Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Marcin Semeniuk
Nr albumu: 214707

Monitorowanie parametrów pracy urządzeń obsługujących SNMP

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Janiny Mincer-Daszkiewicz
Instytut Informatyki

Listopad 2006

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

Streszczenie

W pracy przedstawiono projekt i implementację nowego narzędzia służącego do monitorowania parametrów urządzeń sieciowych. W odróżnieniu od istniejących rozwiązań jest ono zarówno szybkie, jak i posiada dodatkowe funkcje, dzięki czemu umożliwia kompleksową obsługę dużych sieci. Zaprezentowano również wyniki testów wydajnościowych. Opisano wdrożenie nowego narzędzia w Dziale Sieciowym ICM, gdzie jest ono wykorzystywane do monitorowania sieci szkieletowej UW.

Słowa kluczowe

monitorowanie, urządzenie sieciowe, SNMP, Round Robin Database

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

C. Computer Systems Organization
C.2 Computer-Communication Networks
C.2.3 Network Operations
C.2.4 Distributed Systems

Tytuł pracy w języku angielskim

Monitoring of devices using SNMP

Spis treści

Wprowadzenie

Internet od czasu swojego powstania w latach 60 ciągle się rozwija. Na początku było to tylko kilka uczelnianych sieci połączonych ze sobą, ale z czasem do Internetu zaczęły dołączać sieci korporacyjne. Wiele osób zaczęło polegać na możliwościach jakie zapewnia Internet, dlatego pojawiła się potrzeba zapewniania satysfakcjonującej wydajności i możliwości szybkiego reagowania na problemy. Dzięki monitorowaniu urządzeń sieciowych można łatwo wykrywać awarie i miejsca, w których tworzą się wąskie gardła w sieci. Znaczenie monitorowania rośnie wraz ze wzrostem liczby urządzeń, z których zbudowana jest sieć. Ponieważ Internet jest siecią niejednorodną, pojawiła się potrzeba zapewnienia spójnych mechanizmów zarządzania w sieciach o różnorodnej budowie. Na początku 1988 roku zostały opublikowane wymagania, które powinien spełniać protokół, aby mógł stać się standardem. Zgodnie z tymi wymaganiami powinien być możliwie rozbudowany, posiadać możliwości różnorodnej implementacji i być w stanie zarządzać jak największą liczbą warstw protokołów. SNMP (ang. *Simple Network Management Protocol*, por. p. 1.1) został zaprojektowany tak, żeby spełniać te wymagania i szybko stał się standardowym protokołem służącym do monitorowania urządzeń sieciowych.

Sieć szkieletowa Uniwersytetu Warszawskiego jest dość rozbudowana, bo składa się z ponad 80 urządzeń, które trzeba monitorować. Na początku wykorzystywany był do tego celu system zbudowany na bazie programu MRTG (ang. *Multi Router Traffic Grapher*, por. p. 1.4.1). Pierwotnie urządzenia były odpytywane sekwencyjnie, a czas między kolejnymi cyklami wynosił 5 minut. Niestety powodowało to problemy w przypadku kiedy odpytanie jakiegoś urządzenia trwało dużej niż powinno (np. ponieważ urządzenie nie odpowiadało), pozostałe urządzenia były wtedy odpytywane z opóźnieniem. Czasami dochodziło do sytuacji, w której odpytanie wszystkich urządzeń trwało dłużej niż wspomniane wcześniej 5 minut, w efekcie czego niektórych próbek danych nie udawało się zebrać. Dlatego podjęto decyzję o modyfikacji systemu, w efekcie której każde urządzenie było odpytywane przez osobną instancję MRTG. Problemy powodowane przez pojedyncze urządzenia przestały wpływać na zachowanie całego systemu monitorowania. Niestety narzut związany z koniecznością interpretowania kodu kilkunastu instancji MRTG spowodował wysokie obciążenie serwera (ang. *load*). Konieczne stało się stworzenie nowego systemu monitoringu.

Istnieją na rynku rozwiązania komercyjne, które powinny być w stanie sobie poradzić z taką liczbą urządzeń, ale niestety rozwiązania te nie udostępniają kodu źródłowego, przez co nie są możliwe jakiekolwiek modyfikacje mające na celu dostosowanie programu do indywidualnych potrzeb. Dlatego poszukiwania nowego programu, na bazie którego miał być stworzony nowy system monitoringu zostały ograniczone do rozwiązań z dostępnym kodem źródłowym (ang. *open source*). Ponieważ żaden ze znalezionych programów nie spełniał wszystkich wymagań, a dodanie potrzebnej funkcjonalności wiązałoby się ze znacznymi zmianami, zdecydowano, że najlepiej będzie napisać nowe narzędzie.

W niniejszej pracy przedstawię program, nazwany MGR, który napisałem na potrzeby Działu Sieciowego Interdyscyplinarnego Centrum Modelowania Matematycznego i Komputerowego (ICM) [1], który jest odpowiedzialny za utrzymywanie sieci szkieletowej Uniwersytetu

Warszawskiego.

Praca składa się z 7 rozdziałów i 3 dodatków. W rozdziale 1 przedstawiono zagadnienie monitorowania urządzeń sieciowych, wraz z istniejącymi narzędziami wykorzystywanymi do tego celu. Rozdział 2 poświęcono wymaganiom jakie są stawiane nowemu programowi. Projekt nowego narzędzia został przedstawiony w rozdziale 3, a jego implementacja została opisana w rozdziale 4. Wdrożenie i testy przedstawiono odpowiednio w rozdziałach 5 i 6. Rozdział 7 stanowi podsumowanie niniejszej pracy.

W dodatkach umieszczono przykładowy plik konfiguracyjny programu i opis zawartości dołączonej plyty.

Rozdział 1

Monitorowanie urządzeń sieciowych

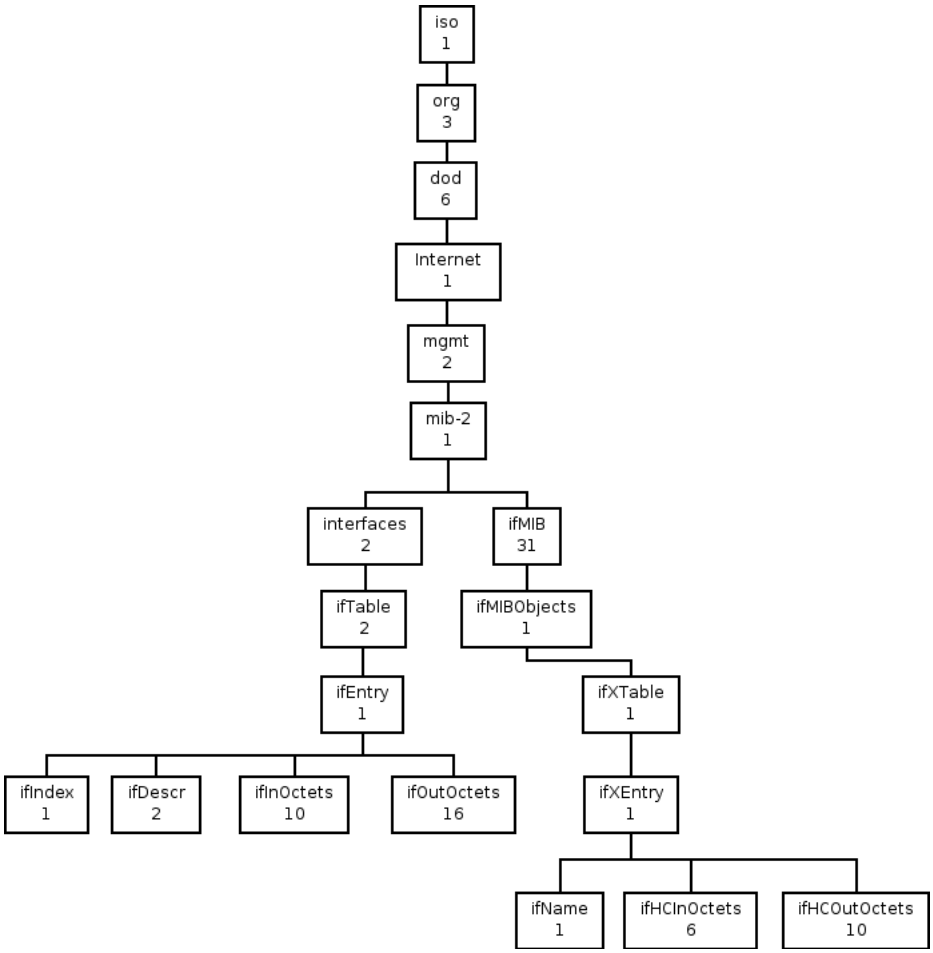
Do zrozumienia niektórych wymagań stawianych programowi i rozwiązań w nim zastosowanych, przydatna może się okazać podstawowa znajomość protokołu SNMP czy formatu RRD.

1.1. SNMP

Protokół SNMP (ang. *Simple Network Management Protocol*) [11] zakłada istnienie dwóch kategorii urządzeń, zarządzanych i zarządzających. Na urządzeniach zarządzanych uruchomiony jest agent SNMP, podczas gdy na urządzeniach zarządzających menedżer SNMP. Agent SNMP przechowuje informacje o stanie urządzenia (np. różnego rodzaju statystyki). Zarządzanie polega na tym, że menedżer SNMP odczytuje lub modyfikuje odpowiednie zmienne. Agent może poinformować menedżera o wystąpieniu nieprzewidzianego zdarzenia, wysyłając specjalny komunikat zwany pułapką, który zawiera informacje o zdarzeniu.

