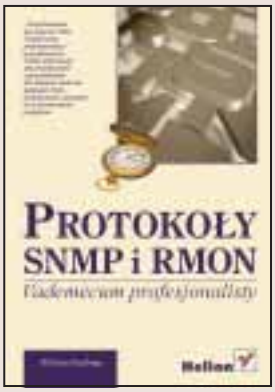
Protokoły SNMP i RMON. Vademecum profesjonalisty



Autor: William Stallings
Tłumaczenie: Mateusz Michalski
ISBN: 83-7197-920-7
Tytuł oryginału: SNMP, SNMPv2, SNMPv3, and RMON 1 and 2
Format: B5, stron: 604

SNMP (Simple Network Management Protocol) wraz z RMON (Remote Network Monitoring) to najefektywniejsze narzędzia do zarządzania współczesnymi, bardzo zróżnicowanymi systemami sieciowymi, co powoduje postrzeganie ich jako standard w zakresie zarządzania sieciami.

„Protokoły SNMP i RMON. Vademecum profesjonalisty" to doskonały podręcznik skierowany do administratorów, menadżerów i projektantów sieci komputerowych, opisujący zagadnienia zarządzania sieciami w oparciu o SNMP. Napisana zwięźle i konkretnie, skupiająca się na zagadnieniach praktycznych książka, opisuje SNMPv1, SNMPv2 oraz najnowszą wersję SNMPv3, a także RMON1 i RMON2 – czyli wszystko to, czego używa się obecnie w sieciach LAN i WAN. Dzięki książce będziesz mógł lepiej określić swoje wymagania co do systemu zarządzania siecią, poznać przesłanki, którymi kierowali się projektanci oraz zdobędziesz niezbędną wiedzę do efektywnego wykorzystania dostępnych produktów wspierających SNMP.

W książce autor zawarł pomocne informacje wprowadzające w tematykę zarządzania sieciami, w tym przegląd wymagań stawianych systemom zarządzania. Znajdziesz w niej wyjaśnienia zagadnień podstawowych, takich jak architektura zarządzania siecią, monitoring wydajności, poprawności działania i wykorzystania zasobów sieciowych oraz kontrola konfiguracji i bezpieczeństwa. Nie zabrakło szczegółowych informacji na temat działania protokołu SNMPv1 oraz jego rozszerzeń wprowadzonych w wersji 2. i 3., ze szczególnym uwzględnieniem mechanizmów bezpieczeństwa – uwierzytelnianiu, szyfrowaniu, modelu bezpieczeństwa USM (User-based Security Model) i modelu kontroli dostępu VACM (View-based Access Control Model).

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 13.

SNMPv2 — bazy MIB i zgodność

Rozpoczniemy ten rozdział opisem bazy MIB dla SNMPv2, która wykorzystywana jest zarówno w SNMPv2, jak i w SNMPv1. Następnie rozpatrzone zostaną wyrażenia zgodności; używane są one do określania wymagań dotyczących zgodności dla ujednoliconych baz MIB i pozwalają producentom określić zakres ich implementacji. Dalej przyjrzymy się rozszerzeniom MIB związanym z grupą `interfaces`, które definiowane są w oparciu o SMI protokołu SNMPv2 i wykorzystują pewne cechy tego protokołu.

13.1. Baza informacji zarządzania w SNMPv2

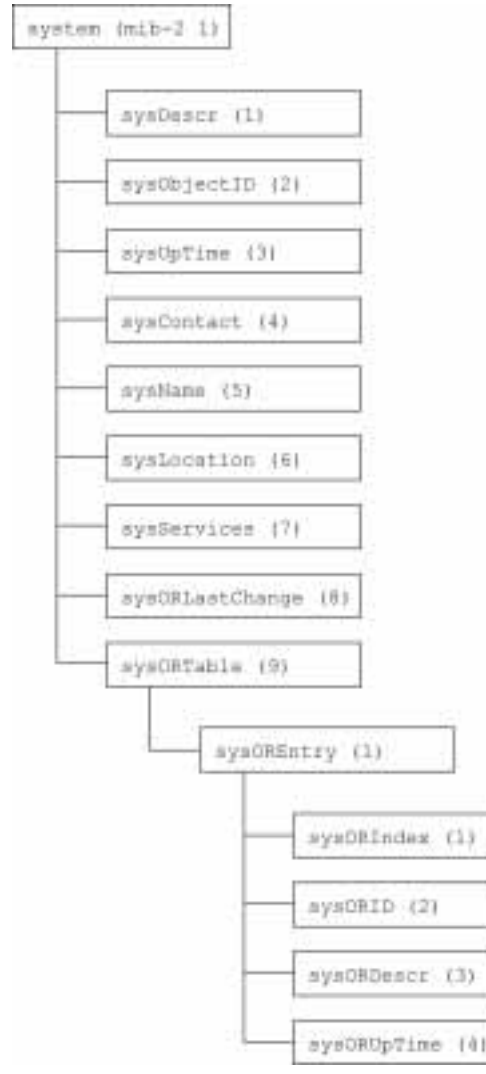
SNMPv2 MIB definiuje obiekty, które opisują zachowanie jednostek SNMPv2. Takie bazy MIB składają się z trzech grup:

- grupy system — rozwinięcie oryginalnej grupy system z bazy MIB-II po to, by zawierała zestaw obiektów pozwalających na pełnienie przez jednostkę SNMPv2 roli agenta przy określaniu swych zasobów dynamicznie konfigurowalnych obiektów,
- grupy SNMP — usprawnienie pierwotnej grupy snmp z bazy MIB-II, składające się z obiektów dostarczających podstawowego oprzyrządowania do działania protokołu,
- grupy MIB objects — zestaw obiektów zajmujących się jednostkami PDU typu SNMPv2-Trap i pozwalających współpracującym jednostkom SNMPv2, wszystkim występującym w roli zarządców, na skoordynowane wykorzystanie operacji `set` protokołu SNMPv2.

Rozpatrzymy kolejno każdą grupę MIB.

13.1.1. Grupa system

Grupa `system` zdefiniowana w SNMPv2 MIB jest faktycznie tą samą grupą, co zdefiniowana w MIB-II, z dodatkiem paru nowych obiektów. Rysunek 13.1 prezentuje skorygowaną grupę `system`, która ciągle pozostaje jednak częścią hierarchii MIB-II.



Rysunek 13.1. ***Skorygowana grupa system***

Porównanie rysunku 13.1 z oryginalną grupą `system` (rysunek 6.1) pokazuje, że wszystkie nowe obiekty mają nazwy zaczynające się prefiksem `sysOR`. Obiekty te mają związek z zasobami systemowymi i używane są przez jednostkę SNMPv2 pełniącą rolę agenta do opisu kontrolowanych przez nią zasobów obiektowych; mogą być dynamicznie konfigurowane przez zarządcę. Tabela 13.1 grupuje wspomniane obiekty[1]. Jak widać, dochodzi jedna wielkość skalarna i pojedyncza tablica obiektów-zasobów. Wielkość skalarna to `snmpORLastChange`, która rejestruje wartość `sysUpTime` w momencie ostatniej zmiany stanu bądź wartości któregokolwiek z obiektów zawartych w tablicy obiektów-zasobów; innymi słowy, jest to czas ostatniej zmiany w zestawie dających się kontrolować zasobów, sterowanych przez tego zarządcę. Tablica obiektów-zasobów ma tryb *RO* (*Tylko-do-odczytuy*) i składa się z pojedynczego wiersza dla każdego zasobu obiektowego konfigurowalnego dynamicznie.

[1] **Użyto następujących oznaczeń**: `read-only` (*tylko-do-odczytu*) — *RO*, `read-write` (*do zapisu i odczytu*) — *RW*, `read-create` (*do odczytu i tworzenia*) — *RC*, `not-accessible` (*niedostępne*) — *NA*.

