

Sieci i chmury teleinformatyczne

Laboratorium nr 3 - Wprowadzenie do SNMP

Rafał Dadura, Juliusz Kuzyka

Listopad 2023

Spis treści

1	Cel laboratorium	2
2	Konfigurowanie środowiska	2
2.1	4.3.R1	2
3	Podstawowe operacje SNMP (interfejs CLI)	2
3.1	Operacja GET (agent lokalny)	2
3.1.1	5.1.R1	2
3.1.2	5.1.R2	2
3.1.3	5.1.R3	4
3.1.4	5.1.R4	5
3.2	Operacja GET-NEXT	6
3.2.1	5.2.R1	6
3.2.2	5.2.R2	6
3.2.3	5.2.R3	6
3.3	Operacja SET	6
3.3.1	5.3.R1	6
3.3.2	5.3.R2	7
4	Podstawowe operacje SNMP (interfejs graficzny)	8
4.1	Zapoznanie się z interfejsem przeglądarki	8
4.1.1	6.2.R1	8
4.2	Operacja NOTIFY	9
4.2.1	6.3.R1	9
4.2.2	6.3.R2	9
4.3	Obserwacja połączeń TCP	10
4.3.1	6.4.R1	10
4.3.2	6.4.R2	10
4.3.3	6.4.R3	11
4.3.4	6.4.R4	11
4.4	Obserwacja interfejsów sieciowych hosta	11
4.4.1	6.5.R1)	11
4.4.2	6.5.R2	11
4.4.3	6.5.R3	12
4.4.4	6.5.R4	12
5	Wnioski	13

1 Cel laboratorium

Cele tego laboratorium związane są z zapoznaniem się z podstawowymi zasadami interakcji z agentem SNMP (Simple Network Management Protocol).

2 Konfigurowanie środowiska

2.1 4.3.R1

Program `snmpd` to agent SNMP (Simple Network Management Protocol), który działa na urządzeniach sieciowych i umożliwia zdalne monitorowanie i zarządzanie tymi urządzeniami. Jest to kluczowy element w architekturze SNMP, który reprezentuje urządzenie w sieci i udostępnia dane o stanie oraz parametrach tego urządzenia.

Wartości jakie przypisaliśmy obiektom:

- `sysLocation` - juliusz_rafal
- `sysContact` - juliusz_rafal@mail.pl

3 Podstawowe operacje SNMP (interfejs CLI)

3.1 Operacja GET (agent lokalny)

3.1.1 5.1.R1

Nasze zadanie rozpoczęliśmy od wykonania poleceń dla wszystkich wartości parametru `x = 1,2,3,4,5,6`:

```
$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.<x>.0
```

```
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.0
iso.3.6.1.2.1.1.1.0 = STRING: "Linux schtlab 5.4.0-150-generic #167~18.04.1-Ubun
tu SMP Wed May 24 00:51:42 UTC 2023 x86_64"
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.2.0
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.8072.3.2.10
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (39254) 0:06:32.54
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.4.0
iso.3.6.1.2.1.1.4.0 = STRING: "juliusz_rafal@mail.pl"
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.5.0
iso.3.6.1.2.1.1.5.0 = STRING: "snmpsandbox"
student@schtlab:~$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.6.0
iso.3.6.1.2.1.1.6.0 = STRING: "juliusz_rafal"
student@schtlab:~$
```

Rysunek 1: Operacje GET w terminalu

3.1.2 5.1.R2

Następnie w katalogu `$HOME/Cwiczenie/SNMP-MIBS` odnaleźliśmy plik **RFC1213-MIB.txt** oraz odszukaliśmy definicje odczytanych obiektów.

```
sysDescr OBJECT-TYPE
SYNTAX  DisplayString (SIZE (0..255))
ACCESS  read-only
STATUS  mandatory
DESCRIPTION
    "A textual description of the entity. This value
    should include the full name and version
    identification of the system's hardware type,
    software operating-system, and networking
```

```

        software. It is mandatory that this only contain
        printable ASCII characters."
 ::= { system 1 }

sysObjectID OBJECT-TYPE
    SYNTAX  OBJECT IDENTIFIER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The vendor's authoritative identification of the
        network management subsystem contained in the
        entity. This value is allocated within the SMI
        enterprises subtree (1.3.6.1.4.1) and provides an
        easy and unambiguous means for determining what
        kind of box' is being managed. For example, if
        vendor Flintstones, Inc.' was assigned the
        subtree 1.3.6.1.4.1.4242, it could assign the
        identifier 1.3.6.1.4.1.4242.1.1 to its `Fred
        Router'."
 ::= { system 2 }

sysUpTime OBJECT-TYPE
    SYNTAX  TimeTicks
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The time (in hundredths of a second) since the
        network management portion of the system was last
        re-initialized."
 ::= { system 3 }

[16:00]
sysContact OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The textual identification of the contact person
        for this managed node, together with information
        on how to contact this person."
 ::= { system 4 }

sysName OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "An administratively-assigned name for this
        managed node. By convention, this is the node's
        fully-qualified domain name."
 ::= { system 5 }

sysLocation OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION

