są dwa porty 20 (do transferu danych) i 22 (do przekazywania komend). W tej sytuacji można zastosować tak zwane proxy:

```
map e1000g0 192.168.0.0/24 -> 10.0.0.10/32 proxy port ftp ftp/tcp
```

Translacja adresów docelowych

Translacja adresów docelowych zwana jest *redyrekcją*. Zasady konstruowania takich reguł są podobne jak przy mapowaniu. Najlepszym chyba przykładem, gdy potrzebujemy zastosować redyrekcję, jest przekierowanie portu 9122 routera na port 22 hosta o adresie `192.168.0.100` w sieci lokalnej:

```
rdr e1000g0 0/0 port 9122 -> 192.168.0.100 port 22 tcp
```

IPFilter dostarcza mnóstwo dodatkowych opcji wspomagających konstruowanie skomplikowanych reguł translacji adresów. Reguły typu `rewrite` pozwalają zmieniać jednocześnie adresy źródłowe i docelowe, a także porty.

Kolejność reguł

Ładowanie reguł do jądra systemu odbywa się w tej kolejności, w jakiej zostały umieszczone w pliku konfiguracyjnym. Dopasowanie jednak odbywa się według maski, od 32 do 0, to znaczy, że bardziej szczegółowe reguły zostaną dopasowane wcześniej. Następujące reguły:

```
rdr e1000g0 192.0.0.0/8 port 80 -> 127.0.0.1 3132 tcp
rdr e1000g0 192.2.0.0/16 port 80 -> 127.0.0.1 3131 tcp
rdr e1000g0 192.2.2.0/24 port 80 -> 127.0.0.1 3129 tcp
rdr e1000g0 192.2.2.1 port 80 -> 127.0.0.1 3128 tcp
```

zostaną dopasowane w tej kolejności:

```
rdr e1000g0 192.2.2.1 port 80 -> 127.0.0.1 3128 tcp
rdr e1000g0 192.2.2.0/24 port 80 -> 127.0.0.1 3129 tcp
rdr e1000g0 192.2.0.0/16 port 80 -> 127.0.0.1 3131 tcp
rdr e1000g0 192.0.0.0/8 port 80 -> 127.0.0.1 3132 tcp
```

1.3.3 `ippool.conf`

Plik `/etc/ipf/ippool.conf` zawiera konfigurację podsystemu puli adresów. Pule adresów ładowane są do jądra systemu za pomocą programu `ippool`. Opcja `-f` pozwala załadować konfigurację z innego, niestandardowego pliku.

W dokumentacji [3] znajdziemy pełny opis wielu możliwych scenariuszy, w których stosuje się różne rodzaje puli adresów. Tutaj ograniczymy się do jednego tylko rodzaju puli adresów używanych do dopasowywania adresów źródłowych lub docelowych w regułach filtra `ipf.conf`.

Typowa deklaracja puli adresów dla `ipf` w postaci drzewa (tree) o nazwie `trusted` może wyglądać tak:

```
pool ipf/tree (name trusted;) { 1.1.1.1/32; 2.2.0.0/16; !2.2.2.0/24; };
```

W nawiasach klamrowych podajemy listę adresów. Rozdzielamy je średnikami. Znak `!` to negacja. Możemy podawać adresy pojedynczych hostów lub całych podsieci zgodnie z notacją CIDR. Czasem wygodniej jest mieć tę listę adresów w osobnym pliku:

```
pool ipf/tree (name trusted;) { "file:///etc/ipf/trusted.pool"; };
```

1.3.4 Parametry tuningowe

Poza zestawem reguł, do swoich potrzeb można dostrajać wartości licznych parametrów IPFiltra, takich jak na przykład: rozmiar tablicy stanów `state_size`, rozmiar tablicy translacji adresów `nat_table_size`, rozmiar bufora dziennika `log_size`. Pełna lista parametrów, uzyskana poleceniem `ipf -T list`, znajduje się w dodatku B. Podane są tam kolejno: nazwa parametru, wartość minimalna, wartość maksymalna oraz wartość bieżąca.

Rozdział 2

Failover oraz High Availability

2.1 Failover

Failover jest bardzo ważnym składnikiem przy konstruowaniu systemu w firmie [10]. Nieprzerwana praca jest podstawą funkcjonowania firmy, a jakiekolwiek przestoje spowodują duże straty pieniężne. Faiolver jest to tryb pracy zapasowego serwera, który zostaje uruchamiany automatycznie, gdy dochodzi do awarii serwera podstawowego. Dochodzi wtedy do przekierowania pracy z uszkodzonego sprzętu do sprawnego, drugiego urządzenia, powodując tym samym brak przestojów w pracy. Takie rozwiązania stosowane są podczas projektowania serwerowni, czy w sieciach komputerowych z wymogiem stałego połączenia. Failover możemy stosować by:

- Chronić bazę danych poprzez automatyczne przełączenie pracy serwera w przypadku awarii głównego serwera na serwer zapasowy.
- Umożliwić automatyczne przeprowadzenie zadań konserwacyjnych. Gdy zachodzi potrzeba przeprowadzenia konserwacji, aktualizacji lub kopii zapasowej na serwerze głównym, sprzęt samodzielnie przełącza pracę na serwer zapasowy.

Usługa failover funkcjonalnością przypomina mechaniczny przełącznik, z tą różnicą, że tutaj przełączenie odbywa się automatycznie, bez ingerencji czynnika ludzkiego, który wymagany jest w przypadku przełączników mechanicznych.

2.2 Single Point Of Failure

W skrócie SPOF, a w dosłownym tłumaczeniu *pojedynczy punkt awarii*, to pojedynczy element infrastruktury, którego awaria uniemożliwia pracę całego systemu. Może to być element sprzętowy jak router, switch albo serwer, ale może to być element oprogramowania. Zdarza się że występuje awaria jakiejś

usługi jak na przykład DNS, czy DHCP, ale może to być także jakaś aplikacja. Zazwyczaj mamy tutaj do czynienia z efektem domina, gdzie jeden składnik systemu ulegający awarii powoduje kolejne awarie i ostatecznie zatrzymuje pracę całego systemu, czasem całej firmy.

Przed wystąpieniem SPOF głównie chroni nas redundancja, inaczej zwana zwielokrotnieniem, która jest stosowana na wszystkich poziomach infrastruktury. Jest to podstawowe rozwiązanie stosowane do uzyskania High Availability.

2.3 Mean Time Between Failures

Skrót MTBF możemy znaleźć na wielu komponentach sprzętowych, w szczególności na dyskach twardych HDD, ale także na nośnikach SSD. Jest to *średni czas pomiędzy awariami*, który pozwala zmierzyć nie przerywalności procesu pracy. Wskaźnik MTBF jest jednym z głównych wskaźników sprawdzania efektywności w Total Productive Maintenance, czyli w *całkowitym produktywnym utrzymaniu ruchu maszyn*. Pozwala on określić czas bezawaryjności i osiągi używanego sprzętu.

2.4 Mean Time To Repair

W skrócie MTTR, to *średni czas potrzebny do usunięcia awarii* od jej powstania [7]. Jest on mierzony wedle ilorazu czasu trwania awarii i liczby zdarzeń naprawczych. Podajemy go zwykle w minutach lub godzinach. Przykładowo, gdy awarii uległy 2 maszyny, naprawa pierwszej trwała 6 godzin, drugiej zaś 3 godziny, wówczas dzięki tym danym możemy wyliczyć wskaźnik MTTR. Wynosi on $(6 + 3)/2 = 4,5$ godziny. Tak obliczona dostępność zazwyczaj przedstawia się, mówiąc żargonowo, w postaci *ilości dziewiątek*.

Jak widzimy w tabeli 2.1, dostępność na poziomie 90 procent (jedna dziewiątka) sprawia, że serwer w ciągu roku będzie nieczynny prawdopodobnie przez 36,53 dnia, co w przełożeniu na dzień mówi nam o niedostępności około 2,4 godziny. Jest to poziom, przy którym nasza firma nie będzie mogła pracować średnio przez prawie 17 godzin w tygodniu, co wiąże się z ogromnymi stratami.

Idąc dalej, przy 99-procentowej niezawodności serwera nasza firma będzie miała problem z dostępem tylko około 10,08 minut w tygodniu. Jest to znaczna poprawa w porównaniu do poprzedniego scenariusza. Adekwatnie, czas przestoju będzie malał wraz ze wzrostem poziomu dostępności. Przy pięciu dziewiątkach ten poziom w skali tygodnia to raptem 6 sekund.

Na określenie dostępności wyrażonej w procentach składa się wiele czynników [8]:

Availability %	Downtime per year[note 1]	Downtime per month	Downtime per week	Downtime per day
90% ("one nine")	**36.53 days**	**73.05 hours**	**16.80 hours**	**2.40 hours**
95% ("one and a half nines")	18.26 days	36.53 hours	8.40 hours	1.20 hours
97%	10.96 days	21.92 hours	5.04 hours	43.20 minutes
98%	7.31 days	14.61 hours	3.36 hours	28.80 minutes
99% ("two nines")	**3.65 days**	**7.31 hours**	**1.68 hours**	**14.40 minutes**
99.5% ("two and a half nines")	1.83 days	3.65 hours	50.40 minutes	7.20 minutes
99.8%	17.53 hours	87.66 minutes	20.16 minutes	2.88 minutes
99.9% ("three nines")	**8.77 hours**	**43.83 minutes**	**10.08 minutes**	**1.44 minutes**
99.95% ("three and a half nines")	4.38 hours	21.92 minutes	5.04 minutes	43.20 seconds
99.99% ("four nines")	**52.60 minutes**	**4.38 minutes**	**1.01 minutes**	**8.64 seconds**
99.995% ("four and a half nines")	26.30 minutes	2.19 minutes	30.24 seconds	4.32 seconds
99.999% ("five nines")	**5.26 minutes**	**26.30 seconds**	**6.05 seconds**	**864.00 milliseconds**
99.9999% ("six nines")	**31.56 seconds**	**2.63 seconds**	**604.80 milliseconds**	**86.40 milliseconds**
99.99999% ("seven nines")	**3.16 seconds**	**262.98 milliseconds**	**60.48 milliseconds**	**8.64 milliseconds**
99.999999% ("eight nines")	**315.58 milliseconds**	**26.30 milliseconds**	**6.05 milliseconds**	**864.00 microseconds**
99.9999999% ("nine nines")	**31.56 milliseconds**	**2.63 milliseconds**	**604.80 microseconds**	**86.40 microseconds**

Rysunek 2.1: Przerwy w pracy systemu w podziale na okres czasu [12].