Dla zapewnienia sprawnego dostępu do informacji, konieczne okazało się usystematyzowanie nazw pod jakimi są one zapisane. W SNMP wykorzystano do tego celu MIB (ang. *Management Information Base*). Każdy obiekt w bazie posiada przyporządkowaną nazwę, wartość, typ, opis zawierający szczegółowe informacje potrzebne do prawidłowej implementacji i zestaw operacji jakie można wykonać na tym obiekcie (odczytanie wartości, zapisanie wartości). Obiekty przechowane są jako liście w drzewiastej strukturze i żeby uzyskać dostęp do obiektu trzeba podać oddzielane kropkami nazwy wszystkich węzłów od korzenia aż do liścia. Fragment takiego drzewa przedstawiony jest na rysunku 1.1. Węzeł jest identyfikowany przez numer i opisową nazwę, które mogą być stosowane zamiennie. Tak więc obiekt może mieć nazwę .iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInOctets lub .1.3.6.1.2.1.2.2.1.10 lub .1.3.6.1.2.1.interfaces.ifTable.ifEntry.ifInOctets i wciąż będzie to ten sam obiekt. Najważniejsze z punktu widzenia monitorowania urządzeń sieciowych obiekty zdefiniowane są w podzbiorze MIB nazwanym IF-MIB [4] Generalnie za przydzielanie nazw i numerów węzłom odpowiada ISO (ang. *International Organization for Standardization*), ale istnieją poddrzewa, w których ISO może utworzyć nowy węzeł i przekazać zarządzanie nim firmie lub organizacji. Wtedy taka firma lub organizacja sama tworzy sobie strukturę, nadaje nazwy i numery. Dzięki takiemu podejściu zapewnione jest unikatowe nazewnictwo obiektów, a całość jest bardzo elastyczna i umożliwia łatwe rozbudowywanie bez konieczności zmian w samym protokole.

Zmienne są lokalnymi instancjami obiektów. Może być kilka instancji tego samego obiektu (np. wartość zmiennej dla różnych interfejsów sieciowych), dlatego odwołując się do zmiennej podajemy nazwę obiektu, a potem po kropce indeks zmiennej. Jeśli obiekt może mieć tylko



Rysunek 1.1: Fragment Drzewa MIB

jedną instancję, to jako indeks podaje się 0.

Do komunikacji między agentem a menedżerem wykorzystywany jest protokół UDP (ang. *User Datagram Protocol*). UDP jest prostym bezpołączeniowym protokołem, dzięki czemu obciąża sieć w mniejszym stopniu niż protokoły połączeniowe (np. TCP). Poza tym świetnie nadaje się do prostych transmisji żądanie-odpowiedź, z jakich korzysta SNMP.

Pakiet SNMP zawiera dwa pola, nagłówek i PDU (ang. *Protocol Data Unit*). Nagłówek składa się z dwóch pól. Jedno z nich zawiera wersję protokołu, w drugim zapisane jest *community* — hasło wysyłane jawnym tekstem, na podstawie którego agent decyduje czy pozwolić na odczytywanie bądź modyfikacje zmiennych. PDU z żądaniem bądź odpowiedzią zawiera:

- typ PDU — SNMP w wersji pierwszej definiuje 4 typy PDU:
  - Get — żądanie odczytu wartości zmiennej; może zawierać kilka identyfikatorów zmiennych, wtedy o ile nie wystąpi błąd, to odpowiedź będzie zawierać wartości wszystkich zmiennych.
  - GetNext — żądanie odczytu następnej zmiennej i jej wartości,
  - Set — żądanie ustawienia wartości zmiennej,
  - Response — odpowiedź na żądanie;
- id żądania — jednoznacznie identyfikuje, na które pytanie przyszła odpowiedź;

- status błędu — informacja czy wystąpił błąd;
- indeks błędu — informacja, przy której zmiennej wystąpił błąd;
- właściwe informacje — pary identyfikator zmiennej i wartość. W przypadku żądania odczytu wartość jest ustawiana na 0 lub NULL.

Nie ma sztywnego ograniczenia na liczbę obiektów, których wartości się odczytuje bądź modyfikuje, natomiast jest ograniczenie na rozmiar PDU, które musi się mieścić w pojedynczym datagramie UDP. Wszystkie urządzenia powinny obsługiwać PDU o rozmiarze co najmniej 484 bajtów.

Ze względu na ograniczenia jakie posiada SNMP w wersji pierwszej, stworzona została wersja druga, SNMP v2. Oferuje ona kilka istotnych ulepszeń:

- poprawiony komunikat Get — jeśli wystąpił błąd, to odpowiedź będzie zawierać wartości zmiennych, w których błąd nie wystąpił;
- nowy komunikat GetBulk — żądanie odczytu wartości kilku kolejnych zmiennych. Główną korzyścią w stosunku do Get z podanymi kilkoma wartościami jest zachowanie się w momencie, w którym agent nie będzie w stanie przetworzyć żądania ze względu na zbyt duży jego rozmiar. W takim przypadku Get przekaże błąd, a GetBulk przekaże tyle wartości ile się zmieści w odpowiedzi. Kolejną zaletą jest możliwość pobierania wartości kilku kolejnych zmiennych, których nazwy się nie zna, w jednym żądaniu. Wykorzystanie do tego celu GetNext wymagałoby wysyłania żądania dla każdej zmiennej.
- wprowadzono 64-bitowe liczniki — w wersji pierwszej protokołu wartości liczników były przechowywane jako 32-bitowe liczby, w przypadku interfejsów pracujących z prędkościami ponad 100 Mb/s pojawiała się możliwość, że liczba przesłanych bajtów, między dwoma kolejnymi pytaniami, będzie większa niż największa 32-bitowa liczba. Ponieważ z punktu widzenia oprogramowania monitorującego sytuacja, w której przesłano 1 bajt i sytuacja w której przesłano $2^{32} + 1$ bajtów wyglądają tak samo, pojawia się ryzyko błędnej interpretacji wyników. Dzięki wykorzystaniu 64-bitowych liczb jako liczników, przy obecnie stosowanych prędkościach, takie ryzyko praktycznie nie występuje.

Wspomnieć tutaj należy jeszcze, że istnieje też wersja trzecia SNMP, która zapewnia dużo bardziej rozbudowane mechanizmy uwierzytelniania, ale nie jest jeszcze powszechnie wykorzystywana.

1.2. RRD

W cyklicznej bazie danych do przechowywania danych wykorzystywana jest stała ilość miejsca. Podczas zapisywania, najstarsze wartości są zastępowane nowymi. Takie podejście sprawia, że można przechowywać tylko określoną ilość danych (np. z ostatniego dnia). W sytuacji gdy interesują nas dane z dłuższego okresu, mamy 2 możliwości: albo zwiększyć rozmiar bazy (ale takie rozwiązanie sprawia, że dane zajmowałyby dużo miejsca), albo wykorzystać konsolidację danych uwzględniając fakt, że informacje archiwalne nie muszą mieć takiej dokładności jak bieżące. Na przykład na podstawie dokładnych danych z ostatniego dnia (próbka danych co 5 minut), wyliczamy uśrednione dane (próbka co godzinę) i takie uśrednione dane przechowujemy przez tydzień (168 próbek), dzięki temu dane za ostatni tydzień zajmują 12 razy mniej miejsca niż gdybyśmy przechowywali dokładne dane.

RRD (ang. *Round Robin Database*) [10] jest formatem plików przeznaczonym do utrzymywania cyklicznej bazy danych. Wywodzi się z MRTG (por. p. 1.4.1) i pierwotnie miał być alternatywą dla starego tekstowego, przez co mało wydajnego, formatu przechowywania zebranych informacji. Jednakże dzięki dużym możliwościom konfiguracyjnym, automatycznemu przeprowadzaniu konsolidacji danych na podstawie dowolnie definiowanych reguł i łatwemu tworzeniu wykresów z zapisanych danych szybko stał się swoistym standardem do przechowania cyklicznych baz danych.

1.3. Podstawowe pojęcia

`Urządzenie` — W niniejszym dokumencie ilekroć odwołuję się do terminu urządzenie, mam na myśli ruter, zarządzalny przełącznik (ang. *switch*) lub komputer, który może być monitorowany z wykorzystaniem protokołu SNMP.
`OID` — Identyfikator węzła przechowującego informacje w MIB.
`Cel` — Zbiór OID, których wartości są zapisywane do jednego pliku RRD (np. dane przychodzące i wychodzące na określonym interfejsie).

1.4. Istniejące rozwiązania

Z dostępnych na rynku programów o otwartym kodzie, służących do monitorowania urządzeń z wykorzystaniem protokołu SNMP warto wymienić dwa.

1.4.1. MRTG

MRTG (ang. *Multi Router Traffic Grapher*) [5] składa się z dwóch części: skryptu napisanego w Perlu służącego do komunikacji z urządzeniami i programu napisanego w C, który dokonuje aktualizacji pliku z wynikami.

MRTG cieszy się dużą popularnością, dzięki czemu powstało do niego sporo dodatkowych programów. Przeważnie są to alternatywne interfejsy użytkownika, które wyświetlają strony WWW z wykresami, wygenerowanymi przez MRTG. Konfiguracja może być rozszerzana poprzez dodanie do niej nowych opcji, które nie wpływają na działanie samego MRTG. Dzięki temu inne programy mogą przechowywać swoją konfigurację w tym samym pliku co MRTG. Takie rozwiązanie jest wygodne ponieważ w razie jakichkolwiek zmian trzeba modyfikować tylko jeden plik. MRTG może wykorzystywać RRD jako alternatywny sposób zapisu danych.