Tabela 13.1. *Uzupełnienie SNMPv2 grupy system*

Obiekt	Składnia	Opis
sysORLastChange	TimeStamp	Wartość sysUpTime z chwili ostatniej zmiany stanu bądź wartości jakiejkolwiek instancji sysORID
sysORTable	SEQUENCE OF sysOREntry	Tablica dynamicznie konfigurowalnych zasobów obiektów w jednostce SNMPv2 pełniącej rolę agenta
sysOREntry	SEQUENCE	Informacja dotycząca poszczególnych dynamicznie konfigurowalnych zasobów obiektów
sysORIndex	INTEGER	Liczba całkowita stanowiąca indeks w tablicy sysORTable
sysORID	OJECT IDENTIFIER	Identyfikator obiektu (ID) danego wpisu. Jest to odpowiednik obiektu sysObjectID z MIB-II
sysORDescr	DisplayString	Tekstowy opis danego zasobu obiektu. Jest to odpowiednik obiektu sysDescr z MIB-II
sysORUpTime	TimeStamp	Wartość sysUpTime w momencie ostatniej modyfikacji wartości tego wiersza

13.1.2. Grupa SNMP

Jest to ta sama grupa, którą zdefiniowano w MIB-II, lecz zawierająca pewne nowe obiekty i pozbawiona zarazem części oryginalnych obiektów. Grupa snmp przechowuje pewne elementarne informacje dotyczące ruchu pakietów, odnoszące się do działania SNMPv2. Wszystkie obiekty, oprócz jednego, są 32-bitowymi licznikami z trybem *RO* (*Tylko-do-odczytu*) — zgrupowano je w tabeli 13.2. Wspomniany wyjątek dotyczy obiektu snmpEnableAuthenTraps, mającego tryb *RW* (*Do-zapisu-i-odczytu*), typu wyliczeniowego całkowitego, przyjmującego wartości enabled(1) i disabled(2), wskazujące, czy jednostka SNMPv2 jest uprawniona do generowania pułapek authenticationFailure.

Porównanie do pierwotnej grupy snmp MIB-II (rysunek 7.5) pokazuje, że analizowana grupa (rysunek 13.2) zawiera zdecydowanie mniej parametrów. Wynika to stąd, że tak szczegółowe dane nie są niezbędne do rozwiązywania faktycznych problemów, a poza tym znacząco zwiększają rozmiar agenta. W związku z tym zaadaptowano bardziej wydajną grupę obiektów.

13.1.3. Grupa MIBObjects

Grupa MIB Objects zawiera dodatkowe obiekty odnoszące się do sterowania obiektami bazy MIB (rysunek 13.3). Pierwsza część tego zestawu jest podgrupą snmpTrap, złożoną z dwóch obiektów związanych z pułapkami:

- snmpTrapOID, który jest identyfikatorem aktualnie wysyłanego obiektu pułapki lub powiadomienia. Wartość tego obiektu występuje jako druga varbind w każdej jednostce PDU typu SNMPv2-Trap i InformRequest.

Tabela 13.2. *Liczniki w uzupełnionej grupie SNMP*

snmpInPackets	Liczba wszystkich pakietów odebranych przez jednostkę SNMPv2 z usługi transportowej
snmpInBadVersions	Liczba wszystkich komunikatów SNMP dostarczonych do jednostki SNMP, przeznaczonych dla nieobsługiwanej wersji SNMP
snmpInBadCommunityNames	Liczba wszystkich komunikatów SNMP dostarczonych do jednostki SNMP, używających nazwy społeczności nieznanej jednostce
snmpInBadCommunityUses	Liczba wszystkich komunikatów SNMP dostarczonych do jednostki SNMP, reprezentujących operacje SNMP nie akceptowane przez społeczność określoną w komunikacie
snmpInASNParseErrs	Liczba wszystkich błędów ASN.1 lub BER, które wystąpiły w trakcie dekodowania otrzymanych komunikatów SNMP
snmpSilentDrops	Liczba wszystkich jednostek PDU typu GetRequest, GetNextRequest, GetBulkRequest, SetRequest i InformRequest, które zostały pominięte, ponieważ rozmiar wiadomości zwrotnej, stanowiącej jednostkę PDU z alternatywną odpowiedzią zawierającą puste pole powiązań zmiennych, był większy albo od ograniczeń lokalnych, albo maksymalnego dopuszczalnego rozmiaru wiadomości w jednostce wysyłającej żądanie
snmpProxyDrops	Liczba wszystkich jednostek PDU typu GetRequest, GetNextRequest, GetBulkRequest, SetRequest i InformRequest, które zostały pominięte, ponieważ kontekst wskazywał na agenta proxy, a transmisja wiadomości (prawdopodobnie przetłumaczonej) nie powiodła się w taki sposób (inny niż przekroczenie dopuszczalnego czasu oczekiwania na odpowiedź), że do jednostki wysyłającej żądanie nie mogła być wysłana żadna jednostka PDU z odpowiedzią

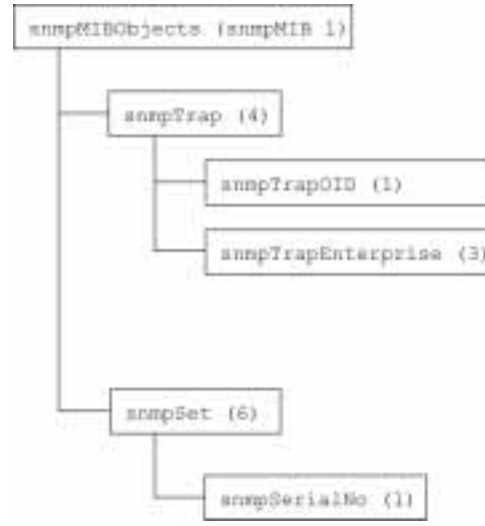
- snmpTrapEnterprise, który jest identyfikatorem obiektu przedsiębiorstwa związanego z aktualnie wysyłaną pułapką. Gdy agent proxy odwzorowuje jednostkę PDU typu Trap (zdefiniowaną w RFC 1157) na jednostkę PDU typu SNMPv2-Trap, wartość tej zmiennej występuje jako ostatnia varbind.

Drugą część zestawu obiektów z tej grupy stanowi podgrupa snmpSet złożona z pojedynczego obiektu snmpSerialNo. Obiekt ten służy do rozwiązania dwóch problemów mogących pojawić się przy korzystaniu z operacji set:

1. Na tym samym obiekcie MIB zarządca może dokonywać wielu operacji typu set i może być ważne, aby wszystkie one wykonane były w porządku ich wysyłania, nawet jeśli w trakcie transmisji ten porządek został zaburzony.
2. Jednoczesne użycie operacji set przez wielu zarządców może skutkować niespójnością bądź błędami w bazie danych.

Dla wyjaśnienia drugiego punktu rozważmy prosty przykład. Przypuśćmy, że wartość obiektu MIB odpowiada adresowi miejsca w buforze, który używany jest do gromadzenia danych pobranych od zarządcy przy użyciu pewnego protokołu transferu plików. Wartość

Rysunek 13.2. *Uzupełniona grupa SNMP*



Rysunek 13.3. *Grupa snmpMIBObjects*

obiektu wskazuje następne dostępne miejsce. Zarządca wykorzystuje tę wartość w następujący sposób: najpierw odczytuje tę wartość, następnie zwiększa ją tak, by wskazywała na następne miejsce, i ostatecznie przesyła odpowiednie dane. Może jednak przy tym zajść następująca sekwencja zdarzeń:

1. Menedżer A pobiera wartość obiektu, która wynosi, załóżmy, *x*.
2. Menedżer B pobiera tę samą wartość.
3. Menedżer A potrzebuje *y* oktetów przestrzeni buforu i dlatego wydaje agentowi polecenie, by zmodyfikował wartość obiektu do *x*+*y*.

4. Menedżer B potrzebuje z oktetów przestrzeni buforu i dlatego wydaje agentowi polecenie, by zmodyfikował wartość obiektu do $x+z$.
5. Oba menedżery (A i B) są przygotowane do wysłania danych do buforu, poczynając od lokacji wyznaczonej przez wartość x.

W rezultacie albo A nadpisze dane B, albo na odwrót. Co więcej, jeśli $z < y$ i A wyśle swoje dane po B, to nie tylko nadpisze dane B, ale jeszcze część danych A zostanie nadpisana przez następnego nadzorcę używającego tego buforu.

Ten problem, w którym rezultat zależny jest od kolejności zachodzenia niezależnych zdarzeń, nazywany jest *sytuacją wyścigu* (*Race*)[2].

Jedyny obiekt w grupie `snmpSet` jest definiowany następująco:

```
snmpSetSerialNo    OBJECT TYPE
   SYNTAX       TestAndIncr
   MAX-ACCESS   read-write
   STATUS       current
   DESCRIPTION
      "Blokada nadzorcza, umożliwiająca współpracującym jednostkom SNMPv2,
      występującym w roli zarządców, skoordynowane wykorzystanie operacji set
      protokołu SNMPv2. Obiekt ten używany jest do koordynacji zgrubnej. Aby otrzymać
      dokładną koordynację, można, w zależności od potrzeb, zdefiniować dodatkowo
      jeden lub więcej podobnych obiektów w każdej grupie MIB"
   ::= { snmpSet 1 }
```

Obiekt `TestAndIncr` jest konwencją tekstową i jest typu całkowitego (`INTEGER`: 0..2 147 483 647). Jej zakres mieści się w przedziale od 0 do $2^{31}-1$. Zasady modyfikacji tego obiektu są następujące. Załóżmy, że aktualna wartość obiektu wynosi K. W tej sytuacji:

1. Jeśli agent otrzyma polecenie `set` dla tego obiektu z wartością K, wartość obiektu jest zwiększana do $(K+1)$ modulo 2^{31}, polecenie wykonywane jest prawidłowo i odsyłana jest wartość K.
2. Jeśli agent otrzyma polecenie `set` dla tego obiektu z wartością różną od K, operacja kończy się niepowodzeniem, a zwrócona zostanie informacja o błędzie typu `inconsistentValue` (*sprzeczna wartość*).