(i) Istotne jest to, z jakiego sprzętu korzystamy. Jakość sprzętu przekłada się na możliwość wystąpienia awarii.

(ii) Odpowiednie oprogramowanie oraz przetestowanie poprawności pracy.

(iii) Dobór administratorów i serwisantów, którzy będą sprawować opiekę w razie awarii. Ich czas reakcji oraz biegłość w obsłudze systemu.

2.5 Service Level Agreement

Po polsku *umowa o gwarantowanym poziomie świadczenia usług*, jest umową między dystrybutorem a klientem dotyczącą poziomu jakości świadczenia usług [6]. Spisana umowa zawiera zapisy, które dystrybutor umawia się zapewnić klientowi. Główną funkcją SLA jest zapewnienie usunięcia problemu w określonym czasie. Poza tym jest to także zbiór deklaracji pomiędzy usługodawcą a usługobiorcą. Takie zapisy zawierają wytyczne o czasie reakcji serwisu, sposobie przeprowadzenia naprawy itp. SLA jest wyznaczany na podstawie poziomu usługi, zaś jego wykonanie mierzy w CSI, czyli Continual Service Improvement. CSI to proces ustawicznego doskonalenia usług oraz dostosowanie ich do zmieniających się potrzeb firmy. Najprostsza umowa SLA obejmuje warunki, w których powinien zostać określony czas niedostępności usługi wyliczony ze wzoru na dostępności. Wzór na dostępność prezentuje się następująco:

$$\text{DOSTĘPNOŚĆ} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}.$$

2.6 High Availability

Dla wielu przedsiębiorców nieprzerwana dostępność do danych, jak i aplikacji to podstawa funkcjonowania firmy [11]. Nie ma w tym nic nadzwyczajnego, skoro z informatyzacją spotykamy się obecnie na każdym kroku. Przerwanie pracy serwera, chociażby na krótki okres czasu może spowodować znaczne straty finansowe. Aktualnie spotkamy się z koniecznością połączenia sieciowego praktycznie wszędzie, na przykład w firmach branży IT, bankach, pocztach, urzędach państwowych, sklepach internetowych. Od niedawna w Polsce obowiązkowe jest połączenie kasy fiskalnej on-line w każdym sklepie. W takich rodzajach działalności nie można pozwolić sobie na awaryjność. Każdy przestój w pracy może również spowodować straty nie do nadrobienia dla firm pracujących całe dnie. Rozwiązaniem tych problemów, jest uzyskanie High Availability, w skrócie HA, co w dosłownym tłumaczeniu na polski oznacza *wysoką dostępność*.

Na komercyjnym rynku High Availability, to możliwość, która robi niebywałą karierę w systemach serwerowych. Aktualnie prawie każdy dystrybutor Unixa posiada swoje oprogramowanie zapewniające HA, które udostępnia swoim użytkownikom po stosunkowo niskich cenach.

Redundancja jest podstawą High Availability. Gwarantuje ona unikanie sytuacji, w której występują pojedyncze punkty awarii (SPOF). Celem takiego zabiegu jest niedoprowadzenie do nieplanowanych przerw w pracy, tak aby użytkownik mógł wrócić, jak najszybciej do swoich zadań.

Innym wariantem jest stworzenie instrukcji, w której opisane jest, w jaki sposób administrator powinien postępować, aby wznowić pracę serwera. Jest to alternatywa, w której istnieje ryzyko, że pracownik wykonujący naprawę popełni błąd. Oprogramowanie HA daje nam możliwość, w której nie trzeba angażować czynnika ludzkiego w systemie naprawczym. Jest to opcja, która przy pomocy skryptów i narzędzi dokona tego samego co operator, lecz w o wiele krótszym czasie i bezbłędnie.

Gwarancja nieprzerwanej, ciągłej pracy systemu jest najważniejszym celem High Availability. Sprowadza się to do wyeliminowania jakichkolwiek przerw w dostawie usług sprzętowych w wyniku awarii, a w oprogramowaniu błędu użytkownika.

Brak dostępu do usług może być zaplanowany ze względu na przeprowadzenie modernizacji, sprawdzenie poprawności działania systemu, czy wymiany sprzętu na nowszy, albo niezaplanowany będący skutkiem awarii, przez co użytkownicy odczują brak poprawności pracy systemu. Te zdarzenia zostają uwzględnione w ustalonej z właścicielami systemu umowie SLA.

2.7 Fault Tolerance

High Availability nie jest jedyną koncepcją, dzięki której możliwe jest nieprzerwane, bezawaryjne świadczenie usług, bez ograniczeń ich dostępności. Inną opcją jest Fault Tolerance, czyli *tolerowanie awarii*. Według tej koncepcji zapewniona jest ciągła praca mimo wystąpienia jednego lub więcej zakłóceń, czy też awarii podczas pracy. Polega ona na pracy zazwyczaj dwóch (a może i więcej) systemów, które pracują jednocześnie, powielając swoje czynności. Znowu mamy tutaj do czynienia z redundancją. W momencie, gdy występuje awaria na głównym sprzęcie i system wykryje problem, przełącza się on wtedy na drugi, identyczny system nie powodując żadnych przestojów w pracy użytkownika.

Główną cechą Fault Tolerance jest fakt, że stan uruchomionych aplikacji oraz ich pamięć nie zostaje utracona w przypadku niesprawności systemu. Może jedynie wystąpić krótkie opóźnienie w przesłaniu danych, z powodu przełączania się systemu.

Takie zastosowanie systemu mogłoby wydawać się idealne. Niestety, gdyby to rozwiązanie było bez wad, każdy decydowałby się na jego zastosowanie. Czemu więc tak nie jest? Ponieważ takie systemy działają jak odbicia lustrzane. W przypadku wystąpienia błędu w oprogramowaniu na bazowym, pierwszym serwerze, wystąpi on również na pozostałych, bo system powiela dane. Następstwem tych zdarzeń będą błędy na poziomie aplikacji, które mogą doprowadzić do przestojów pracy.

2.8 Disaster Recovery

Disaster Rcovery, czyli dosłownie *dochodzenie do siebie po katastrofie*, to procedura związaną z przywróceniem działania systemu, którego praca została przerwana przez jeden z następujących czynników:

- naturalnych jak huragan, powódź, czy trzęsienie ziemi,
- środowiskowych jak awarie pobliskich zakładów,
- wywołanymi poczynaniami ludzkimi takimi jak pożar, kradzież, czy hackerstwo.

Przy formułowaniu planu działań podczas Disaster Rcovery musimy określić dwa następujące parametry:

RPO – Recovery Point Objective Informuje jaki czas upłynął od utworzenia ostatniej kopii danych. Im ta wartość jest większa, tym więcej danych zostanie utraconych w przypadku awarii.

RTO – Recovery Time Objective Informuje o tym jak długo potrwają prace nad przywróceniem systemu, do normalnego stanu.

Disaster Rcovery w głównej mierze oparte jest na wszelkich kopiach zapasowych. Dotyczy to nie tylko danych, ale także oprogramowania i jego konfiguracji. Chodzi o to, by jak najszybciej, w pewny sposób, przywrócić system do pracy. Trzeba tu wziąć pod uwagę, że w wyniku katastrofy nie będziemy w stanie uruchomić zniszczonego sprzętu, albo go wręcz nie będzie. To wymusza odtworzenie systemu na innym, zapasowym sprzęcie, być może w innej lokalizacji.

Aby chronić cenne dane, na potrzeby Disaster Rcovery wykorzystuje się replikację danych. Polega ona na stałym kopiowaniu danych do systemu zapasowego. Niektóre bazy danych, na przykład MySQL, posiadają takie mechanizmy wszyte, bez konieczności stosowania dodatkowego oprogramowania. Po katastrofie, gdy przywracamy system do pracy, wystarczy jako podstawowy użyć system zapasowy, albo pobrać dane z systemu zapasowego na odtwarzany system. Przy ogromnych bazach danych czas konieczny na skopiowanie danych po katastrofie może być nie do przyjęcia. W takich przypadkach warto zadbać, aby mieć kilka system zapasowych, najlepiej w różnych, odległych lokalizacjach.

2.9 Porównanie

Każdy chciałby metody zabezpieczeń danych, która jest w 100% sprawna, bezawaryjna, która nigdy się nie zawiesza i przede wszykim tania do wdrożenia. Firmy nie lubią ponosić dużych kosztów utrzymania. Życie to nie bajka i możemy skorzystać jedynie z bliskich ideałom metod, które zostały przedstawione powyżej.