```

```

        "The physical location of this node (e.g.,
        `telephone closet , 3rd floor ')."
```

::= { system 6 }

[16:01]

sysServices OBJECT-TYPE

SYNTAX INTEGER (0..127)

ACCESS read-only

STATUS mandatory

DESCRIPTION

"A value which indicates the set of services that this entity primarily offers.

The value is a sum. This sum initially takes the value zero, Then, for each layer, L, in the range 1 through 7, that this node performs transactions for, 2 raised to (L - 1) is added to the sum. For example, a node which performs primarily routing functions would have a value of 4 ($2^{(3-1)}$). In contrast, a node which is a host offering application services would have a value of 72 ($2^{(4-1)} + 2^{(7-1)}$). Note that in the context of the Internet suite of protocols, values should be calculated accordingly:

layer	functionality
1	physical (e.g., repeaters)
2	datalink/subnetwork (e.g., bridges)
3	internet (e.g., IP gateways)
4	end-to-end (e.g., IP hosts)
7	applications (e.g., mail relays)

For systems including OSI protocols, layers 5 and 6 may also be counted."

::= { system 7 }

3.1.3 5.1.R3

Następnie wykonaliśmy kilkukrotnie operację GET na obiekcie SysUpTime, którego definicję znaleźliśmy w pliku RFC1213-MIB.txt.

```

student@schtlab:~/Cwiczenie/SNMP-MIB$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (205210) 0:34:12.10
student@schtlab:~/Cwiczenie/SNMP-MIB$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (205369) 0:34:13.69
student@schtlab:~/Cwiczenie/SNMP-MIB$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (205522) 0:34:15.22
student@schtlab:~/Cwiczenie/SNMP-MIB$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (205672) 0:34:16.72
student@schtlab:~/Cwiczenie/SNMP-MIB$ snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (205894) 0:34:18.94
student@schtlab:~/Cwiczenie/SNMP-MIB$
```

Rysunek 2: Kilkukrotna operacja GET na SysUpTime

Zwracane wartości różnią się, ponieważ były one wykonywane w różnym czasie od uruchomienia serwisu. Jak można zauważyć na Rysunku 2, te operacje w naszym przypadku były wysyłane średnio co sekundę.

3.1.4 5.1.R4

Na początku przy pomocy Wiresharka odczytaliśmy strukturę wiadomości GET-request:

```
User Datagram Protocol, Src Port: 38226, Dst Port: 161
  Source Port: 38226
  Destination Port: 161
  Length: 51
  Checksum: 0xfe46 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 11]
Simple Network Management Protocol
  version: v2c (1)
  community: public
  data: get-request (0)
    get-request
      request-id: 833956362
      error-status: noError (0)
      error-index: 0
      variable-bindings: 1 item
        1.3.6.1.2.1.1.3.0: Value (Null)
          Object Name: 1.3.6.1.2.1.1.3.0 (iso.3.6.1.2.1.1.3.0)
          Value (Null)
```

Ta struktura obejmuje poziom UDP z źródłowym portem 38226 i docelowym portem 161, a także poziom SNMP zawierające wersję v2c, społeczność "public", rodzaj operacji get-request, identyfikator żądania (833956362), status błędu (noError), indeks błędu (0) i jedno związanie zmiennej reprezentujące obiekt o identyfikatorze 1.3.6.1.2.1.1.3.0 z wartością pustą (Null).

Następnie odczytaliśmy strukturę wiadomości GET-response:

```
User Datagram Protocol, Src Port: 161, Dst Port: 38226
  Source Port: 161
  Destination Port: 38226
  Length: 54
  Checksum: 0xfe49 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 11]
Simple Network Management Protocol
  version: v2c (1)
  community: public
  data: get-response (2)
    get-response
      request-id: 833956362
      error-status: noError (0)
      error-index: 0
      variable-bindings: 1 item
        1.3.6.1.2.1.1.3.0: 377168
          Object Name: 1.3.6.1.2.1.1.3.0 (iso.3.6.1.2.1.1.3.0)
          Value (Timeticks): 377168
```

Otrzymaliśmy wiadomość GET-response za pomocą protokołu User Datagram Protocol (UDP) z portu źródłowego 161 i portu docelowego 38226. Struktura tego pakietu obejmuje również poziom SNMP, w którym wersja to v2c, społeczność to "public", a rodzaj operacji to get-response. Dodatkowo, w treści get-response znajduje się identyfikator żądania (833956362), status błędu (noError), indeks błędu (0) i jedno związanie zmiennej reprezentujące obiekt o identyfikatorze 1.3.6.1.2.1.1.3.0 z wartością 377168, przedstawiającą czas w tickach.