Niestety MRTG nie jest pozbawione wad. Najpoważniejszą z nich, która ujawnia się dopiero w przypadku dużo bardziej rozbudowanych instalacji, jest słaba wydajność spowodowana koniecznością interpretowania kodu napisanego w Perlu. Innym ograniczeniem, które bezpośrednio wynika z przyjętego formatu zapisu danych jest to, że cel zawsze musi składać się z dwóch OID. Mimo że alternatywny format (RRD) nie posiada takiego ograniczenia, MRTG nie jest w stanie tego wykorzystać. Jak już zostało wspomniane w rozdziale 1.1, składowe elementy identyfikatora obiektu mogą być liczbą lub nazwą, niestety wyszukiwanie powtarzających się OID wykonywane jest poprzez proste sprawdzenie czy tekstowa reprezentacja dwóch OID jest taka sama, z tego powodu może się zdarzyć ze MRTG będzie wysyłać wiele pytań o te same obiekty.

1.4.2. RTG

RTG (ang. *Real Traffic Grabber*) [9] został napisany dlatego, że wydajność MRTG okazała się zbyt niska. Twórcy skupili się na szybkości działania, dlatego program został napisany w języku C. Wyniki są wprowadzane do bazy danych, dzięki czemu możliwe stało się przechowywanie faktycznych danych, bez żadnego uśredniania, co może się przydawać do dokładnego rozliczania klientów za faktyczny transfer danych. Oczywistą wadą takiego podejścia jest rosnący rozmiar bazy danych, w miarę zapisywania kolejnych wyników. Inną konsekwencją jest to, że gdy chcemy przechowywać uśrednione wyniki, to trzeba samemu napisać własne funkcje, które będą dokonywały konsolidacji danych.

W przypadku stosunkowo małych sieci, dla których wydajność MRTG nie stanowi problemu, RTG jest raczej rzadko stosowany. Dużo łatwiej i szybciej jest zainstalować MRTG niż RTG, gdyż w przypadku RTG trzeba dodatkowo skonfigurować serwer bazy danych i ewentualnie pisać funkcje uśredniające. Dodatkowym czynnikiem ograniczającym popularność RTG w małych sieciach jest mała dostępność gotowych narzędzi korzystających z RTG i fakt, że RTG służy tylko do odpytywania urządzeń i zapisywania wyników do bazy danych. Brak w nim dodatkowych funkcji, które udostępnia MRTG, jak np. wykonywanie operacji arytmetycznych na wynikach jeszcze przed zapisaniem ich do bazy, wykonywanie skryptów czy możliwość wyszukiwania interfejsów. Dodatkowo po każdej zmianie konfiguracji urządzenia konieczne jest uruchomienie skryptu `rtgtargmkr.pl`, który tworzy nową listę obiektów do monitorowania, w przeciwnym wypadku RTG nie będzie świadome zmiany i wyniki będą przypisane do złego interfejsu.

1.5. Podsumowanie

Sieć szkieletowa UW jest bardzo rozbudowana i system monitorowania musi być efektywny, zarówno pod względem wydajności, jak i zarządzania konfiguracją. Żadne z dostępnych rozwiązań nie spełnia wymagań stawianych programowi do monitorowania. MRTG jest za wolny, a RTG mimo swojej wysokiej wydajności jest bardzo ubogi funkcjonalnie, przez co utrzymywanie systemu opartego na nim wiązałoby się z koniecznością ręcznego modyfikowania bazy danych po każdej drobnej zmianie w konfiguracji urządzeń.

Dlatego podjąłem się zaprojektowania i wykonania nowego programu opisanego w kolejnych rozdziałach pracy.

Rozdział 2

Wymagania

Wymagania stawiane programowi do monitorowania sieci ulegały zmianom w trakcie implementacji, jednakże ogólne założenia, opisane w tym rozdziale, pozostały prawie niezmienione.

2.1. Wymagania funkcjonalne

1. Niezależne odpytywanie urządzeń
   Odpytywanie pewnych urządzeń może trwać długo. Nie może to powodować, że kolejne urządzenia będą odpytywane z opóźnieniem.

2. Efektywne odpytywanie
   Niektóre urządzenia obsługują protokół SNMP w wersji drugiej. Podczas odpytywania takich urządzeń powinno być możliwe skorzystanie z ulepszeń wprowadzonych w drugiej wersji protokołu SNMP, w celu skrócenia czasu potrzebnego na odpytanie urządzenia. Dodatkowo należy unikać wysyłania kilku pytań o wartość tego samego OID w jednym cyklu.

3. Wyszukiwanie
   Ponieważ numery interfejsów sieciowych urządzenia mogą się dynamicznie zmieniać, monitorowanie ich wyłącznie po numerach mogłoby prowadzić do sytuacji, w której w jednym pliku posiadamy dane zebrane z różnych interfejsów. Z każdym interfejsem powiązane są dodatkowe informacje udostępniane przez SNMP, dlatego należy umożliwić wyszukiwanie interfejsów po tych informacjach. Takimi informacjami są : nazwy, opisy, adresy. Dobrze by było dodatkowo umożliwić wyszukiwanie po występujących w MIB dodatkowych kluczach wskazanych przez użytkownika.

4. Pamięć podręczna
   Wartości niektórych OID zmieniają się rzadko, rozsądne wydaje się więc przechowywanie tych wartości w pamięci podręcznej. Dzięki temu, na przykład, nie trzeba będzie ponownie wyszukiwać interfejsu w każdym cyklu.

5. Operacje matematyczne na wynikach
   Czasem zdarza się, że urządzenie w odpowiedzi przekazuje dane w innej postaci niż ta, która nas interesuje, na przykład bity zamiast bajtów. Program powinien udostępniać możliwość automatycznego wykonywania zadanych operacji matematycznych na wynikach otrzymanych z urządzenia.

6. Wykorzystanie RRD
   Dotychczas stosowany program do przechowywania zebranych wyników wykorzystywał format RRD. Został on zaprojektowany do efektywnego przechowywania informacji otrzymywanych w stałych odstępach czasu. Dobrze by było żeby program korzystał z tego formatu, co umożliwi wykorzystanie już istniejących plików z danymi bez potrzeby konwersji.

7. Możliwość zapisywania dowolnej liczby serii do jednego pliku
   Dotychczasowy program posiadał ograniczenie, że do każdego pliku zapisywał informacje z dwóch serii. Jeśli zachodziła potrzeba zapisania jednej serii, to trzeba było wprowadzić tę serię dwa razy. Natomiast w przypadku liczby serii większej od dwóch, trzeba było zakładać kilka plików i dopiero podczas tworzenia wykresu łączyć dane z kilku plików. Program nie powinien nakładać podobnego ograniczenia, natomiast powinien umożliwiać użytkownikowi zdefiniowanie, w pliku konfiguracyjnym, specyfikacji, na podstawie której będą tworzone pliki RRD.

8. Możliwość wykonywania zewnętrznych skryptów
   Program powinien umożliwiać pobieranie danych z zewnętrznych skryptów. Dodatkowo powinna istnieć możliwość uruchomienia skryptu przed i po odpytaniu urządzenia.

9. Logowanie informacji o zdarzeniach
   Program powinien umożliwiać zapisywanie do dziennika systemowego (ang. *syslog*) informacji o problemach z monitorowanymi urządzeniami, opóźnieniach w transmisji i innych niespodziewanych zdarzeniach. Informacje powinny być zapisywane z różnymi priorytetami, w zależności od stopnia ważności zdarzenia.

2.2. Wymagania niefunkcjonalne

1. Wydajność
   Powinna być możliwie jak największa. W praktyce wystarczy, że program będzie w stanie odpytać 50 urządzeń, każde zawierające 200 OID, w czasie krótszym niż 5 minut.

2. Wielowątkowość
   Pozwoli na równoległe odpytywanie wielu urządzeń i umożliwi wykorzystywanie kilku procesorów.

3. Rozdzielenie części zbierającej informacje od zapisującej
   Dzięki takiemu podejściu możliwe będzie wykorzystanie kilku komputerów do zbierania danych, a wszystkie zebrane dane będą zapisywane na wydzielonym komputerze. W razie potrzeby będzie można zastąpić moduł zapisujący innym, żeby zapisywał dane na przykład do bazy danych.

Rozdział 3

Projekt

3.1. Podział ze względu na funkcje

Program, ze względu na pełnione funkcje, można podzielić na dwa moduły:

- **moduł odpytujący** (ang. *poller*)
  Część programu odpowiedzialna za komunikację z urządzeniami, otrzymane wyniki może zapisywać lokalnie lub przekazywać do zdalnego modułu zapisującego.

- **moduł zapisujący** (ang. *writer*)
  Funkcjonalna część programu, która może być zrealizowana jako osobny program, odpowiedzialna za zapisywanie otrzymanych danych.