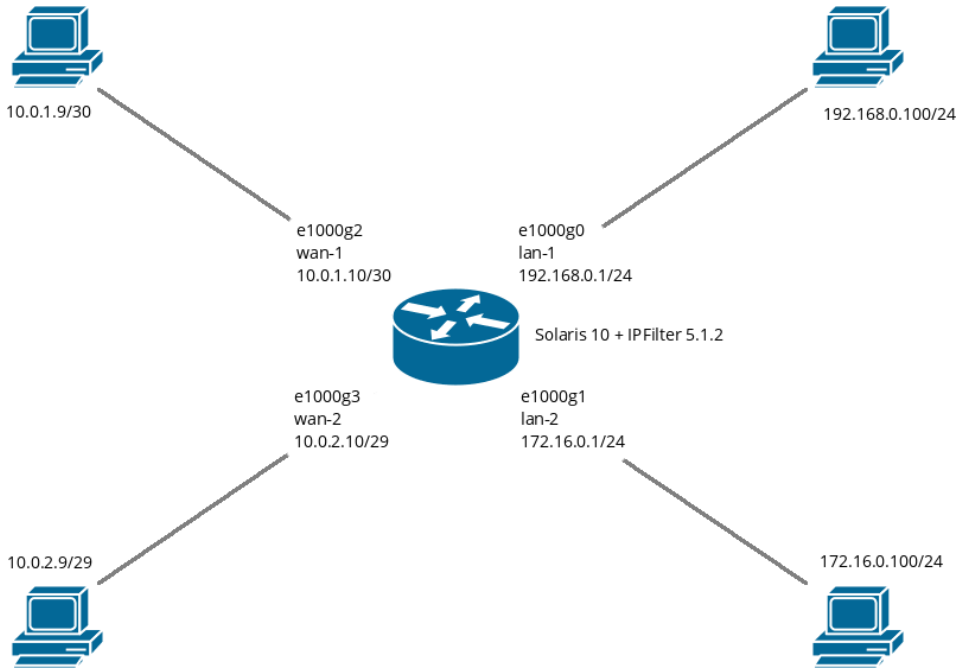
Główną różnicą między High Availability a Fault Tolerance jest to, że pierwsza koncepcja dopuszcza przerwy w działaniu usług a druga nie (por. [9], [13]). W praktyce, gdy przy High Availability wystąpi zakłócenie pracy serwera, prawdopodobnie dostaniemy komunikat o niemożliwości wykonania zadania, natomiast, gdy zaimplementujemy Fault Tolerance, nasz problem powinien zostać rozwiązany automatycznie przez przełączenie się między komponentami sprzętowymi, tak aby przerwa w ogóle nie wystąpiła. Fault Tolerance zasadniczo sprowadza się do redundancji wszystkich komponentów systemu, głównie sprzętu. W High Availabilty natomiast na system patrzy się jako na całość, dubluje się całe serwery i oprogramowanie tworząc klastry. W procesach produkcyjnych, medycznych, tam gdzie od maszyny zależy ludzkie zdrowie, lub wręcz życie, stosuje się znacznie kosztowniejsze rozwiązania Fault Tolerance. Niestety Fault Tolerance daje małe zabezpieczenie przed problemami z oprogramowaniem. Tam, gdzie usługa oparta jest w większym stopniu o oprogramowanie, a mniej o sprzęt, większy sens ma High Availability.

To, co charakteryzuje Disaster Recovery, to przede wszystkim niedrogie wdrożenie w firmie. System taki jedynie zabezpiecza dane przed utratą. Nie zapewnia ciągłości pracy, lecz jedynie w miarę łatwe przywrócenie działalno-

ści. O ile Fault Tolerance czy High Availability polega na redundancji sprzętu, to Disaster Recovery oparte jest na kopiach zapasowych i replikacji danych. Zawsze jest dodatkowy koszt, także i tutaj. Gdy potrzebujemy ustrzec nasze cenne dane przed poważnymi katastrofami, powinniśmy zainwestować w dodatkowe serwery zapasowe oddalone od nas daleko geograficznie.

2.10 Zastosowanie Failover

Celem pracy było opracowanie metody realizującej Failover dla sieci WAN na niedużym routerze. Do testowania wykorzystaliśmy wirtualne środowisko zbudowane w oparciu o VirtualBox. To samo środowisko wykorzystywane jest w pracy [2]. Na rysunku 2.2 mamy diagram połączeń między hostami a routerem oraz podstawowe parametry sieciowe.



Rysunek 2.2: Diagram sieci do testowania Failover i IPFilter.

Mamy tutaj dwóch dostawców usług sieciowych:

- `WAN1` o adresie bramy `10.0.1.9`, oraz
- `WAN2` o adresie bramy `10.0.2.9`.

Domyślna trasa pakietów biegnie przez `WAN1`. Naszym zadaniem jest automatyczne przełączenie naszego routera na `WAN2`, gdy stwierdzimy, że na `WAN1` jest awaria. W odróżnieniu od [2] zakładamy, że cały ruch odbywa się przez jednego dostawcę, a drugi traktowany jest jako zapasowy.

W tym celu opracowany został skrypt `swan` realizujący przełączenie z jednego dostawcy na drugiego, w zależności od tego, który aktualnie jest używany. Kod skryptu przedstawiony jest poniżej.

```
#!/bin/sh

PATH=/usr/bin:/usr/sbin
export PATH

WAN1=10.0.1.10
WAN2=10.0.2.10

ROUTER=`netstat -rn | grep default | awk '{print $2}'`

if [ "$ROUTER" = "$WAN1" ]
then
    OLDROUTE=$WAN1
    NEWROUTE=$WAN2
    WAN=wan2
else
    OLDROUTE=$WAN2
    NEWROUTE=$WAN1
    WAN=wan1
fi

route delete default $OLDROUTE
route add default $NEWROUTE
echo $NEWROUTE > /etc/defaultrouter

(cd /etc/ipf

ln -fs ipf-$WAN.conf ipf.conf
ln -fs ipnat-$WAN.conf ipnat.conf

ipf -F a -f ipf.conf
ipnat -F -C -f ipnat.conf

)
```

Przełączenie polega na usunięciu aktualnej trasy domyślnej pakietów przy pomocy polecenia `route delete` i dodaniu nowej trasy domyślnej za pomocą `route add`. Na wszelki wypadek adres bramy zapisujemy w standardowym dla Solaris pliku `/etc/defaultrouter` tak, aby po ewentualnym restarcie używana była nowa trasa. Na zakończenie musi być przełączony IPFilter. Korzystanie z więcej niż jednego dostawcy ISP wymaga zaimplementowania Policy Routing w IPFilter, opisywanego w pracy [2]. W efekcie konfiguracja filtra pakietów w pliku `/etc/ipf/ipf.conf` oraz konfiguracja translacji adresów w pliku `/etc/ipf/ipnat.conf` są różne dla różnych dostawców. Dlatego mamy dwa pliki `ipf-wan1.conf` i `ipf-wan2.conf` oraz dwa pliki `ipnat-wan1.conf` i `ipnat-wan2.conf`. Zostały one załączone w dodatkach: C, D, E, F. Link

symboliczny prowadzi do aktualnie używanego pliku w zależności od wyboru dastawcy. Poniżej pokazano listę plików w katalogu `/etc/ipf`:

```
-rw-r--r--   1 root   root    378 Feb 14  2020 intruders.pool
-rw-r--r--   1 root   root   6677 Feb  9  2020 ipf-wan2.conf
-rw-r--r--   1 root   root   7822 Feb 23  2020 ipf-wan1.conf
lrwxrwxrwx   1 root   root     12 Feb  9  2020 ipf.conf -> ipf-wan1.conf
-rw-r--r--   1 root   root    636 Feb 14  2020 ipnat-wan2.conf
-rw-r--r--   1 root   root    636 Feb 29  2020 ipnat-wan1.conf
lrwxrwxrwx   1 root   root     14 Feb  9  2020 ipnat.conf -> ipnat-wan1.conf
-rw-r--r--   1 root   root    504 Feb 23  2020 ippool.conf
-rw-r--r--   1 root   root     97 Feb 23  2020 spoofing.pool
-rw-r--r--   1 root   root    288 Feb 19  2020 trusted.pool
```

Skrypt `swan` może być uruchomiony ręcznie przez administratora, gdy zajdzie taka konieczność. Zasadniczym elementem naszego systemu Failover jest skrypt automatycznie próbujący i decydujący o przełączeniu dostawcy. Do próbkowania wykorzystujemy zwykłe polecenie `ping` na bramę aktualnego, domyślnego dostawcy, którą odczytujemy za pomocą polecenia `netstat`. To gwarantuje, że zawsze używamy aktualnej bramy nawet wtedy, gdy administrator ręcznie przełączy sieci WAN. Pomiędzy kolejnymi testami robimy przerwę 10 sekund. Jeśli 3 kolejne próby będą nieudane to program wykona skrypt `swan` przełączając dostawcę. Poniżej załączam kod realizującego to skryptu o nazwie `failover`:

```
#!/bin/sh

PATH=/usr/bin:/usr/sbin
export PATH

DELAY=10
MAXFAILURES=3

FAILURES=0

while true
do
    ROUTER=‘netstat -rn | grep default | awk ’{print $2}’‘
    ping $ROUTER > /dev/null
    if [ $? -ne 0 ]
    then
        FAILURES=‘expr $FAILURES + 1‘
    else
        FAILURES=0
    fi
    if [ $MAXFAILURES -le $FAILURES ]
    then
        /opt/bin/swan
    fi
    sleep $DELAY
done
```

Nowy dostawca, po przełączeniu, będzie tak długo używany, aż u niego wystąpią jakieś problemy i skrypt z powrotem przełączy router na wcześniejszego dostawcę. Gdyby dostawców było więcej, wystarczy zmodyfikowac odpowiednio skrypt `swan` oraz dopisać konfigurację IPFiltra.