3.2 Operacja GET-NEXT

3.2.1 5.2.R1

Zgodnie z instrukcją, zadanie rozpoczęliśmy od uruchomienia dwóch komend

```
$ snmpgetnext -v 2c -c public localhost .1.3.6
$ snmpgetnext -v 2c -c public localhost .1.3.6.1.2.1
```

Poniżej widać rezultaty tych komend.

```
student@schtlab:~/Cwiczenie/SNMP-MIBS$ snmpgetnext -v 2c -c public localhost .1.3.6
iso.3.6.1.2.1.1.1.0 = STRING: "Linux schtlab 5.4.0-150-generic #167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64"
student@schtlab:~/Cwiczenie/SNMP-MIBS$ snmpgetnext -v 2c -c public localhost .1.3.6.1.2.1
iso.3.6.1.2.1.1.1.0 = STRING: "Linux schtlab 5.4.0-150-generic #167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64"
student@schtlab:~/Cwiczenie/SNMP-MIBS$
```

Rysunek 3: Operacje GET-NEXT w terminalu

3.2.2 5.2.R2

Polecenie SNMP `snmpgetnext` służy do pobierania następnego obiektu w drzewie obiektów zarządzanych przez SNMP. Oba polecenia, które podaliśmy, wywołują `snmpgetnext` na różnych początkowych identyfikatorach obiektów:

- `snmpgetnext -v 2c -c public localhost .1.3.6` - Początkowy identyfikator obiektu to `.1.3.6`.
- `snmpgetnext -v 2c -c public localhost .1.3.6.1.2.1` - Początkowy identyfikator obiektu to `.1.3.6.1.2.1`.

W obu przypadkach otrzymaliśmy ten sam wynik, ponieważ `snmpgetnext` zwraca pierwszy następny obiekt, który jest dostępny w danym poddrzewie, niezależnie od tego, jaki był początkowy identyfikator obiektu.

3.2.3 5.2.R3

Przeprowadzając operację GET-NEXT i przekazując uzyskane OID za każdym razem, możliwe jest eksplorowanie kolejnych obiektów w drzewie rejestracji. Taka funkcjonalność pozwala na wypisanie wszystkich obiektów będących w korzeniach danego węzła systemu, co zostało zilustrowane na zrzucie ekranu przedstawiającym serię operacji GET-NEXT prowadzących przez kolejne obiekty tego węzła.

```
student@schtlab:~/Cwiczenie/SNMP-MIBS$ snmpgetnext -v 2c -c public localhost .1.3.6
iso.3.6.1.2.1.1.1.0 = STRING: "Linux schtlab 5.4.0-150-generic #167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64"
student@schtlab:~/Cwiczenie/SNMP-MIBS$ snmpgetnext -v 2c -c public localhost .1.3.6.1.2.1.1.1.0
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.8072.3.2.10
student@schtlab:~/Cwiczenie/SNMP-MIBS$ snmpgetnext -v 2c -c public localhost .1.3.6.1.2.1.1.2.0
iso.3.6.1.2.1.1.3.0 = Timeticks: (530231) 1:28:22.31
student@schtlab:~/Cwiczenie/SNMP-MIBS$ snmpgetnext -v 2c -c public localhost .1.3.6.1.2.1.1.3.0
iso.3.6.1.2.1.1.4.0 = STRING: "juliusz_rafal@mail.pl"
student@schtlab:~/Cwiczenie/SNMP-MIBS$ snmpgetnext -v 2c -c public localhost .1.3.6.1.2.1.1.4.0
iso.3.6.1.2.1.1.5.0 = STRING: "snmpsandbox"
student@schtlab:~/Cwiczenie/SNMP-MIBS$ snmpgetnext -v 2c -c public localhost .1.3.6.1.2.1.1.5.0
iso.3.6.1.2.1.1.6.0 = STRING: "juliusz_rafal"
student@schtlab:~/Cwiczenie/SNMP-MIBS$ snmpgetnext -v 2c -c public localhost .1.3.6.1.2.1.1.6.0
iso.3.6.1.2.1.1.7.0 = INTEGER: 72
student@schtlab:~/Cwiczenie/SNMP-MIBS$
```