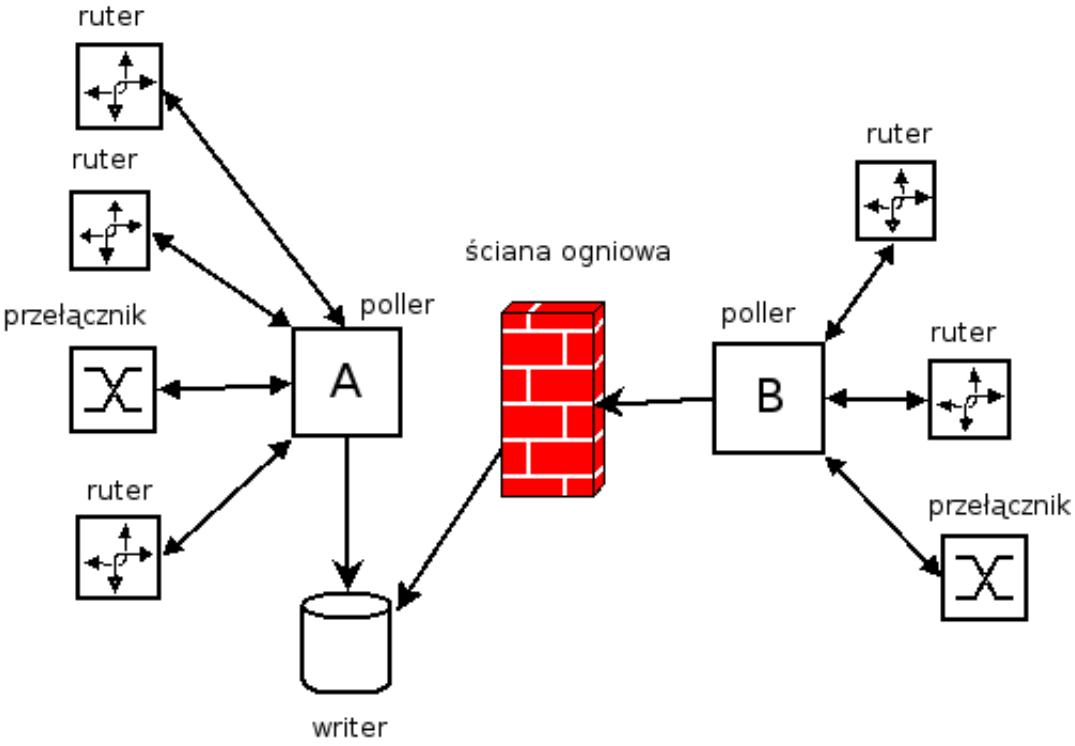
3.2. Architektura systemu

Cały system monitorowania składa się z dużej liczby (kilkadziesiąt) urządzeń i z co najmniej jednego komputera z zainstalowanym modułem odpytującym. Opcjonalnie w systemie mogą istnieć dodatkowe komputery z zainstalowanym wyłącznie modułem zapisującym. Dzięki istnieniu dowolnej liczby modułów odpytujących mamy możliwość monitorowania urządzeń znajdujących się w różnych częściach sieci, do których nie ma bezpośredniego dostępu ze względu na wykorzystanie ścian ogniowych (ang. *firewall*). Można też wykorzystać kilka modułów zapisujących do zapisywania tych samych wyników na kilku komputerach na raz. Na rysunku 3.1 widzimy przykładowy system. Działają w nim dwa komputery z uruchomionym modułem odpytującym (oznaczone jako A i B) i jeden z modułem zapisującym. Komputer B znajduje się za ścianą ogniową i monitoruje urządzenia, których komputer A nie może monitorować. Wszystkie zebrane dane są zapisywane w jednym miejscu.

3.3. Podział na moduły

Program będzie się składał z kilku modułów. Na rysunku 3.2 przedstawiony jest podział na moduły.

1. Wczytywanie konfiguracji
   Moduł ten jest odpowiedzialny za wczytywanie konfiguracji i sprawdzenie jej poprawności.

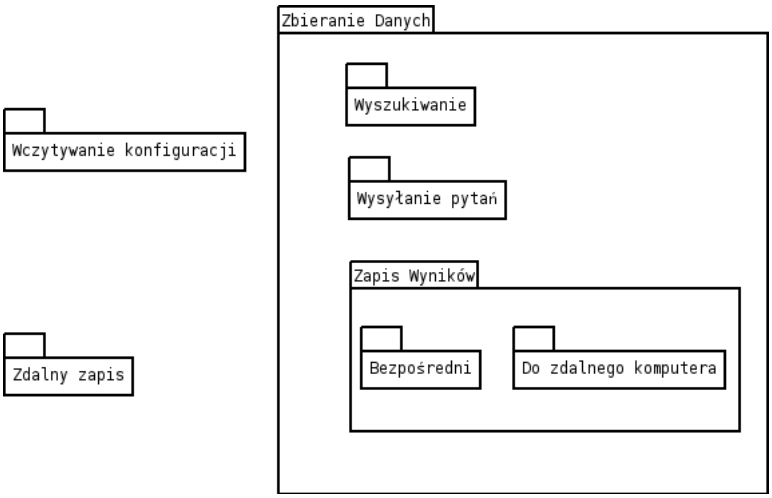


Rysunek 3.1: Architektura systemu

2. Zbieranie danych
   Moduł ten jest odpowiedzialny za cykliczne odpytywanie urządzeń. Rysunek 3.3 przedstawia cykl działania modułu odpowiedzialnego za zbieranie danych. Najpierw wyszukiwane są numery interfejsów o określonych nazwach czy opisach, następnie wysyłane są pytania. Wątek czeka aż odpowiedzi na wszystkie wysłane pytania dotrą z powrotem, a w razie potrzeby ponownie wysyła pytania. Kiedy już wszystkie odpowiedzi zostaną odebrane, następuje faza zapisywania informacji (zdalnie lub lokalnie, zależnie od konfiguracji celu). Po zakończeniu zapisywania wątek czeka na następny cykl.

   - Wyszukiwanie
     Moduł realizujący funkcje wyszukiwania interfejsów, wysyła synchroniczne pytania do urządzenia i zapamiętuje numery interfejsów, które zawierały odpowiednie nazwy/opisy/adresy. Dzięki temu przy kolejnych pytaniach nie trzeba wyszukiwać, a jedynie upewniać się, że nazwa/opis/adres przypisany do interfejsu się nie zmienił.
   - Wysyłanie pytań
     Moduł odpowiedzialny za wysyłanie pytań do urządzeń i odbieranie odpowiedzi.
   - Bezpośredni zapis wyników
     Moduł odpowiedzialny za zapisywanie otrzymanych wyników.
   - Wysyłanie wyników do zdalnego komputera
     Moduł odpowiedzialny za pakowanie i przesyłanie danych do innego komputera.

3. Zdalny zapis wyników
   Moduł, który odbiera dane od innego komputera, rozpakowuje je, sprawdza ich kom-



Rysunek 3.2: Podział programu na moduły

pletność, po czym wywołuje funkcje bezpośredniego zapisu. Może być wydzielony jako niezależny program.

3.4. Podział na wątki

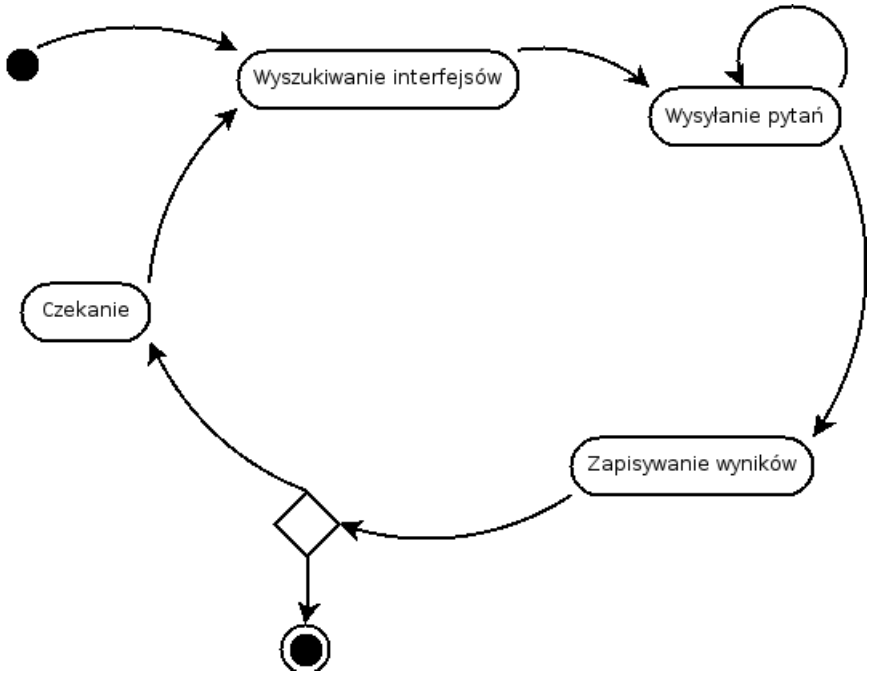
Program składa się z następujących wątków.

- Główny
  Jest odpowiedzialny za wczytywanie konfiguracji i tworzenie pozostałych wątków.
- Wątek odpytujący
  Dla każdego urządzenia tworzony jest dokładnie jeden wątek realizujący funkcjonalność modułu zbierającego dane.
- Wątek zapisujący

Moduł zapisujący wyniki otrzymane od innych komputerów jest realizowany przez osobny wątek, ale nic nie stoi na przeszkodzie, żeby napisać całkiem osobną implementację w postaci programu realizującego tylko tę funkcjonalność.

3.5. Język programowania

Głównym powodem napisania nowego programu jest niewystarczająca wydajność MRTG spowodowana przez konieczność interpretowania kodu napisanego w Perlu. Nowy program, w celu uniknięcia narzutu związanego z interpretowaniem kodu, powinien być napisany w języku programowania dającym możliwość kompilowania do kodu maszynowego.



Rysunek 3.3: Cykl działania modułu zbierającego dane

Rozdział 4

Implementacja

4.1. Biblioteki

Pisząc mój program wykorzystałem biblioteki opisane w kolejnych podpunktach.