Definicja konwencji tekstowej `TestAndIncr` zamieszczona jest w dodatku 13A.

Wiadomo, że polecenie `set` wykonywane jest jako niepodzielna instrukcja atomowa; oznacza to, że po odebraniu jednostki PDU typu `SetRequest` przeprowadzane są wszystkie operacje `set` dla zmiennych zawartych w polu powiązań, jeśli wszystkie one są poprawne i dopuszczalne, lub nie przeprowadzana jest żadna, jeśli choć jedna z nich nie jest poprawna. Tak więc obiekt `snmpSet` może być używany w następujący sposób: gdy zarządca żąda ustawienia jednej lub kilku wartości obiektów agenta, najpierw pobiera wartość obiektu `snmpSet`. Następnie wysyła jednostkę PDU `SetRequest`, której lista powiązań zmiennych zawiera obiekt `snmpSet` z pobraną wartością oraz odpowiednią parę wartości dla każdego ustawianego obiektu. Jeśli dwóch lub więcej zarządców wysyła `SetRequest`, używając tej samej wartości `snmpSet`, pierwszy, który dotrze do agenta, zakończy się powodzeniem (zakładając, że nie wystąpią żadne inne problemy), co w rezultacie spowoduje zwiększenie

[2] W [Stallings (1995b)] znaleźć można szerokie omówienie problemów rozproszonego, jednoczesnego dostępu.

wartości obiektu snmpSet; pozostałe operacje set zakończą się niepowodzeniem z racji niewłaściwej wartości snmpSet. Poza tym jeśli zarządca zażąda przeprowadzenia ustawienia serii obiektów i jednocześnie wymaga gwarancji, że zostaną one wykonane we właściwym porządku, każda operacja zawierać powinna obiekt snmpSet.

Zgodnie z definicją, jest to zgrubna technika koordynacji; jeśli wszyscy zarządcy używają obiektu snmpSet, tylko jeden z nich w danym momencie może prawidłowo wysyłać do agenta żądanie, dotyczące wszystkich obiektów zawartych w MIB. Jeżeli obiekt TestAndIncr jest związany z pojedynczą grupą, wtedy ograniczenia jednoczesnego dostępu dotyczyć będą obiektów tej grupy.

13.2. Wyrażenia zgodności

Specyfikacja SNMPv2 zawiera dokumenty dotyczące zgodności. Ich celem jest definicja notacji pozwalającej określić minimalne wymagania dotyczące implementacji, a także rzeczywisty poziom osiągniętej implementacji.

W dokumencie poruszającym zagadnienia zgodności zdefiniowano cztery makra:

- OBJECT-GROUP — wskazuje te obiekty w MIB, które są częścią grupy zgodności,
- NOTIFICATION-GROUP — identyfikuje zbiór powiadomień,
- MODULE-COMPLIANCE — ustala wymagania w stosunku do agenta względem implementacji modułów i obiektów bazy MIB,
- AGENT-CAPABILITIES — definiuje możliwości poszczególnych implementacji agenta.

13.2.1. Makro OBJECT-GROUP

Makro to używane jest do specyfikacji grup spokrewnionych obiektów zarządzanych. Tak jak w SNMP SMI, grupa zarządzanych obiektów w SNMPv2 jest podstawową jednostką zgodności. Makro OBJECT-GROUP zapewnia producentowi systematyczny sposób opisu stopnia zgodności przez wskazanie, które grupy zostały zaimplementowane.

Specyfikacja SNMPv2 wyjaśnia występującą w SNMP niejednoznaczność, o której wspomniano w podrozdziale 7.5. Ze specyfikacji SNMPv2 wynika, że obiekt jest „zaimplementowany" jedynie wówczas, gdy przy operacji odczytu można otrzymać jakąś sensowną wartość. Poza tym dla obiektów, których wartość można zmieniać, implementacja musi być w stanie wpływać na stosowną zarządzaną jednostkę w odpowiedzi na operację set. Jeżeli agent nie może zaimplementować obiektu, musi zwrócić komunikat o błędzie (taki jak np. noSuchObject) w odpowiedzi na operację protokołu. Niedozwolone jest, aby agent zwracał wartość obiektu, którego nie zaimplementował.

Listing 13.1 prezentuje makro OBJECT-GROUP, które składa się z następujących głównych klauzul:

- klauzula OBJECTS — wykaz wszystkich obiektów w grupie, których klauzula MAX-ACCESS przyjmuje jedną z następujących wartości: accessible-for-notify, read-only, read-write lub read-create (wynika z tego, że obiekty mające klauzulę MAX-ACCESS o wartości not-accessible nie należą do makra OBJECT-GROUP;

Listing 13.1. *Makro OBJECT-GROUP*

```
OBJECT-GROUP MACRO ::= BEGIN
TYPE NOTATION ::= ObjectsPart
                  "STATUS" Status
                  "DESCRIPTION" Text
                  ReferPart
VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
ObjectsPart ::= "OBJECT" "{" Objects "}"
Objects ::= Object | Objects "," Object
Object ::= value (Name ObjectName)
Status ::= "current" | "deprecated" | "obsolete"
ReferPart ::= "REFERENCE" Text | Empty
Text ::= """ string """
END
```

do obiektów takich zaliczają się tablice pojęciowe, wiersze pojęciowe i obiekty będące indeksami wierszy. Każdy z wymienionych w tej klauzuli obiektów musi być zdefiniowany za pomocą makra OBJECT-TYPE w tym samym module, w którym występuje moduł OBJECT-GROUP),

- klauzula STATUS — wskazuje, czy dana definicja jest aktualna, czy przestarzała,
- klauzula DESCRIPTION — zawiera tekstową definicję grupy razem z opisem wszelkich relacji z innymi grupami (wartość podawana przy wywoływaniu makra OBJECT-GROUP jest identyfikatorem obiektu przypisanym do grupy),
- klauzula REFERENCE — może zawierać tekstowy odsyłacz do grupy zdefiniowanej w innym module informacji.

Prostym przykładem definicji z wykorzystaniem makra OBJECT-GROUP jest definicja grupy snmp:

```
snmpGroup OBJECT-GROUP
OBJECTS { snmpInPkts,
          snmpInBadVersions,
          snmpInASNParseErrs,
          snmpBadOperations,
          snmpSilentDrops,
          snmpProxyDrops,
          snmpEnableAuthenTraps, }
STATUS current
DESCRIPTION
"Zbiór obiektów umożliwiających podstawową obsługę i kontrolę jednostek SNMPv2."
::= { snmpMIBGroups 8 }
```

13.2.2. Makro NOTIFICATION-GROUP

Makro NOTIFICATION-GROUP jest używane do definiowania zestawu powiadomień dla potrzeb zgodności. Listing 13.2 przedstawia to makro, składające się z następujących głównych klauzul:

- klauzula NOTIFICATIONS — wykaz wszystkich notyfikacji należących do danej grupy zgodności (każdy z wyszczególnionych obiektów musi być zdefiniowany za pomocą makra NOTIFICATION-TYPE w tym samym module, w którym występuje moduł NOTIFICATION-GROUP),

Listing 13.2. *Makro NOTIFICATION-GROUP*

```
NOTIFICATION-GROUP MACRO ::= BEGIN
TYPE NOTATION ::= NotificationsPart
                  "STATUS" Status
                  "DESCRIPTION" Text
                  ReferPart
VALUE NOTATION ::= value(VALUE OBJECT IDENTIFIER)
NotificationsPart ::= "NOTIFICATIONS" "{" Notifications "}"
Notifications ::= Notification | Notifications "," Notification
Notification ::= value(Name NotificationName)
Status ::= "current" | "deprecated" | "obsolete"
ReferPart ::= "REFERENCE" Text | empty
Text ::= """ string """
END
```

- klauzula STATUS — wskazuje, czy dana definicja jest aktualna, czy przestarzała,
- klauzula DESCRIPTION — zawiera tekstową definicję grupy razem z opisem wszelkich relacji z innymi grupami (wartość podawana przy wywoływaniu makra NOTIFICATION-GROUP jest identyfikatorem obiektu przypisanym do grupy),
- klauzula REFERENCE — może zawierać tekstowy odsyłacz do grupy definiowanej w innym module informacji.