Rysunek 4: Operacje GET-NEXT z następującymi OID w terminalu

3.3 Operacja SET

3.3.1 5.3.R1

Na początku wykonaliśmy komendę

```
$ snmpset -v 2c -c private localhost .1.3.6.1.2.1.1.5.0 s laboratorium
```

a następnie wykonaliśmy operację GET na tym samym obiekcie.

```
student@schtlab:~/Cwiczenie/SNMP-MIB$ snmpset -v 2c -c private localhost .1.3.6.1.2.1.1.5.0 s laboratorium
iso.3.6.1.2.1.1.5.0 = STRING: "laboratorium"
student@schtlab:~/Cwiczenie/SNMP-MIB$ snmpget -v 2c -c private localhost .1.3.6.1.2.1.1.5.0
iso.3.6.1.2.1.1.5.0 = STRING: "laboratorium"
student@schtlab:~/Cwiczenie/SNMP-MIB$
```

Rysunek 5: Operacje SET oraz GET w terminalu

3.3.2 5.3.R2

Na początku przy pomocy Wiresharka odczytaliśmy strukturę wiadomości SET-request:

```
Simple Network Management Protocol
  version: v2c (1)
  community: private
  data: set-request (3)
    set-request
      request-id: 124362484
      error-status: noError (0)
      error-index: 0
      variable-bindings: 1 item
        1.3.6.1.2.1.1.5.0: 6c61626f7261746f7269756d
          Object Name: 1.3.6.1.2.1.1.5.0 (iso.3.6.1.2.1.1.5.0)
          Value (OctetString): 6c61626f7261746f7269756d
          Variable-binding-string: laboratorium
```

Wiadomość SET-request w SNMP, oznaczająca operację ustawiania wartości określonego obiektu, została przeprowadzona z wykorzystaniem protokołu SNMP w wersji 2c oraz społeczności "private". Struktura tej wiadomości obejmuje pola: identyfikator żądania (124362484), status błędu (noError), indeks błędu (0) i jedną zmienną wiązania reprezentującą obiekt o identyfikatorze 1.3.6.1.2.1.1.5.0, którego wartość została ustawiona na "laboratorium". Potem odczytaliśmy strukturę wiadomości GET-response:

```
Simple Network Management Protocol
  version: v2c (1)
  community: private
  data: get-response (2)
    get-response
      request-id: 124362484
      error-status: noError (0)
      error-index: 0
      variable-bindings: 1 item
        1.3.6.1.2.1.1.5.0: 6c61626f7261746f7269756d
          Object Name: 1.3.6.1.2.1.1.5.0 (iso.3.6.1.2.1.1.5.0)
          Value (OctetString): 6c61626f7261746f7269756d
          Variable-binding-string: laboratorium
```

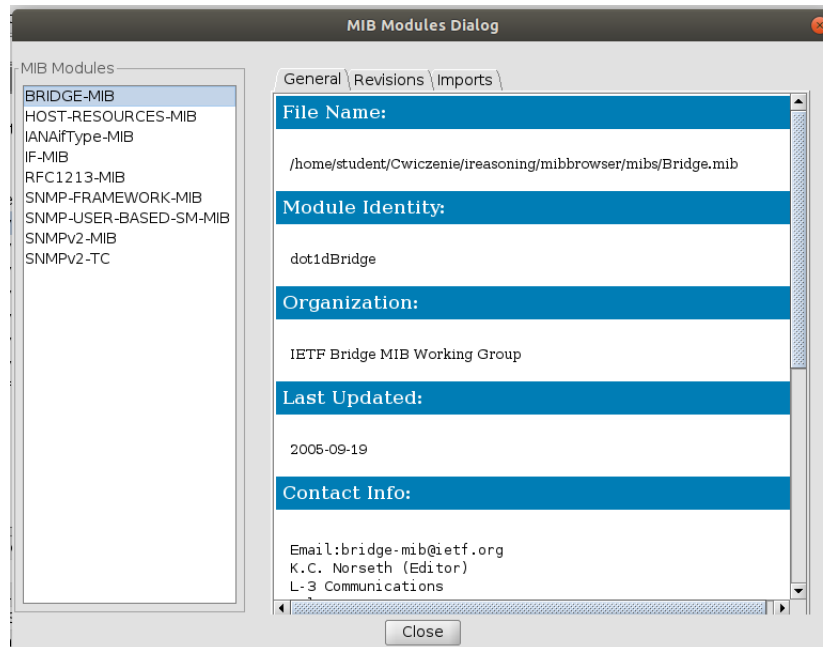
Wiadomość GET-response w SNMP, oznaczająca odpowiedź na operację GET-request, została przesłana z wykorzystaniem protokołu SNMP w wersji 2c oraz społeczności "private". Struktura tej wiadomości obejmuje pola: identyfikator żądania (124362484), status błędu (noError), indeks błędu (0) i jedną zmienną wiązania reprezentującą obiekt o identyfikatorze 1.3.6.1.2.1.1.5.0, którego wartość została pobrana i wynosi "laboratorium".