Aby skrypt `failover` działa nieprzerwanie, także po restarcie serwera, należało opracować specjalną usługę wykorzystując SMF, czyli Service Management Facility w Solaris. W tym celu wystarczy przygotować odpowiedni plik XML (tak zwany manifest) opisujący naszą usługę, którą nazwaliśmy `svc:/network/failover`. Manifest został załączony poniżej:

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">

<service_bundle type='manifest' name='CFWfailover:failover'>

<service
    name='network/failover'
    type='service'
    version='1'>

    <single_instance />

    <dependency
        name='fs-local'
        grouping='require_all'
        restart_on='none'
        type='service'>
        <service_fmri value='svc:/system/filesystem/local' />
    </dependency>

    <dependency
        name='network-service'
        grouping='require_all'
        restart_on='none'
        type='service'>
        <service_fmri value='svc:/network/service' />
    </dependency>

    <dependency
        name='ipfilter'
        grouping='require_all'
        restart_on='refresh'
        type='service'>
        <service_fmri value='svc:/network/ipfilter' />
    </dependency>

    <instance name='failover' enabled='false'>

        <exec_method
            type='method'
            name='start'
```

```
            exec='/opt/bin/failover'
            timeout_seconds='5' />

            <exec_method
                type='method'
                name='stop'
                exec=':kill'
                timeout_seconds='5'>

        <template>
            <common_name>
                <loctext xml:lang='C'>
                    WAN failover and IPFilter switch
                </loctext>
            </common_name>
        </template>

    </instance>

</service>

</service_bundle>
```

Do działania naszej usługi `failover` konieczne jest, aby wcześniej zamontowane zostały wszystkie, lokalne systemy plików, za co odpowiada usługa o nazwie `svc:/system/filesystem/local`, aby zostały skonfigurowane połączenia sieciowe, za co z kolei odpowiada `svc:/network/service` oraz aby uruchomiony był IPFilter, czyli `svc:/network/ipfilter`. Za spełnienie tych zależności zadba już SMF w systemie Solaris.

Rozdział 3

Load Balancing

W latach dziewięćdziesiątych XX wieku, gdy Internet stał się powszechny na dużą skalę, pojawiły się problemy z przepustowością sieci [15]. Niektóre z serwerów nie dawały sobie rady z obciążeniem, przez co zaczęły występować problemy z aplikacjami. Wtedy wymyślono Load Balancing, którego zadaniem jest rozprowadzenie ruchu sieciowego jednakowo między serwery. Dzięki temu rozwiązaniu, każdy z nich zostanie równomiernie obciążony.

3.1 Strategia dystrybucji obciążenia

Load Balancing poprawia szybkość ładowania stron internetowych oraz pracę aplikacji [14]. Monitorowany jest na okrągło status dostępnych zasobów oraz ich podział. Wraz z biegiem czasu Load Balancing był nieustannie udoskonalany. Dzięki temu teraz również wpływa na poprawę bezpieczeństwa w Internecie.

Kiedy firmie udaje się zapewnić poprawną pracę na poziomie aplikacji, wówczas Load Balancing decyduje, który z puli równoważnych serwerów zapewni obsługę ruchu z sieci. Usługa Load Balancing zarządza przepływem danych pomiędzy serwerem a klientem, akceptując przychodzący ruch sieciowy oraz dzieląc go według zaprogramowanego schematu. Dzięki temu wystąpienie awarii na jednej z maszyn nie powoduje przerwy w dostępie do danych i aplikacji. Gdy dojdzie do czarnego scenariusza i sprzęt ulegnie awarii, Load Balancer automatycznie i równomiernie rozprowadzi pracę niedostępnego serwera na pozostałe. W ten sposób do minimum zostaje zredukowana możliwość przerwania pracy całego systemu. Możemy tu mówić w pewnym sensie o High Availability oraz Fault Tolarance.

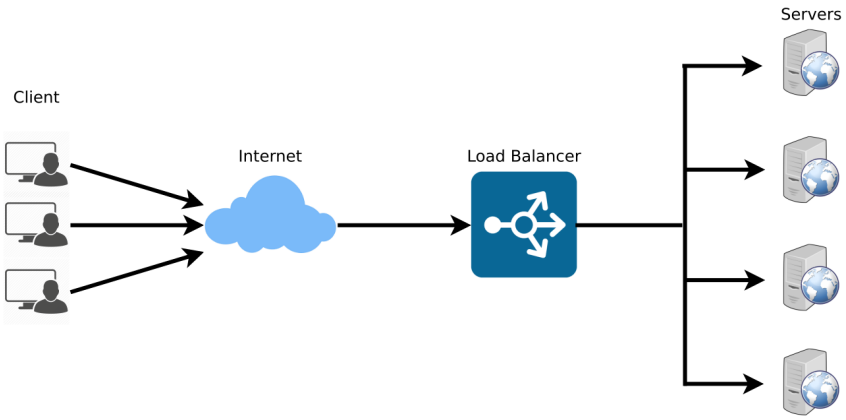
W aplikacji webowej zwykle występują trzy następujące warstwy:

1. serwer webowy przetwarzający żądania od klientów, położony najbardziej na zewnątrz systemu,

2. serwer aplikacji wykonujący operacje, umieszczony wewnątrz systemu,

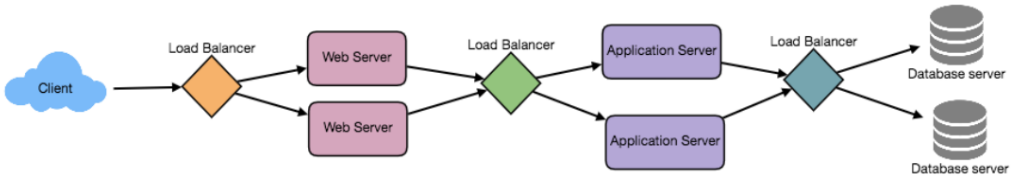
3. serwer bazy danych gromadzący dane, umieszczony na skraju systemu.

Mogą to być trzy programy, na przykład Apache, PHP i MySQL odpowiednio, na jednej maszynie albo grupie bliźniaczych maszyn. W większych przedsięwzięciach będą to nie tylko różne maszyny, ale różne grupy maszyn.



Rysunek 3.1: Typowe zastosowanie pojedynczego Load Balancera [14].

W pierwszym opisywanym przypadku Load Balancer umieszczony jest pomiędzy klientami a wielozadaniowymi serwerami jak na rysunku 3.1. Jeśli trzy warstwy aplikacji realizowane są na jednej maszynie, jest to jedyny sensowny sposób równomiernego rozłożenia obciążenia.



Rysunek 3.2: Zaawansowane zastosowanie Load Balancera [14].

Aby w pełni użyć skalowalności i redundancji, musimy jednakowo obciążyć każdą z trzech warstw systemu. W tym celu optymalnie jest umieścić Load Balancer tak jak to pokazano na rysunku 3.2. Każdy z tych trzech modułów Load Balancera będzie miał inną charakterystykę, ale sama idea równomiernego rozłożenia obciążenia w każdym przypadku pozostaje ta sama. Pierwszy dystrybuuje wyłącznie ruch wchodzący, drugi dystrybuuje operacje na aplikacji, trzeci dystrybuuje zapytania do bazy danych.

Dzięki Load Balancing użytkownik jest w stanie wykonać szybciej swoją pracę. Nie musi czekać, aż przeciążony serwer obsłuży jego zadania. Automatycznie jego żądania będą obsługiwane przez inną mniej obciążoną maszynę.

Korzyścią z używania Load Balancing są mniejsze przestoje w pracy oraz większa przepustowość. W przypadku problemów z serwerami odbiorca usług nie powinien doznać przerw w pracy.

Load Balancer analizuje ruch sieciowy, dzięki czemu jest w stanie przewidzieć, gdzie w danym momencie mogą wystąpić zatory. Analiza ta pozwala przygotować się firmie na dotknięcie problem i podjąć odpowiednią decyzję w celu uniknięcia kłopotów. Load Balancing sprawia, że serwery fizyczne dzięki podzielonej pracy doświadczają mniejszego obiążenia, co przekłada się na rzadsze występowanie awarii. Lepiej, gdy przykładowo 5 serwerów działa z 20% obciążeniem niż jeden, który musi pracować na pełnych obrotach. Zmniejszamy dzięki temu ryzyko wystąpienia problemów w firmie.

3.2 Wybór serwera przez Load Balancer

Usługa Load Balancing przed wysłaniem danych do serwera dokonuje dwóch analiz. Pierwszą z nich jest, sprawdzenie, czy serwer do którego mają zostać wysłane, dane w ogóle odpowiada. Kolejnym krokiem jest przeanalizowanie dostępnych serwerów i wybranie spośród nich odbiorcy.

Zdrowy serwer to taki, który nasłuchuje, czy nie dochodzą do niego jakieś zapytania. Jeżeli wystąpi sytuacja, w której jeden z serwerów backendowych nie jest w stanie nadać odpowiedzi zwrotnej, zostaje on wtedy usunięty z grupy odbiorców. W takiej sytuacji zostaje on poza pulą dostępnych serwerów, dopóki system sam go nie naprawi lub nie zrobi tego administrator.

Load Balancing to technologia, która wykorzystuje kilka różnych algorytmów do osiągnięcia równomiernego rozłożenia obciążęnia.

Metoda najmniejszego obciążenia

Ta metoda przekierowuje ruch do serwera, który jest połączony z najmniejszą liczbą klientów. Jest ona użyteczna tam, gdzie występuje dużo, nierównomiernie rozdystrybuowanych, stałych połączeń z klientami.

Metoda najkrótszego czasu odpowiedzi

W tej metodzie Load Balancer kieruje ruch do serwera, który w danym momencie połączony jest z najmniejszą liczbą klientów i jednocześnie ma najkrótszy czas reakcji.

Metoda najmniejszej przepustowości

Tutaj preferowny jest serwer, który w danym momencie generuje najmniejszy ruch w sieci mierzony w megabajtach na sekundę.

Metoda Round Robin

W tej metodzie serwery wybierane są kolejno z ustalonej listy. Gdy Load Balancer dotrze do końca listy, proces jest zapętlany i cykl dytrybułowania żądań rozpoczyna się od nowa. Ten sposób pracy stosowany jest zazwyczaj, gdy parametry naszych serwerów nie różnią się zbytnio od siebie i nie ma w nim wielu stałych połączeń.

Metoda ważonego Round Robin

Ta metoda została zaprojektowana tak, aby lepiej obsługować serwery o zróżnicowanej wydajności. Różni się tym od poprzedniej, że tutaj każdy z serwerów ma przypisaną wartość całkowitą określającą jego wydajność. Dzięki takiemu zabiegowi wydajniejsze maszyny otrzymują większą liczbę zadań niż te mniej wydajne. Ta metoda ustala hierarchię w systemie optymalizując w ten sposób ich pracę.

Metoda IP Hash

Źródłowy i docelowy adres IP przekształacany jest tutaj w hash. Algorytm haszujący bierze pod uwagę ilość serwerów oraz ich wagi. Na podstawie klucza hash wybierany jest serwer z puli. Ta metoda może być bardzo skuteczna, gdy adresy źródłowe są różne. W przeciwnym razie, wiele zapytań o tym samym źródłowym adresie IP zostanie skierowanych do tego samego serwera.

3.3 Load balancing w IPFilter

IPFilter nie został zaprojektowany z myślą o dystrybucji obciążenia. Nie ma w nim, tak jak na przykład w IPTables, modułu statystycznego, który pomagałby realizować Load Balancing. W IPFilter jest w zasadzie możliwa do uzyskania tylko jedna prosta metoda Round Robin.

Redyrekcja, czyli translacja DNAT, pojedynczego portu routera na hosta w sieci lokalnej wygląda następująco (por. sekcja 1.3.2):