Prostym przykładem definicji NOTIFICATION-GROUP jest definicja powiadomień z bazy SNMPv2 MIB:

```
snmpBasicNotificationsGroup NOTIFICATION-GROUP
   NOTIFICATIONS { coldStart, authenticationFailure }
   STATUS current
   DESCRIPTION
      "Dwie notyfikacje, które jednostka SNMPv2 musi implementować."
   ::= { snmpMIBGroups 7 }
```

13.2.3. Makro MODULE-COMPLIANCE

Makro MODULE-COMPLIANCE określa minimalny zestaw wymagań w odniesieniu do implementacji jednego bądź wielu modułów MIB. Makro to przedstawiono na listingu 13.3. Znaczenie klauzul STATUS, DESCRIPTION i REFERENCE jest analogiczne do tych samych w makrach OBJECTS-GROUP i NOTIFICATION-GROUP.

Listing 13.3. *Makro MODULE-COMPLIANCE*

```
MODULE-COMPLIANCE MACRO ::= BEGIN
TYPE NOTATION ::= "STATUS" Status
                  "DESCRIPTION" Text
                  ReferPart
                  ModulePart
VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
Status ::= "current" | "deprecated" | "obsolete"
ReferPart ::= "REFERENCE" Text | Empty
ModulePart ::= Modules | empty
Modules ::= Module | Modules Module
Module ::= "MODULE" ModuleName          --nazwa modułu
```

```
            MandatoryPart
            CompliancePart
ModuleName ::= modulereference ModuleIdentifier | empty
ModuleIdentifier ::= value (moduleID OBJECT IDENTIFIER) | empty
MandatoryPart ::= "MANDATORY-GROUPS" "{" Groups "}" | empty
Groups ::= Group | Groups "," Group
Group ::= value (group OBJECT IDENTIFIER)
CompliancePart ::= Compliances | empty
Compliances ::= Compliance | Compliances Compliance
Compliance ::= ComplianceGroup | Object
ComplianceGroup ::= "GROUP" value (Name OBJECT IDENTIFIER)
                    "DESCRIPTION" Text
Object ::= "OBJECT" value (Name ObjectName)
            SyntaxPart
            WriteSyntaxPart
            AccessPart
            "DESCRIPTION" Text
--musi być dopasowanie do klauzuli SYNTAX obiektu
SyntaxPart ::= "SYNTAX" type (SYNTAX) | empty
--musi być dopasowanie do klauzuli SYNTAX obiektu
WriteSyntaxPart ::= "WRITE-SYNTAX" type (WriteSYNTAX) | empty
AccessPart ::= "MIN-ACCESS" Access | empty
Access ::= "not-accessible" | "accessible-for-notify" | "read-only" | "read-write" |
"read-create"
Text ::= """ string """
END
```

Klauzula MODULE używana jest raz lub więcej razy, tak aby wymienić każdy moduł objęty wymogiem implementacji. Klauzula, która odnosi się do tego modułu, nie musi zawierać jego nazwy. Inne klauzule MODULE identyfikowane są poprzez nazwę modułu i, opcjonalnie, identyfikator obiektu.

Każda sekcja MODULE określa te grupy, które są obowiązkowe i te, które są opcjonalne dla danej implementacji. Jeżeli występuje chociaż jedna grupa obligatoryjna, wówczas dołączana jest klauzula MANDATORY-GROUPS, która zawiera wykaz wszystkich grup obowiązkowych dla danego modułu. Aby zachować zgodność z danym modułem, implementacja musi obejmować wszystkie grupy obowiązkowe.

Dla każdej grupy, która jest warunkowo obowiązująca lub bezwarunkowo opcjonalna, przewidziano oddzielną klauzulę o nazwie GROUP. Klauzula DESCRIPTION wykorzystywana jest do specyfikacji okoliczności, przy których dana grupa warunkowo obowiązuje (np. jeżeli zaimplementowany jest konkretny protokół bądź jeśli implementowana jest inna grupa).

Wykorzystując klauzulę OBJECT, można określić uściślone wymagania w stosunku do obiektów należących do jednej z wyspecyfikowanych grup. Dla każdego takiego obiektu zamieszcza się oddzielną klauzulę OBJECT. Możliwe są trzy rodzaje dopasowań. Pierwsze dwa stosują się do składni danego obiektu, którego wartość można odczytywać bądź zapisywać. Dopuszczalne są następujące uściślenia:

- zakresu — dla typów INTEGER i Gauge32 zakres dopuszczalnych wartości może być dostosowany przez zwiększenie dolnych ograniczeń, redukcję górnych ograniczeń i (lub) zmniejszenie ilości alternatywnych wyborów wartości i zakresu,

- wyliczeń — dla typów INTEGER i BIT STRING wyliczenie poszczególnych wartości może być dopasowane przez odrzucenie jednej lub więcej wartości,
- rozmiaru — dla typów OCTET STRING rozmiar wartości, wyrażony w znakach, może być uściślony przez podniesienie dolnego ograniczenia, redukcję górnego ograniczenia i (lub) zmniejszenie ilości alternatywnych wyborów wartości i zakresu,
- zestawu — dla typów OCTET STRING zestaw dozwolonych znaków w wartości może być ograniczony przez wprowadzenie kolejnych podtypów (patrz dodatek B.1 — omówienie podtypów).

Wymienione dopasowania definiowane są w klauzuli SYNTAX dla obiektów *tylko do odczytu* i w klauzuli WRITE-SYNTAX dla obiektów, których wartość można nastawiać.

Trzecia grupa uściśleń dotyczy kategorii dostępu do obiektu. W celu zdefiniowania minimalnego poziomu dostępu używana jest klauzula MIN-ACCESS. Implementacja jest zgodna, jeśli poziom dostępu przez nią zapewniany jest większy lub równy określonemu w ten sposób poziomowi minimalnemu lub też mniejszy lub równy poziomowi maksymalnemu specyfikowanemu w klauzuli MAX-ACCESS definicji obiektu.

Wartość podawana przy wywoływaniu makra MODULE-COMPLIANCE jest identyfikatorem obiektu przypisanym do danej definicji zgodności. Przykładowe wyrażenie zgodności dla bazy SNMPv2 MIB wygląda następująco:

```
snmpBasicCompliance MODULE-COMPLIANCE
   STATUS current
   DESCRIPTION
      "Wyrażenie zgodności dla jednostek SNMPv2 implementujących SNMPv2 MIB."
   MODULE
      MANDATORY-GROUPS { snmpGroup, snmpSetGroup, systemGroup,
                         snmpBasicNotificationsGroup }
   GROUP snmpCommunityGroup
   DESCRIPTION
      "Grupa ta jest obligatoryjna dla jednostek SNMPv2, które obsługują
      uwierzytelnianie oparte na społecznościach.
::= { snmpMIBCompliances 2 }
```

Powyższy moduł określa, że agent będzie zgodny, jeśli zaimplementuje wszystkie grupy wymienione w klauzuli MANDATORY-GROUPS i zapewni wsparcie dla mechanizmów uwierzytelniania opartych na społecznościach (zdefiniowanych w SNMPv1).

13.2.4. Definicje możliwości

Makro AGENT-CAPABILITIES jest używane do dokumentowania możliwości jednostki protokołu SNMPv2 pełniącej rolę agenta. Makro to wykorzystywane jest do opisu dokładnego poziomu wsparcia, które zapewnia agent w odniesieniu do wybranej grupy MIB. Definicja taka może określać, że niektóre obiekty mają ograniczoną lub rozszerzoną składnię czy poziom dostępu. Ściśle mówiąc, tego typu określenia możliwości specyfikują dopasowania lub odmiany w odniesieniu do makr OBJECT-TYPE w modułach bazy MIB. Zauważmy, że te dopasowania czy odmiany nie odnoszą się do makr MODULE-CAPABILITIES.

Formalna definicja możliwości agenta może być pomocna przy optymalizacji współdziałania. Jeżeli stacja zarządzająca zawiera określenia możliwości wszystkich agentów, z którymi współdziała, wówczas może tak dopasować ich zachowanie, aby zapewnić optymalne wykorzystanie zasobów własnych, agenta i sieciowych.

Listing 13.4 prezentuje makro AGENT-CAPABILITIES. Klauzula PRODUCT-RELEASE zawiera tekstowy opis wersji produktu zawierającego danego agenta, a klauzula DESCRIPTION zawiera tekstowy opis samego agenta. Reszta definicji zawiera po jednej sekcji dla każdego modułu MIB, dla którego agent zapewnia pełną bądź częściową implementację.