4.1.1. Net-SNMP

Net-SNMP [7] jest pakietem służącym do obsługi SNMP na komputerach. W skład tego pakietu wchodzi agent SNMP, który umożliwia monitorowanie określonych parametrów pracy komputera, przez dowolnego menedżera SNMP. Innym przydatnym składnikiem pakietu jest zbiór gotowych programów, które mogą być wykorzystane podczas pisania skryptów w języku powłoki, ale z mojego punktu widzenia najciekawszym składnikiem jest biblioteka, umożliwiająca pisanie własnych programów wykorzystujących SNMP. Z zalet Net-SNMP można skorzystać na wielu platformach, dzięki temu programy pisane z wykorzystaniem tej biblioteki, o ile są dobrze napisane, to mogą działać na różnych architekturach z różnymi systemami operacyjnymi. SNMP v3 nie zostało zaimplementowane w mojej pracy (bo nie było takiej potrzeby), ale ponieważ biblioteka jest gotowa do obsługi SNMP v3, wiec kiedy zaistnieje potrzeba, dodanie obsługi SNMP v3 nie powinno stanowić dużego problemu.

W bibliotece wykorzystywane są dwie struktury danych:

- `snmp_session` — Zawiera informacje potrzebne do nawiązania sesji SNMP, takie jak wersja protokołu i adres urządzenia,
- `snmp_pdu` — Zawiera PDU, wykorzystywane do tworzenia pytań i odbierania odpowiedzi.

W moim programie korzystam z następujących funkcji:

- `snmp_sess_init()` — przygotowuje strukturę `snmp_session`,
- `snmp_sess_open()` — otwiera sesje SNMP, przekazuje wskaźnik, z którego należy korzystać podczas wysyłania pytań,
- `snmp_sess_send()` — wysyła PDU z pytaniem korzystając z otwartej sesji, przekazuje identyfikator wysłanego pytania,
- `snmp_sess_select_info()` — przekazuje wartości, które powinny być użyte do wywołania funkcji `select()`. Wywołanie `select()` z tymi wartościami zawiesza działanie programu do czasu nadejścia pakietu z odpowiedzią,

- `snmp_sess_read()` — odczytuje odpowiedź,
- `snmp_sess_timeout()` — sprawdza czy podczas czekania na odpowiedź przekroczono zadany limit czasu,
- `snmp_sess_synch_response()` — wysyła pytanie i czeka na odpowiedź lub przekroczenie limitu czasu,
- `snmp_sess_close()` — zamyka sesje SNMP,
- `snmp_pdu_create()` — tworzy puste PDU,
- `snmp_add_null_var()` — dodaje zmienne do PDU,
- `snmp_fix_pdu()` — usuwa z PDU zmienną wskazywaną przez `err_index`,
- `snmp_free_pdu()` — usuwa PDU.

4.1.2. LibXML

LibXML [3] jest biblioteką służącą do obsługi dokumentów w formacie XML i jest w pełni zgodna ze wszystkimi stosowanymi w praktyce standardami i rozszerzeniami XML [2, 6, 13]. Została stworzona na potrzeby środowiska GNOME [12], i jest wykorzystywana w wielu programach, dzięki czemu ewentualne błedy są szybko wykrywane i usuwane. Umożliwia sprawdzanie poprawności struktury dokumentu zgodnie z DTD.

Najważniejsze struktury:

- xmlParserCtxtPtr — kontekst parsera, odpowiedzialny za sterowanie działaniem parsera.
- xmlDocPtr — reprezentacja dokumentu w formie drzewa, umożliwia łatwy dostęp do wybranych elementów i ich atrybutów.

W moim programie korzystam z funkcji:

- `xmlNewParserCtxt()` — tworzy kontekst parsera,
- `xmlCtxtReadFile()` — wczytuje plik, tworzy dokument,
- `xmlFreeParserCtxt()` — zwalnia kontekst,
- `xmlFreeDoc()` — zwalnia dokument,
- `xmlDocGetRootElement()` — przekazuje korzeń dokumentu,
- `xmlStrcmp()` — porównuje xmlString z char*,
- `xmlStrdup()` — tworzy kopie xmlString jako char*,
- `xmlGetProp()` — pobiera wartość atrybutu,
- `xmlFree()` — zwalnia xmlString.

4.1.3. RRDtool

Format RRD został już wcześniej opisany w rozdziale 1.2, tutaj opiszę tylko funkcje dzięki którym mój program zapisuje dane do plików w formacie RRD.

RRDtool był tworzony raczej jako narzędzie do manipulacji plikami w formacie RRD, niż jako biblioteka, dlatego udostępnia interfejs w postaci głównych funkcji programów z pakietu RRDtool, co powoduje że funkcje te biorą jako argument listę wartości parametrów, które normalnie powinny być przekazane w wierszu poleceń. Jest to dość niewygodna metoda. Dodatkowym utrudnieniem jest fakt, że wewnątrz tych funkcji lista wartości jest przetwarzana z wykorzystaniem standardowej funkcji `getopt`, która korzysta ze zmiennych globalnych, co doskonale się sprawdza w przypadku kiedy funkcja wywoływana jest tylko raz, na początku działania programu. Natomiast w przypadku kiedy wywoływana jest kilka razy, przed każdym wywołaniem konieczne jest ręczne ustawianie wartości zmiennych globalnych na 0. Sprawa staje się jeszcze bardziej skomplikowana, kiedy funkcja z biblioteki RRDtool jest wywoływana przez kilka wątków, pojawia się wtedy potrzeba zapewniania synchronizacji. Autorzy RRDtool dostrzegli ten problem i począwszy od wersji 1.2, dostępne są dodatkowe funkcje, które dzięki zmianom w sposobie przekazywania argumentów umożliwiają współbieżne wykonywanie funkcji przez kilka wątków. Opis funkcji, które udostępnia RRDtool:

- `rrd_create_r` — tworzy nowy plik w formacie RRD,
- `rrd_update_r` — dopisuje do pliku nowe dane,
- `rrd_tune` — dostosowuje parametry pliku w formacie RRD.

Opis moich funkcji, które wywołują funkcje z biblioteki RRDtool:

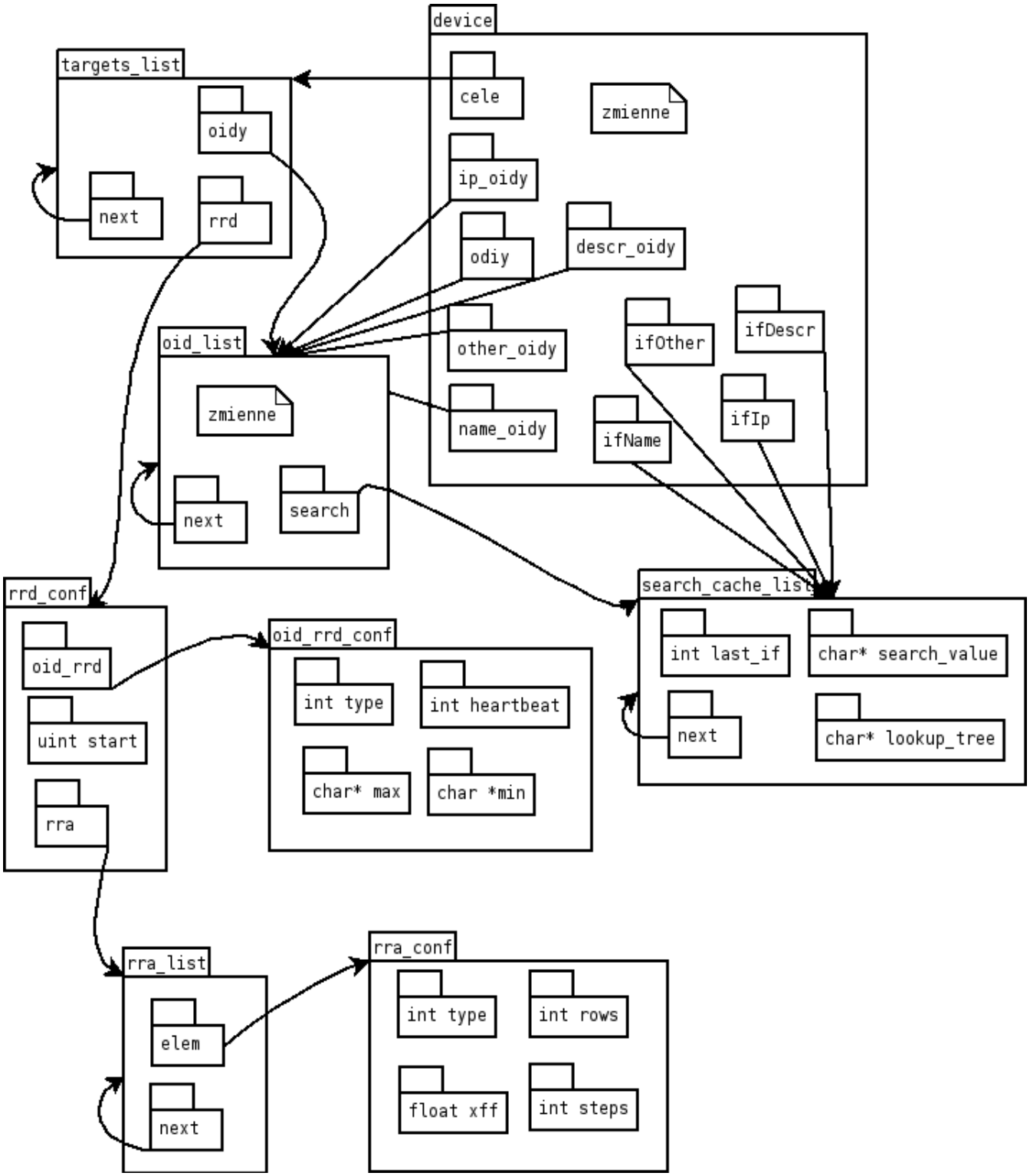
- `mgr_rrd_create` — przetwarza parametry jakie powinien mieć nowy plik, generuje argumenty i wywołuje funkcję `rrd_create_r`,
- `mgr_rrd_tune` — żeby dostosować parametry pliku do tych podanych w pliku konfiguracyjnym, generuje argumenty i wywołuje funkcję `rrd_tune`,
- `mgr_rrd_update` — przygotowuje argumenty i wywołuje `rrd_update_r`.