```
rdr e1000g2 0/0 port 80 -> 192.168.0.100 port 80 tcp
```

W tym akurat przykładzie przekierowujemy cały ruch HTTP trafiający na router do komputera o adresie `192.168.0.100` w sieci lokalnej. W sytuacji, gdy dysponujemy farmą serwerów gotowych do obsługi HTTP możemy ten ruch równomiernie rozłożyć. Powiedzmy, że mamy 3 serwery w naszej farmie o adresach `192.168.0.100`, `192.168.0.101` i `192.168.0.102`. Wówczas następujący zestaw reguł realizuje metodę Round Robin:

```
rdr e1000g2 0/0 port 80 -> 192.168.0.100 port 80 tcp round-robin
rdr e1000g2 0/0 port 80 -> 192.168.0.101 port 80 tcp round-robin
rdr e1000g2 0/0 port 80 -> 192.168.0.102 port 80 tcp round-robin
```

W jednej linii nie można umieszczać więcej niż dwóch adresów docelowych. Umieszczenie każdego hosta w osobnej linii daje bardziej czytelną konfigurację. Poza tym łatwo wtedy możemy usunąć pojedynczego hosta na przykład w przypadku awarii. Niestety pule adresów, które mogły być tutaj bardzo wygodne, nie działają z regułami NAT.

Rozdział 4

IP Multipathing

W każdej firmie ważne jest, by zminimalizować, a najlepiej wyeliminować, możliwości jakichkolwiek awarii. Pisaliśmy już o różnych metodach jak Failover, czy High Availability. W Solaris mamy jeszcze IPMP, czyli IP Multipathing, w tłumaczeniu *wielościeżkowość* (por. [4] oraz [5]). Mechanizm ten łączy w sobie cechy Failover oraz High Availability, a także Load Balancing. Generalnie polega on na konstruowaniu redundantnych interfejsów sieciowych. Co więcej, zwiększając szerokość pasma sieciowego, rozkłada równomiernie ruch między połączenia sieciowe. Aby wykorzystać IPMP musimy, posiadać więcej niż jedną kartę sieciową podłączoną do jednej podsieci.

Solaris wykorzystuje przypisane statycznie testowe adresy IP do badania stanu interfejsu sieciowego. Zadaniem IPMP jest nasłuchiwanie sygnałów echa nadawanych okresowo przez interfejsy testowe i wysyłanie im odpowiedzi. Jeżeli dojdzie do sytuacji, gdzie serwer nie odpowiada przez określony czas, takie połączenie zostaje przerwane. IPMP w takiej sytuacji przechodzi w tryb Failover dla całej puli adresów IP i przenosi pracę aplikacji na inny fizyczny interfejs. Takie działanie zapobiega awariom całych sieci.

4.1 Budowa IPMP

W IPMP mamy następujące elementy składowe:

Demon IPMP Program `in.mpathd` będący sercem systemu IPMP. Wykrywa awarie na dwa sposoby. Pierwszą metodą jest odbieranie echa przez ICMP, druga jest monitorwanie flagi RUNNING przy interfejsie sieciowym. Gdy dochodzi do wykrycia problemu demon decyduje o przełączeniu się na inny, zapasowy interfejs.

Usługa IPMP Usługa SMF odpowiadająca za rozruch i zatrzymanie demona.

Plik konfiguracyjny Zbiór reguł odpowiadających za pracę demona zapisany w pliku `/etc/default/mpathd`. Ustalane są tam parametry interfej-

sów, sposobu badania awarii oraz czasu ich trwania. Służy on także do sprawdzenia stanu sprzętu po awarii.

Polecenia administracyjne Zestaw programów do zarządzania pracą IPMP.

Polecenia informacyjne Zestaw programów narzędziowych wyświetlających informacje o stanie systemu IPMP.

IP Link

IP Link, czy łącze IP, to medium pozwalające na komunikację urządzeń w warstwie łącza danych. Typy łączy IP obejmują między innymi sieci Ethernet oraz ATM. Jedno łącze IP może mieć kilka różnych adresów bazowych podsieci IPv4 lub kilka prefiksów podsieci IPv6. Ten sam adres bazowy podsieci, czy prefix podsieci, nie może być przypisany do więcej niż jednego łącza IP. Z punktu widzenie protokołu ARP (Address Resolution Protocol), pojedyncze łącze IP to zakres, w jakim może działać ARP.

Interfejsy fizyczne

Interfejs fizyczny stanowi przyłącze systemu do łącza IP. To przyłącze to zwykle karta sieciowa oraz odpowiadający jej sterownik. Kiedy w systemie występuje wiele interfejsów podłączonych do tego samego łącza, to można wtedy skonfigurować IPMP w celu wykonania awaryjnego Failover, gdy któryś z interfejsów ulegnie awarii.

Karty sieciowe

Karta sieciowa to fizyczne urządzenie zainstalowane w maszynie, które może być użyte przez system. Istotne jest to, że niektóre karty sieciowe posiadają wiele fizycznych interfejsów: dwa (dual NIC) lub cztery (quad NIC).

Grupy IPMP

Grupa IPMP zawiera jeden lub więcej interfejsów fizycznych zainstalowanych w systemie. Interfejsy te są skonfigurowane z tą samą nazwą grupy. Wszystkie interfejsy wchodzące w skład jednej grupy IPMP muszą być połączone do tego samego łącza IP. W jednej grupie IPMP można umieszczać interfejsy o różnych prędkościach, jeśli karty sieciowe są tego samego typu. W jednej grupie IPMP nie mogą się znaleźć interfejsy działające z różnymi typami mediów (Ethernet lub ATM).

Failure Detecion oraz Failover

Failure Detecion to proces wykrywania, kiedy interfejs ulega awarii lub zerwana zostaje ścieżka pomiędzy interfejsem a urządzeniem warstwy łącza danych

(przełącznikiem, routerem itp.). IPMP wykrywa następujące rodzaje awarii komunikacyjnych:

- Ścieżka transmisji lub odbioru interfejsu jest zerwana.
- Interfejs nie jest podłączony do łącza IP.
- Port w przełączniku nie transmituje ani nie odbiera pakietów.
- Interfejs fizyczny w grupie IPMP nie jest obecny podczas uruchamiania systemu.

Po wykryciu błędu rozpoczyna się proces przełączania awaryjnego Failover. Failover jest procesem automatycznym, który polega na przełączeniu dostępu do sieci z uszkodzonego interfejsu na inny, nieuszkodzony interfejs w grupie. Ten proces jest możliwy jedynie wtedy, gdy w grupie IPMP jest skonfigurowany więcej niż jeden interfejs. Failover zapewnia nieprzerwany dostęp do sieci.

Repair Detection oraz Failback

Repair Detection to proces wykrywania, kiedy karta sieciowa lub ścieżka prowadząca z karty sieciowej do urządzeń w sieci zaczynają działać bezbłędnie. Po wykryciu, że usterka została usunięta, IPMP wykonuje Failback. Failback to proces przywracania dostępu do sieci poprzez interfejs, który został naprawiony.

Systemy docelowe

Systemy docelowe są wykorzystywane do określenia, w jakim stanie jest interfejs w przypadku, gdy wykrywamy awarie na podstawie próbkowania. Każdy system docelowy musi być podłączony do tego samego łącza IP co członkowie w grupy IPMP. Demon `in.mpathd` wysyła komunikaty ICMP do każdego z systemów docelowych, aby określić stan interfejsów.

Rozłożenie obciążenia wyjściowego

Gdy mamy już skonfigurowany system IPMP wychodzące pakiety są rozłożone na różne karty sieciowe w taki sposób, że nie ma to wpływu na kolejność pakietów. Dzięki temu procesowi, uzyskuje się większą przepustowość.

Interfejsy zapasowe w grupach IPMP

Interfejs zapasowy w grupie IPMP nie jest używany do przesyłu danych, dopóki inny interfejs w tej grupie nie ulegnie awarii. Gdy wystąpi usterka, adres IP do przesyłu danych uszkodzonego interfejsu migrowany jest do interfejsu

zapasowego. Od tej pory interfejs zapasowy traktowany jest tak samo, jak inne aktywne interfejsy w grupie do momentu, aż usterka nie zostanie usunięta.

Na interfejsie zapasowy konfiguruje się wyłącznie adresy testowe. IPMP nie pozwoli skonfigurować na nich adresów IP do przesyłu danych.

Link-Based Failure Detection

Wykrywanie awarii na podstawie informacji ze sterownika, czy dany interfejs jest podłączony do urządzenia w sieci, jest domyślnie uruchomioną funkcją w systemie. Większość sterowników w Solaris obsługuje tę funkcjonalność. Dzięki takim sterownikom system IPMP jest w stanie w sposób ciągły monitorować stan łącza, a gdy ten stan ulegnie zmianie, przekazuje o tym wiadomość do pozostałych komponentów IPMP.

Probe-Based Failure Detection

Demon `in.mpathd` wykonuje sondowanie w celu wykrycia awarii na każdym interfejsie grupy IPMP, który posiada przypisany adres testowy. Wykrywanie awarii na podstawie sondy polega na wysyłaniu i odbieraniu komunikatów ICMP na adresy testowe. Te komunikaty wychodzą przez interfejs do jednego lub więcej systemów docelowych, które znajdują się na tym samym łączu IP.

Deamon `in.mpathd` określa dynamicznie, które systemy docelowe zostaną zbadane. Routery znajdujące się na danym łączu IP są wybierane automatycznie jako cele do próbkowania. Jeśli nie ma routerów na łączu IP, wówczas `in.mpathd` wysyła sondy do sąsiednich hostów na łączu. Do wszystkich hostów z adresami multikastowymi, to znaczy `224.0.0.1` w IPv4 oraz `ff02::1` w IPv6, wysyłany jest pakiet multikastowy, na podstawie którego określane są hosty, które mają być używane jako systemy docelowe. Jako cele do sondowania zostają wybrane pierwsze hosty, które zareagują. Jeśli demon `in.mpathd` nie wykryje routerów i hostów, które mu odpowiedzą na komunikaty ICMP, wtedy nie jest w stanie wykrywać awarii na podstawie sondowania.

Aby mieć pewność, że wszystkie interfejsy w grupie IPMP funkcjonują prawidłowo, `in.mpathd` sonduje każdy system docelowy osobno poprzez każdy interfejs w tej grupie. Interfejs jest uznawany za uszkodzony, jeśli nie odpowie na 5 kolejnych sond. Częstotliwość sondowania zależy od parametru FDT – Failure Detection Time. Domyślna wartość tego parametru to 10 sekund. Można oczywiście ten parametr zmodyfikować w konfiguracji.

W przypadku wykrywania naprawy, sondowanie odbywa się w przybliżeniu co dwie sekundy. Czas wykrycia naprawy jest dwukrotnie dłuższy od czasu znalezienia awarii i domyślnie wynosi 20 sekund, ponieważ musi być odebranych 10 kolejnych odpowiedzi na sondy.

Awaria grupy

Z awarią grupy mamy do czynienia, gdy wszystkie interfejsy w grupie IPMP ulegną awarii w tym samym czasie. Demon `in.mpathd` nie jest w stanie wykonać przełączenia Failover w takiej sytuacji. Gdy wszystkie systemy docelowe przestaną działać, wtedy również IPMP nie wykona Failover. Wówczas `in.mpathd` czyści swoją listę systemów docelowych i zaczyna ich wykrywanie od nowa.

Wykrywanie fizycznej naprawy interfejsu

Aby demon `in.mpathd` mógł uznać, że interfejs został naprawiony, w jego ustawieniach musi być obecna flaga `RUNNING`. Jeśli stosowane jest wykrywanie awarii na podstawie sondowania, `in.mpathd` musi otrzymywać odpowiedzi na 10 kolejnych pakietów sondujących przez dany interfejs. Gdy interfejs zostanie uznany jako naprawiony, adresy które zostały przeniesione na inny interfejs wracają z powrotem do naprawiony interfejs.

4.2 Przebieg awarii

Rozważamy przykładową sytuację awarii interfejsu, gdy mamy skonfigurowane IP Multipathing. Mamy dwa fizyczne interfejsy: `e1000g0` oraz `e1000g1`. Nasza grupa IPMP nazywa się `mptest`.