Listing 13.4. *Makro AGENT-CAPABILITIES*

```
AGENT-CAPABILITIES MACRO ::= BEGIN
TYPE NOTATION ::= "PRODUCT-RELEASE" Text
                  "STATUS" Status
                  "DESCRIPTION" Text
                  ReferPart
                  ModulePart
VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
Status ::= "current" | "obsolete"
ReferPart ::= "REFERENCE" Text | Empty
ModulePart ::= Modules | empty
Modules ::= Module | Modules Module
Module ::= "SUPPORTS" ModuleName
           "INCLUDES" "{" Groups "}"
           VariationPart
ModuleName ::= identifier ModuleIdentifier
ModuleIdentifier ::= value (moduleID OBJECT IDENTIFIER) | empty
Groups ::= Group | Groups "," Group
Group ::= value (Name OBJECT IDENTIFIER)
VariationPart ::= Variations | empty
Variations ::= Variation | Variations Variation
Variation ::= ObjectVariation | NotificationVariation
NotificationVariation ::= "VARIATION" value (Name NotificationName)
                          AccessPart
                          "DESCRIPTION" Text
ObjectVariation ::= "VARIATIONS" value (Name ObjectName)
                    SyntaxPart
                    WriteSyntaxPart
                    AccessPart
                    CreationPart
                    DefValPart
                    "DESCRIPTION" value (description Text)
SyntaxPart ::= "SYNTAX" type (SYNTAX) | empty
WriteSyntaxPart ::= "WRITE-SYNTAX" type (WriteSYNTAX) | empty
AccessPart ::= "ACCESS" Access | empty
Access ::= "not-implemented" | "accessible-for-notify" | "read-only" | "read-write" |
           "read-create" | "write-only"
CreationPart ::= "CREATION-REQUIRES" "{" Cells "}" empty
Cells ::= Cell | Cells "," Cell
Cell ::= value (Cell ObjectName)
DefValPart ::= "DEFVAL" "{" value (Defval ObjectSyntax) "}" | empty
Text ::= """ string """
END
```

Opis każdego modułu MIB rozpoczyna się klauzulą SUPPORTS, określającą nazwę modułu. Następnie klauzula INCLUDES specyfikuje wykaz grup MIB z tego modułu, które agent implementuje. Ostatecznie, dla każdej obsługiwanej grupy MIB, w definicji zawartych może być zero lub więcej specyfikacji obiektów, które agent implementuje w odmienny lub dopasowany sposób w porównaniu z makrodefinicją OBJECT-TYPE tych obiektów. Dla każdego z takich obiektów określa się, co następuje. Po pierwsze, klauzula VARIATIONS określa nazwę takiego obiektu. Następnie wystąpić może jedna lub więcej części określających dopasowania. SyntaxPart i WriteSyntaxPart mają tę samą semantykę, co analogiczne elementy makra MODULE-COMPLIANCE. AccessPart używane jest do oznaczenia, że agent zapewnia niższy poziom dostępu niż określony w klauzuli MAX-ACCESS definicji danego obiektu. CreationPart określa nazwy obiektów kolumnowych z wierszy pojęciowych, którym należy bezpośrednio przypisać wartości poprzez operację set protokołu zarządzania, aby agent umożliwił zmianę stanu instancji kolumny statusowej tych wierszy na aktywny (active (4)). Element DefVal określa uściśloną wartość DEFVAL dla danego obiektu. Klauzula DESCRIPTION zawiera tekstowy opis odmiany bądź dopasowania implementacji.

Wartość podawana przy wywoływaniu makra AGENT-CAPABILITIES jest identyfikatorem obiektu przypisanym do danej definicji możliwości.

Specyfikacja SNMPv2 zawiera użytkowy przykład określania możliwości, pokazany na listingu 13.5. W przykładzie tym agent implementuje SNMPv2, interfejsy, IP, TCP, UDP i moduły EVAL bazy MIB. Dla każdego z tych modułów określono zakres implementacji.

Listing 13.5. *Przykład określenia możliwości agenta z wykorzystaniem makra AGENT-CAPABILITIES*

```
example-agent AGENT-CAPABILITIES
  PRODUCT RELEASE          "Wydanie 1.1 agenta ACME dla 4BSD"
  STATUS                   current
  DESCRIPTION              "agent ACME dla 4BSD"

  SUPPORTS                 SNMPv2-MIB
    INCLUDES               { systemGroup, snmpGroup, snmpSetGroup,
                           snmpBasicNotificationsGroup }
      VARIATION            coldStart
      DESCRIPTION          "Pułapka coldStart generowana przy każdym ponownym
                           ↳uruchamianiu"
  SUPPORTS                 IF-MIB
    INCLUDES               { ifGeneralGroup, ifPacketGroup }

      VARIATION            ifAdminStatus
        SYNTAX             INTEGER { up(1), down(2) }
        DESCRIPTION        "W 4BSD nie można ustawić trybu testowania"
      VARIATION            ifOperStatus
        SYNTAX             INTEGER { up(1), down(2) }
        DESCRIPTION        "W 4BSD informacja jest ograniczona"

  SUPPORTS                 IP-MIB
    INCLUDES               { ipGroup, icmpGroup }

      VARIATION            ipDefaultTTL
        SYNTAX             INTEGER (255..255)
        DESCRIPTION        "Taka jest wartość w 4BSD"
```

```
    VARIATION           ipInAddrErrors
      ACCESS            not-implemented
      DESCRIPTION       "Informacja niedostępna w 4BSD"

    VARIATION           ipNetToMediaEntry
      CREATION-REQUIRES { ipNetToMediaPhysAddress }
      DESCRIPTION       "Odwzorowanie adresu w 4BSD wymaga zarówno adresu
                        ↳protokołu, jak i adresu medium"

  SUPPORTS              TCP-MIB
    INCLUDES            { tcpGroup }

    VARIATION           tcpConnState
      ACCESS            read-only
      DESCRIPTION       "W 4BSD nie można tego zmieniać"

  SUPPORTS              UDP-MIB
    INCLUDES            { udpGroup }

  SUPPORTS              EVAL-MIB
    INCLUDES            { functionsGroup, expressionsGroup }

    VARIATION           exprEntry
      CREATION-REQUIRES { evalString }
      DESCRIPTION       "Można tworzyć wiersze"

  ::= { acmeAgents 1 }
```

13.3. Rozwinięcie grupy interfaces z bazy MIB-II

Jak wyjaśniono w rozdziale 6., dokument RFC 1573 (*Evolution of the Interfaces Group of MIB-II — Rozwinięcie grupy interfaces w bazach MIB-II*) porządkuje i udoskonala grupę `interfaces` z RFC 1213 (MIB-II), wykorzystując w definicjach SMIv2.

Grupa `interfaces` w bazach MIB-II definiuje ogólny zestaw zarządzanych obiektów, tak by każdy interfejs sieciowy mógł być zarządzany niezależnie od jego typu. Takie ogólne podejście jest dostosowane do typowych architektur protokołów, w których protokół międzysieciowy, taki jak IP, zaprojektowany został do pracy ponad jakimkolwiek interfejsem sieciowym. Poza tym dzięki użyciu modułów dopasowanych do typu architektur, takich bazy MIB dla sieci ethernet czy token-ring, możliwe jest dodanie kolejnych obiektów wymaganych w danym typie interfejsu sieciowego.

Doświadczenia z pracy z grupą `interfaces` i modułami specyficznymi dla różnych typów sieci wykazały istnienie pewnych niedostatków tej grupy, zdefiniowanej w MIB-II. Dokument RFC 1573 zajmuje się tymi brakami przez wyjaśnienia, korekty i rozwinięcie struktury MIB przeznaczonej dla interfejsów. Dokument ten obejmuje w szczególności następujące problemy:

1. *Numeracja interfejsu* (*Interface Numbering*) — grupa `interfaces` z MIB-II (rysunek 6.2) definiuje obiekt `ifNumber` jako liczbę interfejsów sieciowych obecnych w systemie i specyfikuje, że każda wartość obiektu `ifIndex` musi