4 Podstawowe operacje SNMP (interfejs graficzny)

4.1 Zapoznanie się z interfejsem przeglądarki

4.1.1 6.2.R1

Po konfiguracji wstępnej interfejsu oraz wykonaniu szeregu poleceń zrobiliśmy screena okna *mibmodules*.



Rysunek 6: Okno mib modules

Otrzymaliśmy 9 modułów, z czego każdy ma 3 zakładki: General - informacje podstawowe, Revisions - poprawki, Imports - informacje, skąd są importowane dane. Opisy do czego służą wymienione moduły MIB (Management Information Base):

- **BRIDGE-MIB:** Służy do zarządzania urządzeniami obsługującymi IEEE 802.1D.
- **HOST-RESOURCES-MIB:** Służy do zarządzania systemami hostów, czyli dowolnymi urządzeniami, które komunikują się z innymi podobnymi urządzeniami podłączonymi do Internetu i z którego bezpośrednio korzysta jedna lub więcej osób.
- **IANAifType-MIB:** definiuje konwencje tekstowe IANAifType, a co za tym idzie wyliczone wartości obiektu ifType zdefiniowanego w ifTable MIB-II
- **IF-MIB:** Opisuje ogólne obiekty podwarstw interfejsów sieciowych. Ta baza MIB jest zaktualizowaną wersją ifTable MIB-II i zawiera rozszerzenia zdefiniowane w RFC 1229.
- **RFC1213-MIB:** Jest to podstawowy moduł MIB, który zawiera informacje ogólne o systemie, interfejsach sieciowych, statystykach, tabelach trasowania i innych ważnych parametrach.
- **SNMP-FRAMEWORK-MIB:** Architektura zarządzania SNMP Management Architecture MIB.
- **SNMP-USER-BASED-SM-MIB:** Definiuje informacje zarządzania dla SNMP User-based Security Model.
- **SNMPv2-MIB:** Moduł MIB dla podmiotów SNMP.
- **SNMPv2-TC:** Zawiera definicje konwencji tekstowych dla podmiotów SNMP.


```

User Datagram Protocol, Src Port: 43701, Dst Port: 162
  Source Port: 43701
  Destination Port: 162
  Length: 52
  Checksum: 0xfe47 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 192]
Simple Network Management Protocol
  version: version-1 (0)
  community: public
  data: trap (4)
    trap
      enterprise: 1.3.6.1.4.1.8072.3.2.10 (iso.3.6.1.4.1.8072.3.2.10)
      agent-addr: 192.168.0.230
      generic-trap: coldStart (0)
      specific-trap: 0
      time-stamp: 2
      variable-bindings: 0 items




```

Wiadomość trap SNMP, wysłana za pomocą protokołu UDP, zawiera informacje o numerze portu źródłowego (43701) oraz docelowym (162), przy długości pakietu wynoszącej 52 i niezweryfikowanej sumie kontrolnej (0xfe47). W poziomie protokołu SNMP, używając wersji 1 i społeczności "public", treść wiadomości trap zawiera kluczowe informacje: rodzaj trapu to 'trap', numer OID instytucji generującej trap to 1.3.6.1.4.1.8072.3.2.10, adres IP urządzenia generującego trap to 192.168.0.230, ogólny rodzaj trapu to coldStart (0), konkretny rodzaj trapu to 0, znacznik czasowy to 2, a lista zmiennych powiązanych z trape jest pusta (0 items). W tym przypadku, trap reprezentuje zgłoszenie rozpoczęcia pracy systemu (coldStart).

4.3 Obserwacja połączeń TCP

4.3.1 6.4.R1

Po odnalezieniu w drzewie grupy tcp i obiektu tcpConnTable, otworzyliśmy menu i wybraliśmy opcję TableView. Poniżej widoczny jest tego efekt.