```
e1000g0: flags=9000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4>
     mtu 1500 index 2
     inet 192.168.0.100 netmask ffffff00 broadcast 192.168.0.255
     groupname mptest
e1000g0:1: flags=9000843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER>
     mtu 1500
     index 2 inet 192.168.0.110 netmask ffffff00 broadcast 192.168.0.255
e1000g1: flags=9000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
     inet 192.168.0.111 netmask ffffff00 broadcast 192.168.0.255
     groupname mptest
e1000g1:1: flags=9000843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER>
     mtu 1500
     index 2 inet 192.168.0.112 netmask ffffff00 broadcast 192.168.0.255
```

Adres `192.168.0.100` to nasz podstawowy adres używany do przesyłu danych. Adresy `192.168.110`, `192.168.111` i `192.168.112` to adresy testowe służące do sondowania połączeń.

Po awarii interfejsu `e1000g0` sytuacja przedstawia się następująco:

```
e1000g0: flags=19000842<BROADCAST,RUNNING,MULTICAST,IPv4,
      NOFAILOVER,FAILED> mtu 0 index 2
      inet 0.0.0.0 netmask 0
      groupname mptest
e1000g0:1: flags=19040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,
      NOFAILOVER,FAILED> mtu 1500 index 2
```

```
      inet 192.168.0.110 netmask ffffff00 broadcast 192.168.0.255
e1000g1: flags=9000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
      inet 192.168.0.111 netmask ffffff00 broadcast 192.168.0.255
      groupname mptest
e1000g1:1: flags=9000843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,
      NOFAILOVER> mtu 1500
      index 2 inet 192.168.0.112 netmask ffffff00 broadcast 10.0.0.255
e1000g1:2: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 6
      inet 192.168.0.100 netmask ffffff00 broadcast 192.168.0.255
```

Na `e1000g0` mamy flagę `FAILED` i adres `0.0.0.0`. Pojawił się nowy interfejs logiczny `e1000g1:2` z adresem `192.168.0.100` przejętym z uszkodzonego interfejsu `e1000g0`.

4.3 Przykładowa konfiguracja

Przedstawimy tutaj konfigurację wykonaną na Solaris 10, na maszynie wirtualnej wyposażonej w 3 interfejsy `e1000g0`, `e1000g1` i `e1000g2`. Wszystkie karty podpięte zostały do jednej wirtualnej sieci w VirtualBox i należą do jednej grupy IPMP o nazwie `mptest`. Założenia są następujące:

```
Network: 192.168.0.0/24
Hostname: alpha
Domain: internal
Interfaces: e1000g0 e1000g1 e1000g2
Failover group name: mptest
Main address: 192.168.0.100
Test address for e1000g0: 192.168.0.110
Test address for e1000g1: 192.168.0.111
Test address for e1000g2: 192.168.0.112
```

Plik `/etc/hosts` wygląda jak poniżej:

```
127.0.0.1   localhost loghost
192.168.0.100  alpha.internal alpha
192.168.0.110  alpha-e1000g0
192.168.0.111  alpha-e1000g1
192.168.0.112  alpha-e1000g2
```

W pliku `/etc/netmasks` mamy:

```
192.168.0.0        255.255.255.0
```

Pliki odpowiadające za konfigurację poszczególych interfejsów podczas rozruchu systemu wyglądają następująco:

`/etc/hostname.e1000g0`:

```
alpha netmask + broadcast + up group mptest \
addif alpha-e1000g0 netmask + broadcast + deprecated -failover up
```

/etc/hostname.e1000g1:

```
alpha-e1000g1 netmask + broadcast + group mptest \
deprecated -failover standby up
```

/etc/hostname.e1000g2:

```
alpha-e1000g2 netmask + broadcast + group mptest \
deprecated -failover standby up
```

Konfiguracja, która została tutaj przedstawiona, jest w zupełności wystarczająca, aby doszło do Failover w razie awarii. Aby wykonać tę konfigurację bez restartu maszyny należy użyć polecenie `ifconfig` dla każdego interfejsu. Zaczynamy od obejrzenia aktualnego stanu:

```
alpha# ifconfig -a
e1000g0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 192.168.0.100 netmask ffffff00 broadcast 192.168.0.255
        ether 8:0:20:c5:10:15
```

Teraz przypisujemy e1000g0 do grupy Failover i dodajemy do niej alias dla adresu testowego:

```
alpha# ifconfig e1000g0 group mptest
alpha# ifconfig e1000g0 addif 192.168.0.110 netmask 255.255.255.0 \
    broadcast 192.168.0.255 -failover deprecated up
```

Następnie dodajemy e1000g1 i e1000g2:

```
alpha# ifconfig e1000g1 plumb 192.168.0.111 netmask 255.255.255.0 \
    broadcast 192.168.0.255 group mptest deprecated -failover standby up
alpha# ifconfig e1000g2 plumb 192.168.0.112 netmask 255.255.255.0 \
    broadcast 192.168.0.255 group mptest deprecated -failover standby up
```

Finalna konfiguracja prezentuje się następująco

```
alpha# ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 192.168.0.100 netmask ffffff00 broadcast 192.168.0.255
        groupname mptest
        ether 8:0:20:c5:10:15
e1000g0:1: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,
            NOFAILOVER> mtu 1500 index 2
        inet 192.168.0.110 netmask ffffff00 broadcast 192.168.0.255
e1000g1: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,
            NOFAILOVER,STANDBY,INACTIVE> mtu 1500 index 3
        inet 192.168.0.111 netmask ffffff00 broadcast 192.168.0.255
        groupname mptest
        ether 8:0:20:c5:10:16
e1000g2: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,
            NOFAILOVER,STANDBY,INACTIVE> mtu 1500 index 4
        inet 192.168.0.112 netmask ffffff00 broadcast 192.168.0.255
        groupname mptest
        ether 8:0:20:c5:10:17
```

Podsumowanie

Po napisaniu całej pracy oraz doprowadzeniu programu do wersji ostatecznej można wyciągnąć następujące wnioski. Po pierwsze, napisanie skryptów do przełączenia awaryjnego sieci WAN nie było takie proste. Wersja, która została tu przedstawiona nie jest jedynym wariantem, który można zastosować. Możliwe jest ulepszanie kodu. Można na przykład dostosować go do większej liczby dostępnych sieci WAN. Natomiast po awarii u jednego z dostawców usług sieciowych i przełączeniu na dostawcę rezerwowego, można by automatycznie wykrywać naprawę łącza i przełączać na poprzedniego, podstawowego dostawcę. Coś podobnego technicznie, ale dla pojedynczego łącza, zapewnia Repair Discovery i Failback w IP Multipathing. W naszym przypadku przełączamy się między różnymi łączami IP.

Dodatek A

Gramatyka reguł filtra pakietów