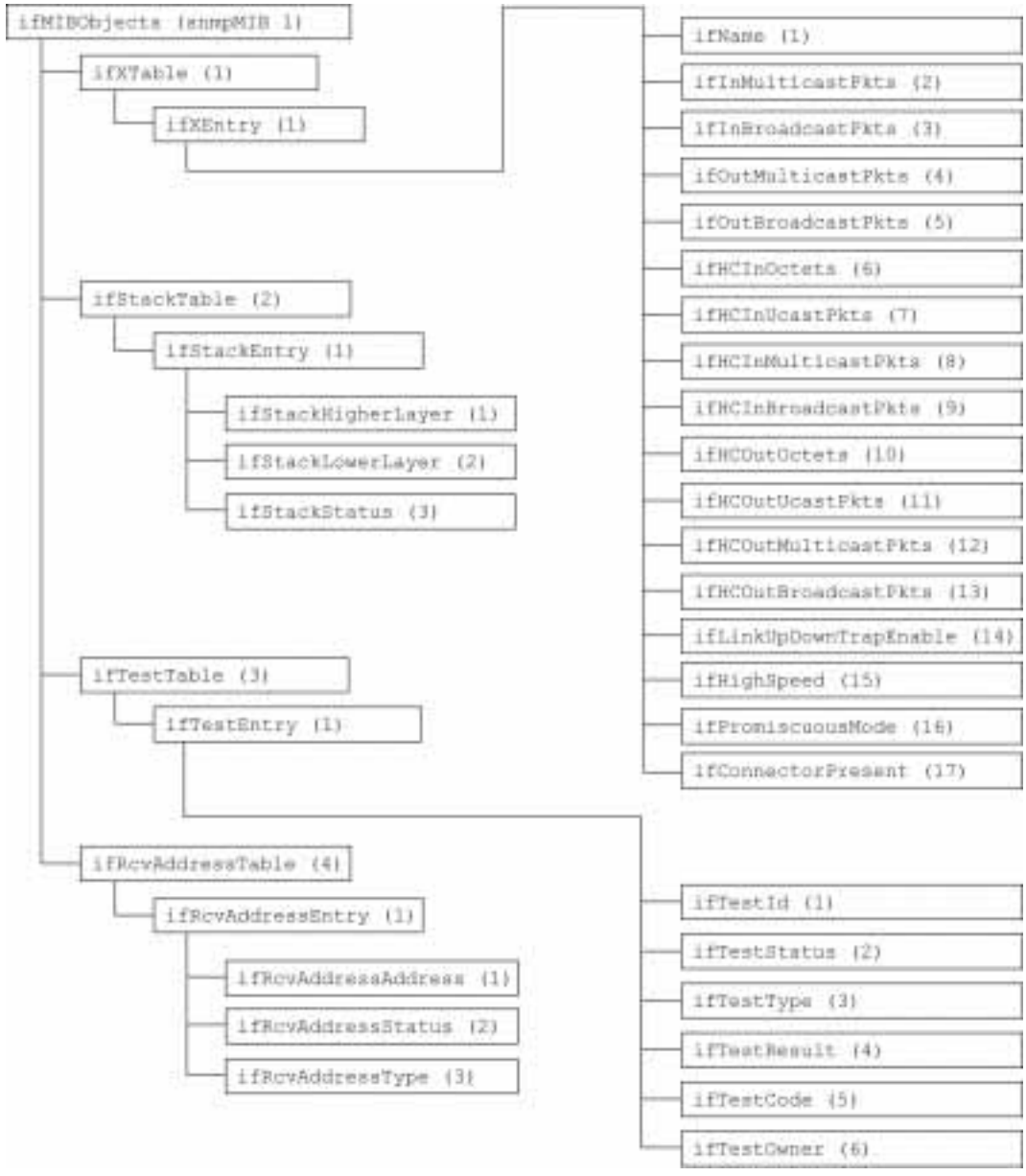
zawierać się w przedziale od 1 do wartości `ifNumber` i pozostawać niezmienna. To wymaganie jest kłopotliwe w urządzeniach pozwalających na dynamiczne dodawanie i usuwanie interfejsów sieciowych, tak jak ma to miejsce na przykład w przypadku połączeń typu SLIP/PPP.

2. *Podwarstwy interfejsu* (*Interface Sublayers*) — istnieje konieczność wyróżniania kilku podwarstw poniżej warstwy międzysieciowej.
3. *Połączenia wirtualne* (*Virtual Circuits*) — potrzebne jest miejsce na odnotowywanie faktu, że pod warstwą międzysieciową danego interfejsu znajdować się mogą różne połączenia wirtualne.
4. *Interfejsy bitowe, znakowe i o stałej długości* (*Bit, Charakter and Fixed-Length Interfaces*) — zorientowanie tablicy `ifTable` na transmisję pakietową może być nieodpowiednie w przypadku interfejsów niepakietowych z natury, jak choćby wykorzystujących transmisję znakową (przykładowo PPP na EIA-232), bitową (na przykład DS1) czy przesyłających paczki o stałej, określonej długości (ATM).
5. *Rozmiar liczników* (*Counter Size*) — wraz ze wzrostem szybkości sieci minimalny czas dla 32-bitowych liczników uległ skróceniu, powodując powstawanie problemów z przepełnieniami.
6. *Prędkość interfejsu* (*Interface Speed*) — przedział wartości obiektu `ifSpeed` jest ograniczony od góry do $2^{31} - 1$ b/s lub poniżej 2,2 Gb/s. Taka prędkość jest osiągana lub nawet przekraczana przez niektóre interfejsy (np. SONET OC-48 – 2,448 Gb/s).
7. *Liczniki Multicast/Broadcast* (*Multicast/Broadcast Counters*) — liczniki w `ifTable` przewidziane są do łącznego obejmowania transmisji typu *Multicast* i *Broadcast*. Niekiedy przydatne są jednak odrębne liczniki dla pakietów obu typów.
8. *Dodanie nowych wartości* `ifType` — potrzebna jest możliwość dodawania nowych wartości wyliczeniowych obiektu `ifType`. Sposób zdefiniowania obiektu `ifType` w bazie MIB-II powoduje, że nowe wartości dostępne są tylko w nowych wydaniach MIB, co zdarza się raz na kilka lat.
9. `ifSpecific` — definicja obiektu `ifSpecific` w MIB-II jest niejednoznaczna. Niektórzy implementatorzy nadali temu obiektowi wartość identyfikatora typu `OBJECT IDENTIFIER` bazy MIB dostosowanej do rodzaju stosowanego medium. Inni wykorzystali do tego celu identyfikator tabeli dostosowanej do rodzaju medium lub identyfikator wpisu z tej tabeli bądź nawet identyfikator obiektu indeksowego tej tabeli.

Dalej przyjrzymy się, w jaki sposób uwzględniono każdy z tych problemów.

Dokument RFC 1573 zawiera powtórzenie, z niewielkimi modyfikacjami, definicji grupy `interfaces` z MIB-II. Dodatkowo wprowadza cztery nowe tabele (rysunek 13.4):

- *Tabelę rozszerzeń* (*Extensions Table*) (`ifXTable`),
- *Tabelę stosu* (*Stach Table*) (`ifStackTable`),
- *Tabelę testów* (*Test Table*) (`ifTestTable`),
- *Tabelę otrzymanych adresów* (*Receive Address Table*) (`ifRcvAddressTable`).



Rysunek 13.4. *Dodatki do grupy interfaces wprowadzone w SNMPv2*

13.3.1. Grupa interfaces

Grupa ta w RFC 1573 ma identyczną strukturę jak grupa `interfaces` w MIB-II. Składa się więc z obiektu `ifNumber` i tabeli `ifTable`. Ważną różnicą jest klauzula `DESCRIPTION` obiektu `ifIndex`, która brzmi następująco:

Wielkość jednoznaczna, większa od zera, dla każdego interfejsu bądź podwarstwy interfejsu w zarządzanym systemie. Zaleca się, by przydzielane były kolejne numery, poczynając od 1. Wartość ta dla każdej podwarstwy interfejsu musi pozostawać niezmienna przynajmniej od momentu inicjalizacji danej jednostki systemu zarządzania do momentu kolejnej jej inicjalizacji.

Obiekt `ifNumber` nadal odzwierciedla liczbę interfejsów, a więc również liczbę wierszy w tabeli `ifTable`. Jednakże nie jest konieczne ograniczanie zakresu numeracji do przedziału od 1 do wartości `ifNumber`. Pozwala to na dynamiczne dodawanie i usuwanie interfejsów.

Dokument RFC 1573 pozwala, aby do jednego fizycznego interfejsu odnosiło się wiele wierszy tabeli `ifTable`, po jednym dla każdej logicznej podwarstwy. Jednak dokument ten zaleca oszczędne stosowanie tego typu strategii. Poza tym zaleca się niestosowanie oddzielnych wierszy w tabeli dla połączeń wirtualnych.

Ranga niektórych obiektów tabeli `ifTable` została zdeprecjonowana. I tak ograniczono znaczenie obiektów `ifInNUcastPkts` i `ifOutNUcastPkts`, zliczających wszystkie pakiety typu *Nonunicast*, wprowadzając w tabeli `ifXTable` liczniki odrębnie zliczające pakiety typu *Multicast* i *Broadcast*. Podobnie ma się rzecz z obiektem `ifOutQLen`, który był rzadko implementowany. Także obiekt `ifSpecific` stracił znaczenie ze względu na swoją niejednoznaczność i fakt, że nie dostarcza żadnych dodatkowych informacji poza tym, co zapewnia obiekt `ifType`.