Result Table		Trap Receiver		localhost - tcpConnTable ×		
 Rotate	 Refresh	 Export	Poll	SNMP SET	Create Row	Delete Row
tcpConnSt...	tcpConnL...	tcpConnL...	tcpConnR...	tcpConnR...	Index Value	
listen	0.0.0.0	22	0.0.0.0	0	[1] 0.0.0.0.22.0.0.0.0	
listen	127.0.0.1	631	0.0.0.0	0	[2] 127.0.0.1.631.0.0.0.0	
listen	127.0.0.53	53	0.0.0.0	0	[3] 127.0.0.53.53.0.0.0.0	

Rysunek 8: TableView obiektu tcpConnTable

4.3.2 6.4.R2

Tabela zawiera 6 kolumn. Te kolumny opisują:

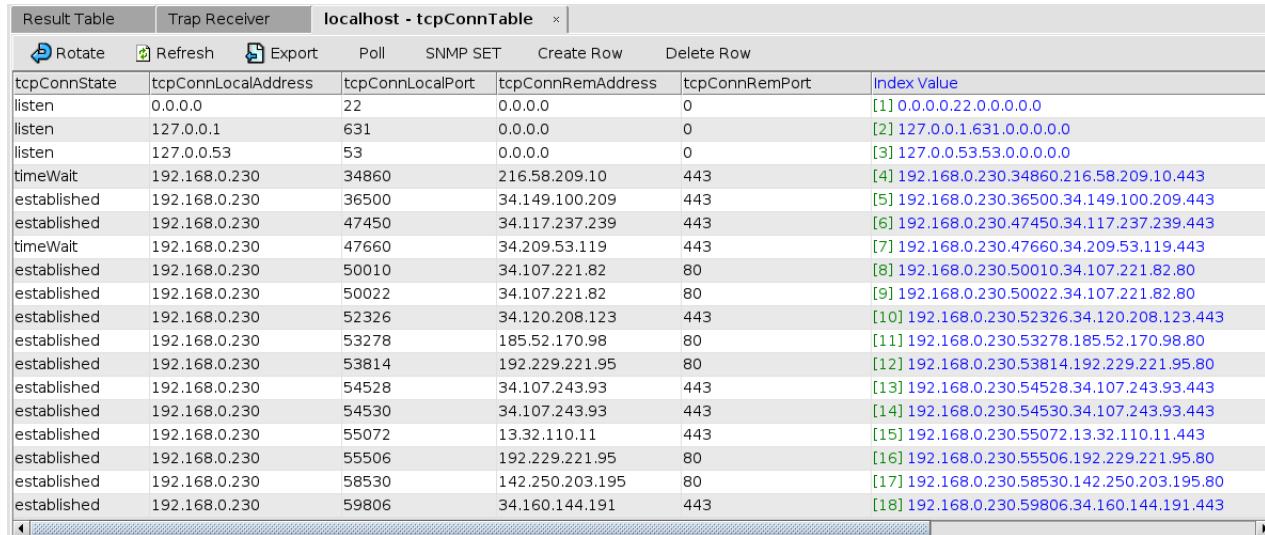
- **tcpConnState:** stan połączenia TCP.
- **tcpConnLocalAddress:** lokalny adres IP połączenia TCP.
- **tcpConnLocalPort:** lokalny numer portu połączenia TCP.
- **tcpConnRemAddress:** zdalny adres IP połączenia TCP.
- **tcpConnRemPort:** zdalny numer portu połączenia TCP
- **Index Value:** wartość, która łączy wartości kolumn tcpConnLocalAddress, tcpConnLocalPort, tcpConnRemAddress, tcpConnRemPort.

4.3.3 6.4.R3

W wiresharku pokazały się operacje GET-NEXT-request na przemian z GET-response. Poprzez iteracyjne używanie operacji "GET-NEXT", przeglądarka jest w stanie odczytać i wyświetlić wszystkie wpisy w tabeli, pomimo braku dedykowanej operacji "GetTable". To podejście opiera się na strukturze drzewa MIB, gdzie każdy obiekt ma unikalny identyfikator, a operacja "GetNext" umożliwia przeglądanie drzewa w porządku leksykograficznym.

4.3.4 6.4.R4

Na samym początku pomyśleliśmy, czy ruch w przeglądarce może stworzyć nowe połączenia TCP. Następnie otworzyliśmy wiele kart w Firefoxie i zgodnie z naszymi przewidywaniami w tabeli pojawiło się znacznie więcej wierszy.