```
filter-rule = [ insert ] action in-out [ options ] [ tos ] [ ttl ]
      [ proto ] [ ip ] [ group ] [ tag ] [ pps ] .

insert = "@" decnumber .
action = block | "pass" | log | "count" | auth | call .
in-out = "in" | "out" .
options = [ log ] [ "quick" ] [ onif [ dup ] [ froute ] ] .
tos = "tos" decnumber | "tos" hexnumber .
ttl = "ttl" decnumber .
proto = "proto" protocol .
ip = srcdst [ flags ] [ with withopt ] [ icmp ] [ keep ] .
group = [ "head" decnumber ] [ "group" decnumber ] .
pps = "pps" decnumber .

onif = "on" interface-name [ "out-via" interface-name ] .
block = "block" [ return-icmp[return-code] | "return-rst" ] .
auth = "auth" | "preauth" .
log = "log" [ "body" ] [ "first" ] [ "or-block" ] [ "level" loglevel ] .
tag = "tag" tagid .
call = "call" [ "now" ] function-name "/" decnumber.
dup = "dup-to" interface-name[":"ipaddr] .
froute = "fastroute" | "to" interface-name .
replyto = "reply-to" interface-name [ ":" ipaddr ] .
protocol = "tcp/udp" | "udp" | "tcp" | "icmp" | decnumber .
srcdst = "all" | fromto .
fromto = "from" object "to" object .

return-icmp = "return-icmp" | "return-icmp-as-dest" .
loglevel = facility"."priority | priority .
object = addr [ port-comp | port-range ] .
addr = "any" | nummask | host-name [ "mask" ipaddr | "mask" hexnumber ] .
port-comp = "port" compare port-num .
port-range = "port" port-num range port-num .
flags = "flags" flag { flag } [ "/" flag { flag } ] .
with = "with" | "and" .
icmp = "icmp-type" icmp-type [ "code" decnumber ] .
return-code = "("icmp-code")" .
```

```
keep = "keep" "state" [ "limit" number ] | "keep" "frags" .

nummask = host-name [ "/" decnumber ] .
host-name = ipaddr | hostname | "any" .
ipaddr = host-num "." host-num "." host-num "." host-num .
host-num = digit [ digit [ digit ] ] .
port-num = service-name | decnumber .

withopt = [ "not" | "no" ] opttype [ [ "," ] withopt ] .
opttype = "ipopts" | "short" | "nat" | "bad-src" | "lowttl" | "frag" |
          "mbcast" | "opt" ipopts  .
optname = ipopts [ "," optname ] .
ipopts  = optlist | "sec-class" [ secname ] .
secname = seclvl [ "," secname ] .
seclvl  = "unclass" | "confid" | "reserv-1" | "reserv-2" | "reserv-3" |
  "reserv-4" | "secret" | "topsecret" .
icmp-type = "unreach" | "echo" | "echorep" | "squench" | "redir" |
    "timex" | "paramprob" | "timest" | "timestrep" | "inforeq" |
    "inforep" | "maskreq" | "maskrep"  | "routerad" |
    "routersol" | decnumber .
icmp-code = decumber | "net-unr" | "host-unr" | "proto-unr" | "port-unr" |
    "needfrag" | "srcfail" | "net-unk" | "host-unk" | "isolate" |
    "net-prohib" | "host-prohib" | "net-tos" | "host-tos" |
    "filter-prohib" | "host-preced" | "cutoff-preced" .
optlist = "nop" | "rr" | "zsu" | "mtup" | "mtur" | "encode" | "ts" | "tr" |
  "sec" | "lsrr" | "e-sec" | "cipso" | "satid" | "ssrr" | "addext" |
  "visa" | "imitd" | "eip" | "finn" .
facility = "kern" | "user" | "mail" | "daemon" | "auth" | "syslog" |
   "lpr" | "news" | "uucp" | "cron" | "ftp" | "authpriv" |
   "audit" | "logalert" | "local0" | "local1" | "local2" |
   "local3" | "local4" | "local5" | "local6" | "local7" .
priority = "emerg" | "alert" | "crit" | "err" | "warn" | "notice" |
   "info" | "debug" .

hexnumber = "0" "x" hexstring .
hexstring = hexdigit [ hexstring ] .
decnumber = digit [ decnumber ] .

compare = "=" | "!=" | "<" | ">" | "<=" | ">=" | "eq" | "ne" | "lt" | "gt" |
  "le" | "ge" .
range = "<>" | "><" .
hexdigit = digit | "a" | "b" | "c" | "d" | "e" | "f" .
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" .
flag = "F" | "S" | "R" | "P" | "A" | "U" | "C" | "W" .
```

Dodatek B

Parametry tuningowe

```
ipf_flags           min 0   max 4294967295  current 0
active              min 0   max 0           current 1
control_forwarding  min 0   max 1           current 0
update_ipid         min 0   max 1           current 0
chksrc              min 0   max 1           current 0
min_ttl             min 0   max 1           current 4
icmp_minfragmtu     min 0   max 1           current 68
default_pass        min 0   max 4294967295  current 134217730
tcp_idle_timeout    min 1   max 2147483647  current 864000
tcp_close_wait      min 1   max 2147483647  current 480
tcp_last_ack        min 1   max 2147483647  current 60
tcp_timeout         min 1   max 2147483647  current 480
tcp_syn_sent        min 1   max 2147483647  current 480
tcp_syn_received    min 1   max 2147483647  current 480
tcp_closed          min 1   max 2147483647  current 60
tcp_half_closed     min 1   max 2147483647  current 14400
tcp_time_wait       min 1   max 2147483647  current 480
udp_timeout         min 1   max 2147483647  current 240
udp_ack_timeout     min 1   max 2147483647  current 24
icmp_timeout        min 1   max 2147483647  current 120
icmp_ack_timeout    min 1   max 2147483647  current 12
ip_timeout          min 1   max 2147483647  current 120
intercept_loopback  min 0   max 1           current 0
log_suppress        min 0   max 1           current 1
log_all             min 0   max 1           current 0
log_size            min 0   max 524288      current 32768
state_max           min 1   max 2147483647  current 4013
state_size          min 1   max 2147483647  current 5737
state_lock          min 0   max 1           current 0
state_maxbucket     min 1   max 2147483647  current 26
state_logging       min 0   max 1           current 1
state_wm_high       min 2   max 100         current 99
state_wm_low        min 1   max 99          current 90
state_wm_freq       min 2   max 999999      current 20
nat_lock            min 0   max 1           current 0
nat_table_size      min 1   max 2147483647  current 2047
nat_table_max       min 1   max 2147483647  current 30000
```

```
nat_rules_size      min 1   max 2147483647  current 127
rdr_rules_size      min 1   max 2147483647  current 127
hostmap_size        min 1   max 2147483647  current 2047
nat_maxbucket       min 1   max 2147483647  current 22
nat_logging         min 0   max 1           current 1
nat_doflush         min 0   max 1           current 0
nat_table_wm_low    min 1   max 99          current 90
nat_table_wm_high   min 2   max 100         current 99
proxy_debug         min 0   max 31          current 0
ftp_debug           min 0   max 127         current 0
ftp_pasvonly        min 0   max 1           current 0
ftp_insecure        min 0   max 1           current 0
ftp_pasvrdr         min 0   max 1           current 0
ftp_forcepasv       min 0   max 1           current 1
ftp_single_xfer     min 0   max 1           current 0
tftp_read_only      min 0   max 1           current 1
ftp_debug           min 0   max 127         current 0
ftp_pasvonly        min 0   max 1           current 0
ftp_insecure        min 0   max 1           current 0
ftp_pasvrdr         min 0   max 1           current 0
ftp_forcepasv       min 0   max 1           current 1
ftp_single_xfer     min 0   max 1           current 0
```

Dodatek C

Plik konfiguracyjny
ipf-wan1.conf

```
##############################################################################
#
# LAN-1 : e1000g0 : 192.168.0.1/24 : biuro
#
### Inbound traffic
#
# Spoofing
#
block in quick on e1000g0 from 172.16.0.0/12 to any

##############################################################################
#
# LAN-2 : e1000g1 : 172.16.0.1/24 : serwis
#
### Inbound traffic
#
# Spoofing
#
block in quick on e1000g1 from 192.168.0.0/16 to any
block in quick on e1000g1 from any to 192.168.0.0/16
#
### Outbound traffic
#
# LAN-1 biuro to LAN-2 serwis
#
pass out quick on e1000g1 proto tcp \
    from 192.168.0.0/16 to 172.16.0.0/12 flags S keep state
pass out quick on e1000g1 proto udp \
    from 192.168.0.0/16 to 172.16.0.0/12 keep state
pass out quick on e1000g1 proto icmp \
    from 192.168.0.0/16 to 172.16.0.0/12 keep state

##############################################################################
#
# WAN-1 : e1000g2 : 10.0.1.10/30 : ISP-1
```