4.2. Struktury danych

Ze względu na dążenie do maksymalizacji szybkości działania, struktury danych zostały zoptymalizowane tak, żeby umożliwić szybki dostęp do często wykorzystywanych danych, a jednocześnie, żeby narzut pamięciowy był możliwie jak najmniejszy. Niestety takie podejście sprawia, że struktury stają się dość złożone i czasami ciężko jednoznacznie wskazać gdzie przebiega podział między dwoma strukturami.

Dla każdego procesu potomnego, zarówno odpytującego jak i zapisującego, tworzona jest struktura zawierająca wszystkie informacje niezbędne do działania. W przypadku procesu odpytującego taką strukturą jest `device`. Przechowuje ona zarówno informacje o urządzeniu, celach, obiektach, których wartości chcemy monitorować, jak też specyfikację, na podstawie której powinny być tworzone pliki w formacie RRD, do których zapisywane są cele. Uproszczona wersja tej struktury (z pominięciem niektórych zmiennych) przedstawiona została na rysunku 4.1.

Przeznaczenie ważniejszych struktur z tego diagramu jest następujące:
`oid_list` — Struktura ta jest listą wszystkich obiektów na danym urządzeniu, których wartości nas interesują. Każdy element listy zawiera dane o pojedynczym obiekcie. W nim są



Rysunek 4.1: Struktury odpowiedzialne za przechowywanie informacji o urządzeniu

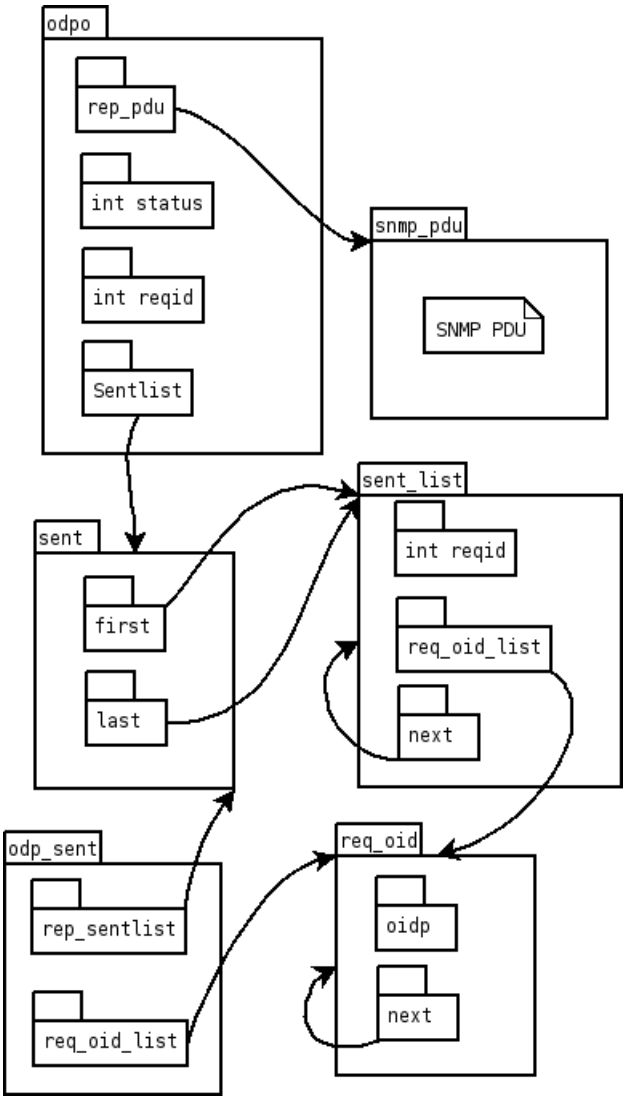
przechowywane informacje o numerycznej i tekstowej reprezentacji identyfikatora obiektu, typie, wartości i czasie, kiedy wartość została otrzymana. Na podstawie tej listy wysyłane są pytania i do niej zapisywane są odpowiedzi. W przypadku obiektów, które muszą być wyszukiwane w drzewie obiektów na urządzeniu, przechowywany jest wskaźnik do struktury zawierającej informacje, na podstawie których przeprowadzamy wyszukiwanie.
`targets_list` — W tej strukturze przechowywane są informacje o celach. Pojedynczy cel składa się z listy obiektów i konfiguracji pliku w formacie RRD, do którego powinien zostać zapisany. Dodatkowo przechowywane są też informacje o tym jakie obliczenia należy wykonać na wartościach wszystkich obiektów przed wysłaniem ich do modułu zapisującego oraz z którego modułu zapisującego należy skorzystać.
`search_cache_list` — Lista pełniąca rolę pamięci podręcznej dla interfejsów wymagających wyszukiwania. Zawiera informacje o drzewach, w których wyszukujemy i wartości przypisane do kluczy. Wszystkie obiekty, które wymagają wyszukiwania po tych samych kluczach, zawierają wskaźnik do tego samego elementu listy. Dzięki temu wyszukiwanie odbywa się tylko raz, bez względu na to ile obiektów korzysta z tych samych wartości.

Ze względu na asynchroniczne wysyłanie pytań, istnieje konieczność utrzymywania listy pytań, które zostały wysłane, a na które jeszcze nie przyszła odpowiedź. Wysyłanie pytań odbywa się poprzez wywołanie funkcji udostępnianych przez bibliotekę Net-SNMP. Biblioteka ta zajmuje się niskopoziomową częścią obsługi pytań, w szczególności sama zajmuje się obsługą pakietów, dla których przekroczono limit czasu. Sprawia to lekki problem, ponieważ jeśli pakiet zaginie, to powinien być usunięty z listy pytań, które zostały wysłane i na których odpowiedź czekamy. Można to zrobić wyłącznie wewnątrz funkcji zdefiniowanej przez użytkownika, a która jest wywoływana podczas przetwarzania każdej odpowiedzi. Funkcji takiej można przekazać tylko jeden wskaźnik, a ponieważ funkcja ta musi przekazywać liczbę całkowitą, ewentualny wynik (jak np. informacja czy wystąpił błąd lub numer pytania, dla którego przetwarzamy odpowiedź) też powinien być przekazany w strukturze podanej jako parametr. Na rysunku 4.2 przedstawiłem struktury, które są wykorzystywane do obsługi listy wysłanych pytań.

Struktura `odpo` jest przekazywana jako argument dla funkcji przetwarzającej odpowiedz.
`sent` — Struktura zawierająca informacje o liście wysyłanych żądań.
`sent_list` — Zawiera informacje o pojedynczym żądaniu.
`req_oid` — Przechowuje listę OID, które były wysłane w żądaniu.
`odp_sent` — Struktura pomocnicza używana do usuwania bieżącego żądania z listy wysłanych żądań.

Ze względu na bardzo rozbudowane możliwości konfiguracyjne, w szczególności fakt stosowania szablonów prawie na każdym etapie tworzenia konfiguracji, wczytywanie jej wymaga stosowania kilku struktur pomocniczych, w których przechowywane są częściowe wyniki. Wzajemne relacje tych struktur są przedstawione na rysunku 4.3.

- conf_globalne — Struktura przechowująca wartości globalnych atrybutów i wskaźniki do pozostałych struktur pomocniczych. W strukturze device_list tworzony jest wynik wczytywania konfiguracji.
- `templates` — Struktura przechowująca informacje o szablonie reguł.
- `rules` — Struktura przechowująca informacje o poszczególnych regułach.
- `ds_template` — Struktura przechowująca informacje o liście szablonów definicji typów przechowywanych danych.