tcpConnState	tcpConnLocalAddress	tcpConnLocalPort	tcpConnRemAddress	tcpConnRemPort	Index Value
listen	0.0.0.0	22	0.0.0.0	0	[1] 0.0.0.0.22.0.0.0.0.0
listen	127.0.0.1	631	0.0.0.0	0	[2] 127.0.0.1.631.0.0.0.0.0
listen	127.0.0.53	53	0.0.0.0	0	[3] 127.0.0.53.53.0.0.0.0.0
timeWait	192.168.0.230	34860	216.58.209.10	443	[4] 192.168.0.230.34860.216.58.209.10.443
established	192.168.0.230	36500	34.149.100.209	443	[5] 192.168.0.230.36500.34.149.100.209.443
established	192.168.0.230	47450	34.117.237.239	443	[6] 192.168.0.230.47450.34.117.237.239.443
timeWait	192.168.0.230	47660	34.209.53.119	443	[7] 192.168.0.230.47660.34.209.53.119.443
established	192.168.0.230	50010	34.107.221.82	80	[8] 192.168.0.230.50010.34.107.221.82.80
established	192.168.0.230	50022	34.107.221.82	80	[9] 192.168.0.230.50022.34.107.221.82.80
established	192.168.0.230	52326	34.120.208.123	443	[10] 192.168.0.230.52326.34.120.208.123.443
established	192.168.0.230	53278	185.52.170.98	80	[11] 192.168.0.230.53278.185.52.170.98.80
established	192.168.0.230	53814	192.229.221.95	80	[12] 192.168.0.230.53814.192.229.221.95.80
established	192.168.0.230	54528	34.107.243.93	443	[13] 192.168.0.230.54528.34.107.243.93.443
established	192.168.0.230	54530	34.107.243.93	443	[14] 192.168.0.230.54530.34.107.243.93.443
established	192.168.0.230	55072	13.32.110.11	443	[15] 192.168.0.230.55072.13.32.110.11.443
established	192.168.0.230	55506	192.229.221.95	80	[16] 192.168.0.230.55506.192.229.221.95.80
established	192.168.0.230	58530	142.250.203.195	80	[17] 192.168.0.230.58530.142.250.203.195.80
established	192.168.0.230	59806	34.160.144.191	443	[18] 192.168.0.230.59806.34.160.144.191.443

Rysunek 9: TableView obiektu tcpConnTable po odpaleniu wielu kart w przeglądarce

Przeglądarki internetowe często nawiązują wiele połączeń jednocześnie, na przykład dla otwartych kart, do wczytywania zasobów strony internetowej, skryptów, czy obrazów. Każde z tych połączeń może być reprezentowane przez wpis w tabeli połączeń TCP.

4.4 Obserwacja interfejsów sieciowych hosta

4.4.1 6.5.R1)

Po odnalezieniu grupy interfaces, znaleźliśmy obiekt ifNumber, który określa liczbę interfejsów sieciowych (niezależnie od ich aktualnego stanu) obecnych w tym systemie. Przechowuje on wartość 2.

4.4.2 6.5.R2

Po otwarciu menu na węźle ifTable, wybraliśmy opcję TableView i 6 pierwszych kolumn tej tabeli prezentowało się następująco.

ifIndex	ifDescr	ifType	ifMtu	ifSpeed	ifPhysAddress
1	lo	softwareLoo...	65536	10000000	
2	Intel Corpor...	ethernetCs...	1500	1000000000	08-00-27-A6-45-C6

Rysunek 10: 6 pierwszych kolumn TableView ifTable

Opis poszczególnych kolumn:

- **ifIndex:** numer identyfikacyjny interfejsu. Jest to unikalny identyfikator dla każdego interfejsu w tabeli.

- **ifDescr:** opis interfejsu. Zawiera ludzko-czytelną nazwę lub opis interfejsu, co ułatwia identyfikację.
- **ifType:** typ interfejsu. Określa rodzaj interfejsu, np. Ethernet, software loopback. Numer typu interfejsu jest zdefiniowany w standardzie MIB.
- **ifMtu:** maksymalna jednostka transmisji (Maximum Transmission Unit) dla interfejsu. Określa największy rozmiar pakietu, jaki może być przesłany przez interfejs bez fragmentacji.
- **ifSpeed:** prędkość interfejsu. Określa maksymalną prędkość transmisji w bitach na sekundę.
- **ifPhysAddress:** adres fizyczny interfejsu, podany na przykład w formie szesnastkowej.