```
#
### Inbound traffic
#
# Restrictive policy
#
block in on e1000g2 all
#
# ICMP ping
#
pass in on e1000g2 proto icmp \
    from any to 10.0.1.10/32 icmp-type 8 keep state
#
# SSH
#
block in log quick on e1000g2 proto tcp \
    from any to 10.0.1.10/32 port = 22 flags S keep state
#
# HTTP
#
pass in on e1000g2 proto tcp \
    from any to 10.0.1.10/32 port = 80 flags S keep state
pass in on e1000g2 proto tcp \
    from any to 10.0.1.10/32 port = 443 flags S keep state
#
### Outbound traffic
#
# Locally generated traffic
#
pass out quick on e1000g2 proto tcp all flags S keep state
pass out quick on e1000g2 proto udp all keep state
pass out quick on e1000g2 proto icmp all keep state
pass out quick on e1000g2 proto gre all keep state
#
### NAT traffic
#
pass in on e1000g2 proto tcp \
    from pool/trusted to 192.168.0.100/32 port = 22 flags S keep state
pass in on e1000g2 proto tcp \
    from pool/trusted to 192.168.0.100/32 port = 80 flags S keep state
pass in on e1000g2 proto tcp \
    from pool/trusted to 172.16.0.100/32 port = 3389 flags S keep state


##############################################################################
#
# WAN-2 : e1000g3 : 10.0.2.10/29 : ISP-2
#
### Inbound traffic
#
# Restrictive policy
#
block in on e1000g3 all
```

```
#
# ICMP ping
#
pass in on e1000g3 reply-to e1000g3:10.0.2.9 proto icmp \
    from any to 10.0.2.10/32 icmp-type 8 keep state
#
# SSH
#
block in log quick on e1000g3 proto tcp from any to any port = 22
#
# HTTP
#
pass in on e1000g3 reply-to e1000g3:10.0.2.9 proto tcp \
    from any to 10.0.2.10/32 port = 80 flags S keep state
pass in on e1000g3 reply-to e1000g3:10.0.2.9 proto tcp \
    from any to 10.0.2.10/32 port = 443 flags S keep state
```

Dodatek D

Plik konfiguracyjny
ipf-wan2.conf

```
#########################################################################
#
# LAN-1 : e1000g0 : 192.168.0.1/24 : biuro
#
### Inbound traffic
#
# Spoofing
#
block in quick on e1000g0 from 172.16.0.0/12 to any

#########################################################################
#
# LAN-2 : e1000g1 : 172.16.0.1/24 : serwis
#
### Inbound traffic
#
# Spoofing
#
block in quick on e1000g1 from 192.168.0.0/16 to any
block in quick on e1000g1 from any to 192.168.0.0/16
#
### Outbound traffic
#
# LAN-1 biuro to LAN-2 serwis
#
pass out quick on e1000g1 proto tcp \
    from 192.168.0.0/16 to 172.16.0.0/12 flags S keep state
pass out quick on e1000g1 proto udp \
    from 192.168.0.0/16 to 172.16.0.0/12 keep state
pass out quick on e1000g1 proto icmp \
    from 192.168.0.0/16 to 172.16.0.0/12 keep state

#########################################################################
#
# WAN-1 : e1000g2 : 10.0.1.10/30 : ISP-1
```

```
#
### Inbound traffic
#
# Restrictive policy
#
block in on e1000g2 all
#
# ICMP ping
#
pass in on e1000g2 reply-to e1000g2:10.0.1.9 proto icmp \
    from any to 10.0.2.10/32 icmp-type 8 keep state
#
# SSH
#
block in log quick on e1000g2 proto tcp from any to any port = 22
#
# HTTP
#
pass in on e1000g2 reply-to e1000g2:10.0.1.9 proto tcp \
    from any to 10.0.2.10/32 port = 80 flags S keep state
pass in on e1000g2 reply-to e1000g2:10.0.1.9 proto tcp \
    from any to 10.0.2.10/32 port = 443 flags S keep state

##############################################################################
#
# WAN-2 : e1000g3 : 10.0.2.10/29 : ISP-2
#
### Inbound traffic
#
# Restrictive policy
#
block in on e1000g3 all
#
# ICMP ping
#
pass in on e1000g3 proto icmp \
    from any to 10.0.2.10/32 icmp-type 8 keep state
#
# SSH
#
block in log quick on e1000g3 proto tcp \
    from any to 10.0.2.10/32 port = 22 flags S keep state
#
# HTTP
#
pass in on e1000g3 proto tcp \
    from any to 10.0.2.10/32 port = 80 flags S keep state
pass in on e1000g3 proto tcp \
    from any to 10.0.2.10/32 port = 443 flags S keep state
#
### Outbound traffic
#
```

```
# Locally generated traffic
#
pass out quick on e1000g3 proto tcp all flags S keep state
pass out quick on e1000g3 proto udp all keep state
pass out quick on e1000g3 proto icmp all keep state
pass out quick on e1000g3 proto gre all keep state
#
### NAT traffic
#
pass in on e1000g3 proto tcp \
    from pool/trusted to 192.168.0.100/32 port = 22 flags S keep state
pass in on e1000g3 proto tcp \
    from pool/trusted to 192.168.0.100/32 port = 80 flags S keep state
pass in on e1000g3 proto tcp \
    from pool/trusted to 172.16.0.100/32 port = 3389 flags S keep state
```

Dodatek E

Plik konfiguracyjny
ipnat-wan1.conf

```
##############################################################################
#
# WAN-1 : e1000g2 : 10.0.1.10/30 : ISP-1
#
### Destination NAT

rdr e1000g2 0/0 port 9022 -> 192.168.0.100 port 22 tcp
rdr e1000g2 0/0 port 9080 -> 192.168.0.100 port 80 tcp
rdr e1000g2 0/0 port 9033 -> 172.16.0.100 port 3389 tcp

### Source NAT

map e1000g2 192.168.0.0/24 -> 0/32 proxy port ftp ftp/tcp
map e1000g2 192.168.0.0/24 -> 0/32 portmap tcp/udp 10000:30000
map e1000g2 192.168.0.0/24 -> 0/32

map e1000g2 172.16.0.0/24 -> 0/32 proxy port ftp ftp/tcp
map e1000g2 172.16.0.0/24 -> 0/32 portmap tcp/udp 10000:30000
map e1000g2 172.16.0.0/24 -> 0/32
```

Dodatek F

Plik konfiguracyjny
ipnat-wan2.conf

```
##############################################################################
#
# WAN-2 : e1000g3 : 10.0.2.10/29 : ISP-2
#
### Destination NAT

rdr e1000g3 0/0 port 9022 -> 192.168.0.100 port 22 tcp
rdr e1000g3 0/0 port 9080 -> 192.168.0.100 port 80 tcp
rdr e1000g3 0/0 port 9033 -> 172.16.0.100 port 3389 tcp

### Source NAT

map e1000g3 192.168.0.0/24 -> 0/32 proxy port ftp ftp/tcp
map e1000g3 192.168.0.0/24 -> 0/32 portmap tcp/udp 10000:30000
map e1000g3 192.168.0.0/24 -> 0/32

map e1000g3 172.16.0.0/24 -> 0/32 proxy port ftp ftp/tcp
map e1000g3 172.16.0.0/24 -> 0/32 portmap tcp/udp 10000:30000
map e1000g3 172.16.0.0/24 -> 0/32
```

Spis rysunków

1.1 Diagram przepływu pakietów przez jądro systemu i IPFilter. . 6

2.1 Przerwy w pracy systemu w podziale na okres czasu [12]. . . . 21
2.2 Diagram sieci do testowania Failover i IPFilter. 25

3.1 Typowe zastosowanie pojedynczego Load Balancera [14]. . . . 31
3.2 Zaawansowane zastosowanie Load Balancera [14]. 31

Bibliografia

[1] Kevin Fall, Richard Stevens, *TCP/IP Illustrated*, Volume 1 The Protocols, Pearson Education Inc., 2012.

[2] Mateusz Maciejczuk, *Policy routing oraz VPN na firewallu opartym o IPFilter*, Praca licencjacka, Instytut Informatyki, Uniwersytet w Białymstoku, 2020.

[3] Darren Reed, Dokumentacja IPFilter, wersja 1.5.2, 2012.

[4] Dokumentacja Sun Solaris 10, `in.mpathd`, 2006.

[5] Dokumentacja Oracle Solaris Administration, IP Services, `https://docs.oracle.com/cd/E26505_01/html/E27061/ipmptm-1.html`
Dostęp na dzień 2020-07-12.

[6] Czym jest SLA? `https://kotrak.pl/blog/service-level-agreement-sla-co-to-jest/#czym-jest-service-level-agreement`
Dostęp na dzień 2020-06-28.

[7] Czym jest wskaźnik MTTR? `https://qrmaint.pl/blog/czym-jest-wskaznik-mttr/`
Dostęp na dzień 2020-06-28.

[8] Czym jest wysoka dostepność? `http://ideas2action.pl/2012/10/09/czym-jest-wysoka-dostepnosc/`
Dostęp na dzień 2020-06-30.

[9] Disaster Recovery vs High Availability vs Fault Tolerance what are the differences? `https://blog.interfaceware.com/disaster-recovery-vs-high-availability-vs-fault-tolerance-what-are-the-differences/`
Dostęp na dzień 2020-07-06.

[10] Failover,
https://www.barracuda.com/glossary/failover
Dostęp na dzień 2020-07-10.

[11] Fault Tolerance vs High Availability,
https://www.vxchnge.com/blog/fault-tolerance-vs-high-availability/
Dostęp na dzień 2020-07-01.

[12] High Availability,
https://en.wikipedia.org/wiki/High_availability
Dostęp na dzień 2020-06-29.

[13] High Availability, Fault Tolerance, Disaster Recovery,
https://www.greenhousedata.com/blog/high-availability-vs-fault-tolerance-vs-disaster-recovery
Dostęp na dzień 2020-07-03.

[14] Load Balancing - Grokking the System Design,
https://www.educative.io/courses/grokking-the-system-design-interview/3jEwl04BL7Q
Dostęp na dzień 2020-07-12.

[15] What is Load Balancing?
https://avinetworks.com/what-is-load-balancing/
Dostęp na dzień 2020-07-03.