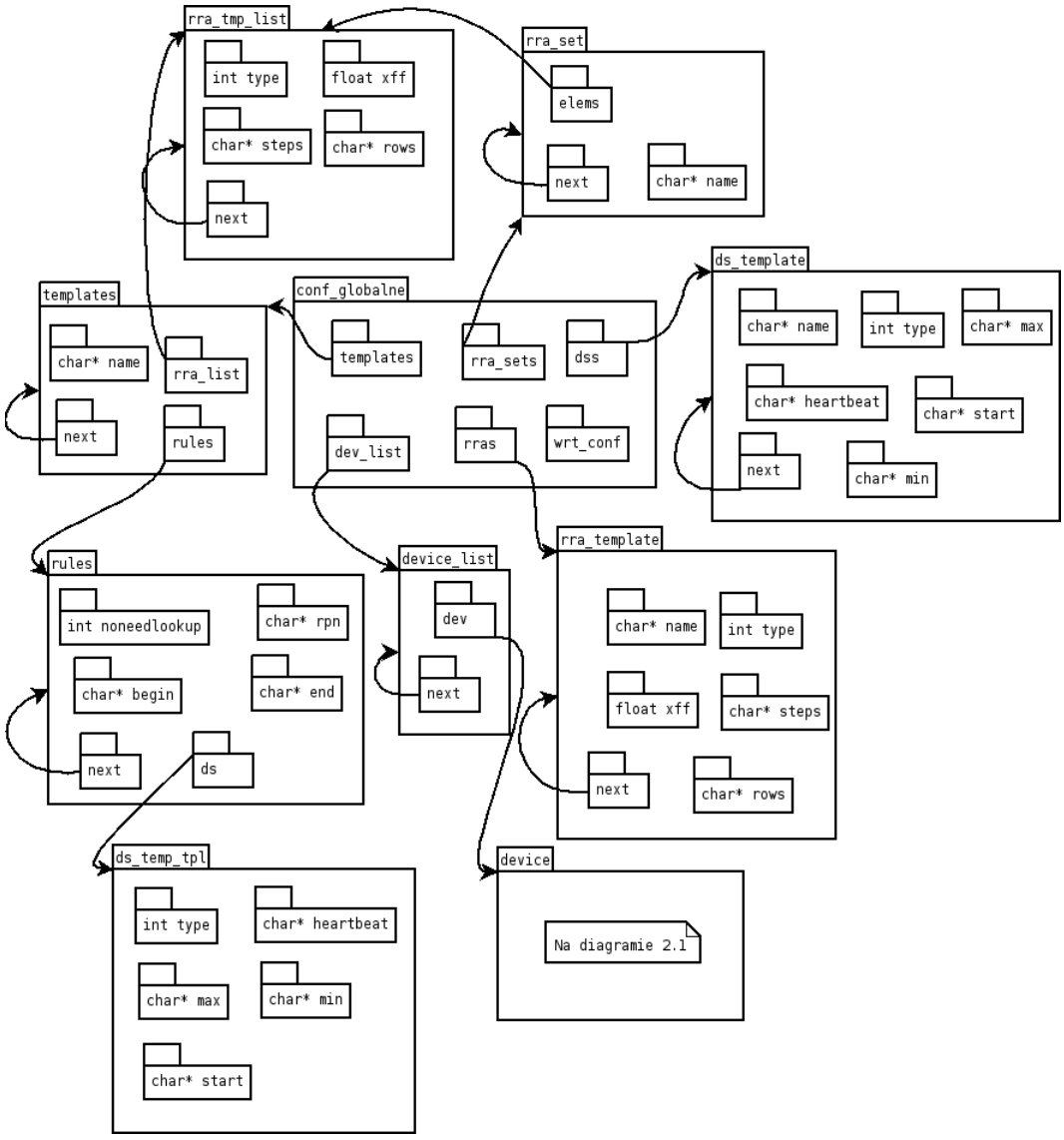


Rysunek 4.2: Struktury odpowiedzialne za komunikację z biblioteką Net-SNMP

- `ds_temp_tpl` — Struktura wykorzystywana do budowania szablonu definicji typów przechowywanych danych.
- `rra_template` — Struktura przechowująca informacje o liście szablonów reguł konsolidacji danych.
- `rra_tmp_list` — Struktura wykorzystywana do budowania szablonu reguł konsolidacji danych.
- `rra_set` — Struktura przechowująca informacje o zbiorze szablonów reguł konsolidacji danych.

4.3. Opis najważniejszych funkcji

Ze wszystkich funkcji z jakich składa się mój program, warto wymienić te najważniejsze:



Rysunek 4.3: Struktury odpowiedzialne za wczytywanie konfiguracji

- `parseFile` — wczytuje plik z konfiguracją, sprawdza jego zgodność z DTD, przekazuje strukturę zawierającą konfigurację.
- `rozdziel_procesy` — dla każdego urządzenia zdefiniowanego w pliku konfiguracyjnym uruchamia nowy proces do jego obsługi.
- `ask` — funkcja odpowiedzialna za obsługę urządzenia. Cyklicznie wysyła pytania i przetwarza odpowiedzi.
- `search_interface` — przetwarza listy obiektów, które wymagają wyszukiwania i jeśli znajdzie element nie posiadający prawidłowego numeru interfejsu, to wysyła do urządzenia pytania. W przypadku znalezienia pasującego interfejsu, zapisuje informacje o nim w pamięci podręcznej.
- `zapisuj_do_rrd` — zapisuje wyniki do pliku, jeśli plik w formacie RRD nie istnieje, to wywoływana jest funkcja `mgr_rrd_create`, w przeciwnym wypadku przed pierwszym

wywołaniem funkcji `mgr_rrd_update` wywoływana jest funkcja `mgr_rrd_tune`.

4.4. Plik konfiguracyjny

W początkowym etapie tworzenia programu założono, że format pliku konfiguracyjnego może być zgodny z wykorzystywanym w MRTG. Niestety format MRTG, mimo że umożliwiający dodawanie własnych znaczników, nie był w stanie umożliwić w prosty i wygodny sposób definiowania celu zawierającego więcej niż dwa parametry i który w dalszym ciągu byłby zgodny z MRTG. Stanąłem przed koniecznością wprowadzania zmian, które mimo że zachowają ogólny format sprawią, że pliki konfiguracyjne mojego programu nie będą zgodne z MRTG. Postanowiłem porzucić ograniczenia stawiane przez format MRTG i stworzyć własny format, wykorzystując do tego celu język XML. Dzięki temu można było wprowadzić kilka ciekawych rozwiązań, takich jak automatyczna walidacja dzięki sprawdzaniu zgodności z DTD, czy system szablonów znacznie ułatwiający konfiguracje w przypadku dużej liczby urządzeń. Przykładowy plik konfiguracyjny został zaprezentowany w dodatku A.

4.5. Protokół komunikacji

Między modułem odpytującym a zdalnym modułem zapisującym dane przesyłane są za pomocą połączenia TCP/IP. Ze względu na koszt otwierania nowego połączenia, otwierane jest ono tylko w razie potrzeby (przed pierwszą transmisją i w przypadku gdy zostanie zerwane).

Protokół wykorzystywany do obsługi połączenia jest bardzo prosty. Po nawiązaniu połączenia moduł odpytujący, dla każdego celu, który ma być zapisywany z wykorzystaniem tego połączenia, przesyła parametry plików w formacie RRD. Potem przy każdej aktualizacji danych, dla każdego celu przesyła gotowe argumenty używane do wywołania funkcji `rrd_update_r`. Moduł zapisujący dokonuje aktualizacji analogicznie do tego co zostało opisane w punkcie 4.1.3.

Aktualna wersja nie zawiera żadnych mechanizmów potwierdzających tożsamość stron połączenia. Można więc podszyć się pod dowolną ze stron komunikacji (i np. fałszować zapisywane dane). Prostym rozwiązaniem zwiększającym bezpieczeństwo i zapewniającym uwierzytelnianie może być wykorzystanie SSL. Dzięki szyfrowaniu przesyłanych danych nie będzie możliwe ich podsłuchiwanie, a wykorzystanie certyfikatów umożliwi autoryzację.

4.6. Sposób instalacji

4.6.1. Instalacja ręczna

- Wymagane komponenty
  Należy się upewnić, że w systemie znajdują się biblioteki wymagane do poprawnego działania programu. Program wymaga następujących bibliotek:
  – `Net-SNMP` w wersji nowszej niż 5, program testowany był z wersją 5.2.1.2,
  – `libxml2` w wersji co najmniej 2.6, program testowany był z wersją 2.6.26,
  – `RRDtool` ze względu na problemy ze współbieżnym wywoływaniem funkcji, opisane w rozdziale 4.1.3, wersje rrdtool starsze niż 1.2 nie są już wspierane przez MGR. Testowany z wersją 1.2.6.

- Kompilacja
  Archiwum należy rozpakować do dowolnego katalogu. W przypadku gdy RRDtool jest

zainstalowany w katalogu, który nie znajduje się w ścieżce wyszukiwania bibliotek, należy poprawić plik Makefile, dodając do zmiennej LIBS opcje -L ze ścieżką do katalogu, w którym zainstalowana jest biblioteka (np. -L `/usr/local/rrdtool-1.2.6/lib/`). Polecenie *make* skompiluje program.

- Instalacja
  Główny plik programu należy przegrać w docelowe miejsce (np. `/usr/local/bin/mgr`). Przykładowe pliki konfiguracyjne (`mgr.dtd` i `mgr.xml`) należy dostosować do własnych potrzeb i skopiować w wybrane miejsce (np. do katalogu `/etc/`).

4.6.2. Instalacja w dystrybucji Gentoo Linux

Istnieje gotowy plik z regułami instalacji dla dystrybucji Gentoo Linux. Do skorzystania z niego potrzebna jest przynajmniej średnia znajomość dystrybucji Gentoo Linux, w szczególności umiejętność skonfigurowania systemu nakładek na Portage (ang. *Portage overlays*). Opis tego systemu wykracza poza ramy tej pracy, dlatego w razie wątpliwości polecam dokumentację dystrybucji [8]. W celu zainstalowania programu z wykorzystaniem mechanizmów dystrybucji Gentoo Linux należy:

1. Rozpakować plik mgr-gentoo.tar.bz2 do katalogu, na który wskazuje zmienna PORTDIR_OVERLAY.
2. Wykonać polecenie *emerge mgr* z uprawnieniami administratora systemu (ang. *root*). W systemie zainstalowany zostanie program wraz z bibliotekami, które są niezbędne do jego działania.
3. Należy dostosować plik konfiguracyjny (`/etc/mgr.xml`) do własnych potrzeb.
4. Do uruchomienia programu można wykorzystać skrypt startowy `/etc/init.d/mgr`, skrypt pobiera swoją konfigurację z pliku `/etc/conf.d/mgr`.