Kolejną zmianą w tabeli `ifTable` jest inna składnia `ifType`, będąca obecnie konwencją tekstową `IANAifType`, która może zostać przedefiniowana (przez wprowadzenie nowych wartości) bez konieczności ogłaszania nowej wersji bazy MIB. Za aktualizację `IANAifType` jest odpowiedzialna organizacja *IANA* (*Internet Assigned Number Authority — Władze przydzielania numerów w Internecie*).

13.3.2. Rozszerzenie tabeli interfejsu

Tabela `ifXTable` dostarcza dodatkowych informacji w uzupełnieniu tabeli `ifTable`. Tabela ta i obiekty do niej wpisywane definiowane są następująco:

```
IfXTable OBJECT-TYPE
  SYNTAX       SEQUENCE OF IfXEntry
  MAX-ACCESS   not-accessible
  STATUS       current
  DESCRIPTION
    "Wykaz wpisów dotyczących interfejsów. Liczba wpisów określona jest przez wartość
    obiektu ifNumber. Tabela ta zawiera dodatkowe obiekty dla tabeli interfejsu."
  ::= { ifMIBObjects 1 }
ifXEntry OBJECT-TYPE
  SYNTAX       IfXEntry
  MAX-ACCESS   not-accessible
  STATUS       current
  DESCRIPTION
    "Wpis zawierający dodatkowe informacje zarządzania, odpowiednie dla danego
    interfejsu"
  AUGMENTS { ifEntry }
  ::= { ifXTable 1 }
```

```
IfXEntry ::= SEQUENCE { ifName                    DisplayString,
                        IfInMulticastPkts         Counter32,
                        IfInBroadcastPkts         Counter32,
                        IfOutMulticastPkts        Counter32,
                        IfOutBroadcastPkts        Counter32,
                        IfHCInOctets              Counter64,
                        IfHCInUcastPkts           Counter64,
                        IfHCInMulticastPkts       Counter64,
                        IfHCInBroadcastPkts       Counter64,
                        IfHCOutOctets             Counter64,
                        IfHCOutUcastPkts          Counter64,
                        IfHCOutMulticastPkts      Counter64,
                        IfHCOutBroadcastPkts      Counter64,
                        IfLinkUpDownTrapEnable    INTEGER,
                        IfHighSpeed               Gauge32,
                        IfPromiscuousMode         TruthValue,
                        IfConnectorPresent        TruthValue }
```

Jak widać, tabela `ifXTable` jest poszerzeniem tabeli `ifTable`. Dlatego też indeksowana jest przez `ifIndex` z `ifTable`. Tabela ta zawiera obiekt `ifName` będący tekstowym odwołaniem do interfejsu. Jeśli różne wpisy tej tabeli odwołują się do różnych podwarstw tego samego interfejsu, wówczas wszystkie one mają taką samą wartość `ifName`.

Następne cztery obiekty kolumnowe (`ifInMulticastPkts`, `ifInBroadcastPkts`, `ifOutMulticastPkts`, `ifOutBroadcastPkts`) zliczają pakiety typu *Mulicast* i *Broadcast* odebrane i transportowane przez dany interfejs. Liczniki te zastąpiły wcześniejsze `ifInNUcastpkts` i `ifOutNUcastPkts` z tabeli `ifTable`.

Dalsze osiem obiektów (`ifHCInOctets`, `ifHCInUcastPkts`, `ifHCInMulticastPkts`, `ifHCInBroadcastPkts`, `ifHCOutOctets`, `ifHCOutUcastPkts`, `ifHCOutMulticastPkts`, `ifHCOutBroadcastPkts`) określa się jako liczniki „dużej pojemności". Wszystkie one są 64-bitowymi wersjami analogicznych liczników z tabeli `ifTable`, z tą samą semantyką. Dzięki nim agent jest w stanie poprawnie zliczać pakiety w interfejsach o dużej prędkości przesyłu danych.

Obiekt `ifLinkUpDownTrapEnable` jest wyliczeniowym typem `INTEGER` z wartościami `enabled(1)` i `disabled(2)`. Obiekt ten wskazuje, czy dla danego wpisu powinny być generowane pułapki typu `linkUp` i `linkDown`. Domyślnie obiekt ten powinien mieć wartość `disabled(2)`, jeśli dany wpis definiuje interfejs działający w oparciu o inny interfejs (jak określono w `ifStackTable`). W przeciwnym razie generowanie wspomnianych pułapek jest dopuszczalne i obiekt ten powinien być ustawiony na wartość `enable(1)`.

Następny obiekt, `ifHighSpeed`, to miernik estymujący aktualną szybkości transferu danych interfejsu wyrażoną w Mb/s. Jeżeli obiekt ten ma wartość n, to rzeczywista prędkość przesyłu danych mieści się w jednostronnie domkniętym przedziale $(n-0.5, n+0.5]$.

Obiekt `ifPromiscuousMode` jest wyliczeniowym typem `INTEGER` z wartościami `true(1)` i `false(2)`. Obiekt ten ma wartość `true(1)` wówczas, gdy interfejs akceptuje wszystkie transportowane ramki lub pakiety, i `false(2)`, gdy interfejs akceptuje tylko te pakiety (ramki), które adresowane są dla danej stacji. Definicja ta nie obejmuje pakietów typu *Multicast* i *Broadcast*.

Ostatni obiekt, `ifConnectorPresent`, ma wartość `true(1)`, gdy podwarstwa interfejsu posiada łącze fizyczne, i wartość `false(2)` w przeciwnym razie.

13.3.3. Tabela stosu interfejsu

Tabela `ifStackTable` ukazuje relacje pomiędzy różnymi wierszami tabeli `ifTable` związanymi z tym samym fizycznym interfejsem do medium. Wskazuje te podwarstwy, które działają ponad innymi. Każdy wpis tabeli `ifStackTable` definiuje powiązania między dwoma wpisami w `ifTable`. Obiekt `ifStackHigherLayer` zawiera wartość indeksu `ifIndex` wyższej z dwóch powiązanych podwarstw; z kolei obiekt `ifStackLowerLayer` ma wartość indeksu `ifIndex` niższej podwarstwy. Tabela ta jest podwójnie indeksowana przez te obiekty. Do tworzenia i usuwania wpisów tej tabeli wykorzystywany jest obiekt `ifStackStatus` posiadający składnię konwencji `RowStatus` (zobacz dodatek 11A).

13.3.4. Tabela testów interfejsu

Tabela `ifTestTable` określa obiekty, umożliwiające zarządcy nakazanie agentom przeprowadzenie różnych testów interfejsu. Tabela ta zawiera jeden wpis dla każdego interfejsu. Obiekty z tej tabeli zebrano w tabeli 13.3.

Tabela 13.3. *Obiekty tabeli ifTestTable*

Obiekt	Składnia	Tryb dostępu	Opis
`ifTestTable`	`SEQUENCE OF ifTestEntry`	NA	Tabela umożliwiająca zarządcy przeprowadzanie testów
`ifTestEntry`	`SEQUENCE`	NA	Opis testu dla danego interfejsu
`ifTestId`	`TestAndIncr`	RW	Identyfikuje bieżące wywołanie testu
`ifTestStatus`	`INTEGER`	RW	Wskazuje, czy któryś zarządca ma aktualnie prawa wymagane do wywołania testu danego interfejsu; przyjmuje wartości `InUse(1)` i `InUse(2)`
`ifTestType`	`AutonomousType`	RW	Identyfikuje test aktualnie trwający bądź taki, który ma zostać wywołany
`ifTestResult`	`INTEGER`	RO	Wskazuje wynik ostatniego testu
`ifTestCode`	`OBJECT IDENTIFIER`	RO	Kod zawierający szczegółowe informacje o wyniku testu
`ifTestOwner`	`OwnerString`	RW	Wskazuje na właściciela danego wpisu tabeli

Każdy wpis w tabeli `ifTestTable` daje trzy możliwości:

1. Umożliwia zarządcy, przez ustawienie wartości obiektu `ifTestType`, określenie testu, jakiemu poddany ma zostać interfejs. Po zadaniu tej wartości agent uruchamia test.
2. Umożliwia zarządcy uzyskanie wyników testu przez odczyt wartości obiektów `ifTestResult` i `ifTestCode`. Wyniki są rejestrowane we wspomnianych obiektach po zakończeniu testu przez agenta.
3. Zapewnia mechanizm umożliwiający poprawne przeprowadzenie testu tylko jednemu zarządcy w danej chwili. Jednocześnie gwarantuje, że żaden test nie zostanie zażądany w trakcie trwania innego testu. Wykorzystuje się w tym celu obiekty `ifTestId` oraz `ifTestStatus`.