4.4.3 6.5.R3

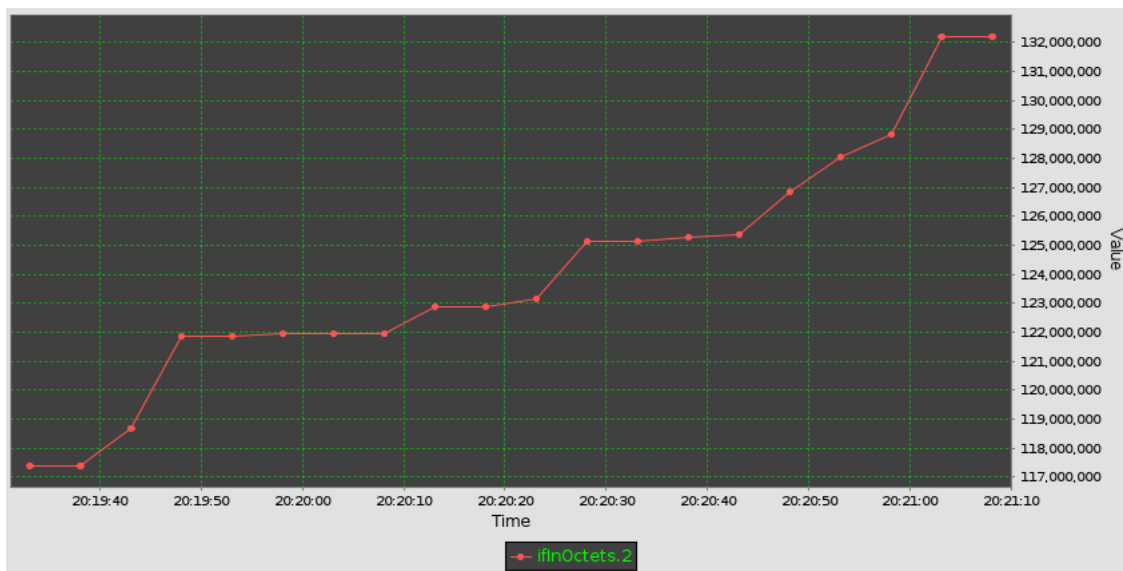
Definicje następujących obiektów brzmią:

- **ifAdminStatus:** pożądaný stan interfejsu.
- **ifOperStatus:** bieżący stan operacyjny interfejsu.
- **ifInOctets:** całkowita liczba oktetów odebranych przez interfejs, łącznie ze znakami ramek.

W skrócie, te obiekty zapewniają informacje o pożądanym stanie administracyjnym interfejsu, aktualnym stanie operacyjnym oraz ilości odebranych danych przez interfejs. Są one przydatne w monitorowaniu i zarządzaniu siecią, umożliwiając administratorom kontrolę nad stanem i wydajnością poszczególnych interfejsów sieciowych.

4.4.4 6.5.R4

Po wybraniu obiektu ifInOctets (interfejs nie loopback), otworzyliśmy menu i wybraliśmy opcję GraphView. Następnie, aby spowodować, żeby na wykresie były widoczne jakieś istotne zmiany postanowiliśmy odpalić przeglądarkę Firefox oraz zaczęliśmy wchodzić na różne strony internetowe. Przy każdym wejściu na nową stronę internetową wartości na wykresie rosły, a gdy pozostowaliśmy na tej stronie nie zmieniały się. Na samym końcu przy zamknięciu przeglądarki wartość value była stała.



Rysunek 11: Wykres zależności wartości od czasu obiektu ifInOctets

5 Wnioski

W trakcie laboratorium, mającego na celu zapoznanie się z podstawowymi zasadami interakcji z agentem SNMP (Simple Network Management Protocol), przeprowadziliśmy szereg praktycznych operacji zarówno w interfejsie CLI, jak i graficznym. Wykonaliśmy podstawowe operacje SNMP, takie jak GET, GET-NEXT i SET, co umożliwiło nam zrozumienie procesu komunikacji między menedżerem a agentem podczas pobierania, pobierania kolejnego i ustawiania danych. Dodatkowo, korzystając z interfejsu przeglądarki SNMP, zdobyliśmy umiejętność intuicyjnego konfigurowania i monitorowania urządzeń, a operacja NOTIFY pozwoliła nam zaznajomić się z powiadomieniami o zdarzeniach.

W trakcie laboratorium skupiliśmy się także na obserwacji połączeń TCP pomiędzy agentem a menedżerem, co umożliwiło nam lepsze zrozumienie procesu zestawiania i utrzymywania komunikacji SNMP. Dodatkowo, analizując interfejsy sieciowe hosta, uzyskaliśmy praktyczne doświadczenie w monitorowaniu danych dostępnych za pomocą SNMP oraz zrozumieniu, jakie informacje są udostępniane w kontekście zarządzania siecią.

W efekcie, laboratorium umożliwiło nam skuteczne wykorzystanie operacji SNMP w praktyce, integrując wiedzę z zakresu interfejsu CLI, interfejsu graficznego, obserwacji połączeń TCP i monitorowania interfejsów sieciowych. Zdobytą wiedzę możemy teraz efektywnie stosować w rzeczywistych scenariuszach administrowania siecią, co stanowi zgodność z głównym celem tego laboratorium.