Listing 13.6, zaczerpnięty z definicji tabeli ifTestTable, objaśnia logikę korzystania z tej tabeli. Kiedy zarządca zechce przeprowadzić test na danym interfejsie, najpierw wysyła rozkaz GetRequest w celu pobrania właściwego wiersza tabeli i odczytania wartości obiektów ifTestId i ifTestStatus. Jeśli status testu ma wartość notInUse, zarządca może kontynuować procedurę; w przeciwnym razie musi ponawiać pytanie aż do zwolnienia wiersza.

Listing 13.6. *Logika tabeli ifTestTable*

```
nowa_próba:
      pobierz (ifTestId, ifTestStatus)
      dopóki (ifTestStatus != notInUse) {
               /*
                *Pętla powtarzana tak długo,
                *jak długo trwa test lub inny zarządca go konfiguruje
                */
                krótkie opóźnienie
                pobierz(ifTestId, ifTestStatus)
                }
      /*
       *Test nie używany - spróbujmy go przejąć
       */
       wartość_blokady = ifTestId
       jeśli( ustaw(ifTestId = wartość_blokady, ifTestStatus = InUse,
                    ifTestOwner = 'mój-adres-IP') == BŁĄD)
                    /*
                     *Inny zarządca przejął ten wiersz - powrót do 'nowa_próba'
                     */
                    skok_do nowa_próba;
      /*
       *Blokada ustawiona
       */
      ustawienie parametrów testu
      /*
       *Uruchomienie testu
       */
      ustaw(ifTestType = test_do_przeprowadzenia);
      oczekiwanie na zakończenie testu i odczytywanie wartości ifTestResult
      po zakończeniu testu agent ustawia wartość ifTestResult
      agent ustawia poza tym obiekt ifTestStatus na 'notInUse'
      pobranie wszelkich dodatkowych wyników testu oraz ifTestId
      jeśli(ifTestId == wartość_blokady+1) wyniki są poprawne
```

Gdy okaże się, że dany wiersz nie jest zajęty, zarządca będzie próbował zażądać testu, używając w tym celu tej samej wartości ifTestId, którą ostatnio odebrał. Wartość ta jednoznacznie identyfikuje każdy test. Zarządca wysyła SetRequest, próbując ustawić pole ifTestId na odebraną wartość. Obiekt ifTestId ma składnię TestAndIncr. Przypomnijmy z podrozdziału 12.3.5, że obiekt o takiej składni używany jest w następujący sposób: jeżeli bieżąca wartość instancji tego obiektu w agencie wynosi K i od zarządcy odebrane zostanie żądanie ustawienia go na tę samą wartość, operacja ta kończy się pomyślnie, a wartość obiektu zwiększana jest o 1. W przypadku, gdy odebrana wartość jest różna od K, test się nie udaje. Tak więc jeśli zarządca odczyta wartość pola ifTestId, a następnie spróbuje ustawić je na taką samą wartość, to próba ta zakończy się pomyślnie jedynie wówczas, gdy żaden inny zarządca nie przeszkodził i nie rozpoczął własnego testu.

Jeżeli ustawienie `ifTestId` powiedzie się, agent zmieni wartość `ifTestStatus` na `InUse`, blokując w ten sposób próby innych zarządców, a `ifSetOwner` na wartość przesłaną przez zarządcę. Następnie agent wyśle PDU z odpowiedzią informującą zarządcę o pomyślnym przeprowadzeniu operacji. W ten sposób dany zarządca przejmuje konkretny wiersz w tymczasowe posiadanie.

Po przejęciu wiersza zarządca może kontynuować wywołanie testu. Odbywa się to przez wysłanie rozkazu `SetRequest` ustawiającego `ifTestType` na wartość wskazującą, który test należy przeprowadzić. W odpowiedzi agent rozpoczyna wykonywanie testu i ustawia pole `ifTestResult` na wartość `inProgress(3)`. Po zakończeniu testu umieszcza jego wyniki w polu `ifTestResult`, które przybierać może następujące wartości:

- `none(1)` — do tej pory nie zażądano przeprowadzenia żadnego testu,
- `success(2)` — test zakończony pomyślnie,
- `inProgress(3)` — test trwa,
- `notSupported(4)` — test nie jest zaimplementowany,
- `unAbletoRun(5)` — test nie może zostać przeprowadzony w związku ze stanem systemu,
- `aborted(6)` — test został przerwany,
- `failed(7)` — test zakończył się niepomyślnie.

Dodatkowe informacje mogą się znajdować w `ifTestCode`.

13.3.5. Ogólna tabela odebranych adresów

Tabela ta zawiera po jednym wpisie dla każdego adresu (typu *Multicast*, *Broadcast* i *Unicast*), dla którego dany system odbierać będzie pakiety w jednym z interfejsów, z wyjątkiem pracy w trybie *Promiscuous*. Oznacza to, że tabela ta zawiera wszystkie adresy, które system rozpoznaje i dla których przechwytywać będzie pakiety zawierające te adresy jako jeden z adresów docelowych.

Tabela ta składa się z trzech obiektów kolumnowych:

- `ifRcvAddressAddress` — konkretny adres typu *Multicast*, *Broadcast* lub *Unicast*, który system rozpoznaje jako odpowiedni adres docelowy do przechwytywania pakietów,
- `ifRcvAddressStatus` — używany do tworzenia i likwidacji wierszy w tej tabeli, posiada składnię `RowStatus`,
- `ifRcvAddressType` — wskazuje, czy adres jest typu `other(1)`, `volatile(2)` czy `nonVolatile(3)`; adres typu `nonvolatile` (*nieulotny*) będzie istniał także po restarcie systemu, natomiast adres typu `volatile` (*ulotny*) zostanie utracony. Adres typu `other` (*inny*) oznacza, że informacja ta nie została udostępniona w danej tablicy.

13.4. Posumowanie

W rozdziale tym opisano dwie bazy MIB związane ze specyfikacją SNMPv2. Baza SNMPv2 MIB zawiera informacje dotyczące wykorzystania samego protokołu. Rozszerzenie grupy `interfaces`, wprowadzone w RFC 1573, zdefiniowane jest przy użyciu SMIv2 i wykorzystuje niektóre właściwości SNMPv2 omówione w tym rozdziale.

Specyfikacja SNMPv2 zawiera mechanizm opisu wymagań zgodności z daną bazą MIB oraz środki umożliwiające producentom określanie zakresu swoich implementacji. W dokumencie poruszającym zagadnienia zgodności zdefiniowano cztery makra:

- `OBJECT-GROUP` — wskazuje te obiekty w MIB, które są częścią grupy zgodności,
- `NOTIFICATION-GROUP` — identyfikuje zbiór powiadomień,
- `MODULE-COMPLIANCE` — ustala wymagania w stosunku do agenta pod względem implementacji modułów i obiektów bazy MIB,
- `AGENT-CAPABILITIES` — definiuje możliwości poszczególnych implementacji agenta.

Dodatek 13A. Konwencja tekstowa TestAndIncr

TestAndIncr ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION

„Reprezentuje informację w postaci liczby całkowitej, wykorzystywaną do operacji atomowych. Jeżeli protokół zarządzania wykorzystany zostanie do zmiany wartości instancji obiektu o tej składni, nowa wartość dostarczana poprzez protokół zarządzania musi być identyczna z aktualną wartością tego obiektu. W przeciwnym razie operacja `set` protokołu zarządzania zakończy się niepowodzeniem i zgłoszeniem błędu `inconsistentValue` (*sprzeczna wartość*). W przypadku zgodności nadesłanej wartości: jeżeli aktualna wartość tego obiektu jest maksymalna (tj. 2^{31} czyli 2 147 483 647 dziesiętnie), wówczas zostanie wyzerowana, w innym przypadku zwiększana jest o jeden (zauważmy, że niezależnie od tego, czy operacja `set` protokołu zarządzania się powiedzie, pola `variable-bindings` w jednostce PDU żądania i odpowiedzi są identyczne).

Wartość klauzuli `ACCESS` dla obiektu mającego taką składnię jest albo *read-write*, albo *read create*. W momencie tworzenia obiektu kolumnowego o takiej składni jego wartość może być dowolnie określona poprzez protokół zarządzania.

W przypadku reinicjalizacji części systemu związanej z zarządzaniem siecią wartość wszystkich instancji obiektów o tej składni musi być albo zwiększana od wartości sprzed reinicjalizacji, albo (jeśli wartość ta jest nie znana) musi być nadana jej wartość pseudolosowa."

SYNTAX INTEGER (0..2147483647)