4.6.3. Sposób uruchamiania

Sposób działania programu zależy od argumentów podanych w wierszu poleceń i w pliku konfiguracyjnym (por. p. 4.4). Oto dostępne argumenty, które można przekazać w wierszu poleceń:

`-c conf` — wczytuje konfiguracje z pliku `conf`, jeśli brak, to program szuka pliku konfiguracyjnego `/etc/mgr.xml` , a w przypadku jego braku, `mgr.xml`,
`-d` — tryb pracy demona; w tym trybie program odłącza się od konsoli i działa w tle,
`-h` — wyświetla na ekran spis dostępnych opcji i kończy działanie,
`-p` — wczytuje konfiguracje urządzeń i dla każdego urządzenia uruchamia proces, który zajmie się jego obsługą,
`-w` — wczytuje konfigurację modułu zapisującego i uruchamia proces zapisujący,
`-u` — identyfikator użytkownika, z którego uprawnieniami ma działać program,
`-g` — identyfikator grupy, z której uprawnieniami ma działać program.
`-i` — ścieżka do pliku kontrolnego z identyfikatorem procesu; w przypadku braku tej opcji program próbuje utworzyć plik `/var/run/mgr.pid`.

Opcje podane w wierszu poleceń przesłaniają opcje podane w pliku konfiguracyjnym.

Rozdział 5

Wdrożenie

Ze względu na rosnąca konieczność zastąpienia MRTG jako narzędzia do monitorowania sieci szkieletowej UW, w czerwcu 2005 została wdrożona, w dziale sieciowym ICM, wersja beta mojego programu. Wersja ta zawierała jedynie podstawową funkcjonalność niezbędną do działania. Obciążenie systemu okazało się 10 razy mniejsze w porównaniu z MRTG. Od tamtej pory MGR przeszedł kilka modyfikacji dodających najbardziej potrzebne w danej chwili opcje i obecnie zawiera prawie całą funkcjonalność opisaną w rozdziale 2.

Do automatycznego generowania plików konfiguracyjnych wykorzystywany jest skrypt, który pobiera ogólne informacje o typie urządzenia z bazy danych i na tej podstawie odpytuje urządzenie, żeby dowiedzieć się jakie interfejsy są dostępne i wygenerować pliki konfiguracyjne. Obecnie ciągle jeszcze wykorzystywana jest podwójna konfiguracja: jeden plik dla MGR i jeden w formacie MRTG dla interfejsu generującego wykresy. Trwają prace nad natywną obsługą pliku konfiguracyjnego MGR przez interfejs wyświetlający wykresy, co pozwoli zrezygnować z konieczności generowania konfiguracji MRTG. Przykładowe wykresy stworzone na podstawie danych zebranych przez MGR zostały zaprezentowane w punkcie 6.2.

Przez długi czas wszystkie pliki RRD przechowywane były w jednym katalogu, ale wraz ze wzrostem liczby monitorowanych urządzeń, zaczęło to prowadzić do problemów wydajnościowych. Przy ponad 6500 plikach okazało się, że czas jakiego system operacyjny potrzebuje, żeby zlokalizować i otworzyć plik jest za długi. Rozwiązaniem problemu mogłoby być ograniczenie liczby plików, tak żeby wszystkie informacje o pojedynczym interfejsie były przechowywane w jednym pliku. Obecnie zdarza się że informacje te są przechowywane w 5 różnych plikach. Takie rozwiązanie dla MGR nie stanowi żadnego problemu, ale niestety wymaga żeby interfejs wyświetlający wykresy obsługiwał konfiguracje MGR. Inną możliwością, którą można było wprowadzić bez żadnych dodatkowych zmian, a która okazało się wystarczająco skuteczna, jest tworzenie osobnego katalogu dla każdego urządzenia. Dzięki temu w każdym katalogu znajduje się mniej niż 300 plików.

Rozdział 6

Testy

6.1. Testy wydajności

Celem testów jest porównanie obciążenia systemu przez MGR, MRTG i RTG, w przypadku monitorowania dużej liczby obiektów.

6.1.1. Założenia

Testy wydajności przeprowadzono na komputerze wyposażonym w procesor Pentium 4 o prędkości 3.06 GHz i 512 MB pamięci RAM, korzystającym z jądra systemu Linux w wersji 2.6.18-gentoo. Na czas pomiarów wyłączono wszystkie zbędne usługi, żeby zminimalizować ich wpływ na otrzymane wyniki.

Testy były przeprowadzone przy użyciu następujących wersji bibliotek:

- libxml2 wersja 2.6.26,
- net-snmp wersja 5.2.1.2,
- rrdtool wersja 1.2.6.

Do testów użyto następujących wersji programów:

- mgr 18062006,
- mrtg 2.14.5,
- rtg 0.7.4.

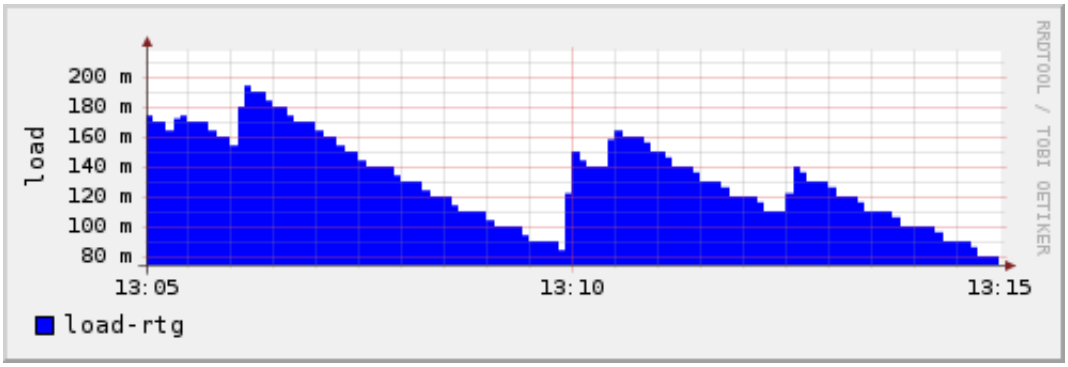
Komputer podłączony był do sieci szkieletowej w węźle na Krakowskim Przedmieściu, za pomocą karty sieciowej zgodnej ze standardem 100BASE-TX pracującej z prędkością 100 Mb/s. Do testów wykorzystano pełną konfigurację, która na co dzień jest wykorzystywana do monitorowania sieci szkieletowej UW. W efekcie tego, w trakcie testu monitorowano prawie 3800 celów znajdujących się na 80 urządzeniach.

Jako kryterium pomiaru przyjęto wartość średniego 5 minutowego obciążenia systemu, oznaczającego ile procesów, gotowych do działania, czeka na dostęp do procesora. Obciążenie systemu powinno się mieścić w przedziale od 0 do liczby procesorów zainstalowanych w komputerze. Większe wartości oznaczają, że niektóre procesy muszą czekać na dostęp do procesora, co może się niekorzystnie odbić na ich wydajności. Dodatkowo niskie wartości obciążenia oznaczają, że komputer jest w stanie wykonywać inne zadania lub monitorować

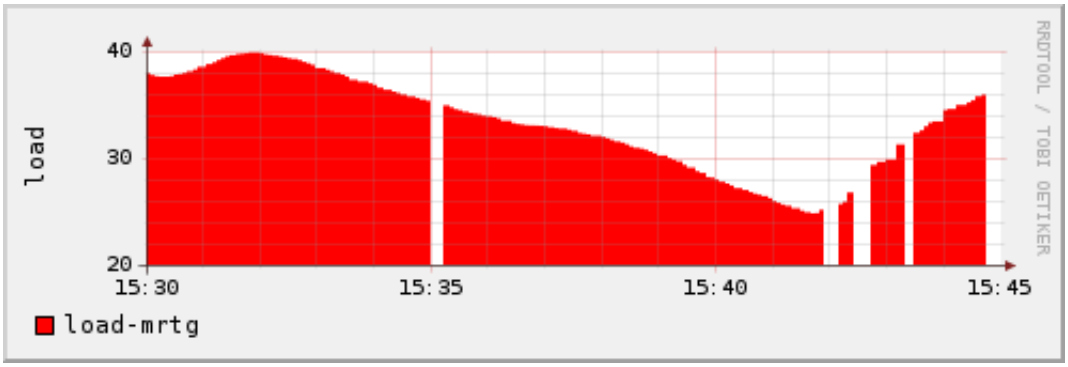
większą liczbę urządzeń. Programy były uruchamiane kolejno, tak żeby wyniki jednego programu nie wpływały na inne. Wartość obciążenia pobierano co 5 sekund z systemowego pliku `/proc/loadavg` i zapisywano do pliku w formacie RRD. W celu zapisywania dokładnych danych nie wykorzystano funkcji konsolidującej.

6.1.2. Wyniki

Na rysunku 6.1 przedstawiono wykres obciążenia systemu podczas działania RTG. Na osi Y przedstawiono wartość obciążenia w tysięcznych częściach jednostki (200 m oznacza obciążenie na poziomie 0.2), podczas gdy na osi X znajduje się czas w jednostkach 5 sekundowych. Na rysunku 6.3 znajduje się analogiczny wykres do tego z rysunku 6.1, ale przedstawiający obciążenie podczas działania MGR. Rysunek 6.2 przedstawia obciążenie systemu podczas działania MRTG, białe plamy na wykresie spowodowane są tym, że obciążenie systemu było tak duże, że czasami były problemy z uzyskaniem dwóch kolejnych próbek obciążenia.



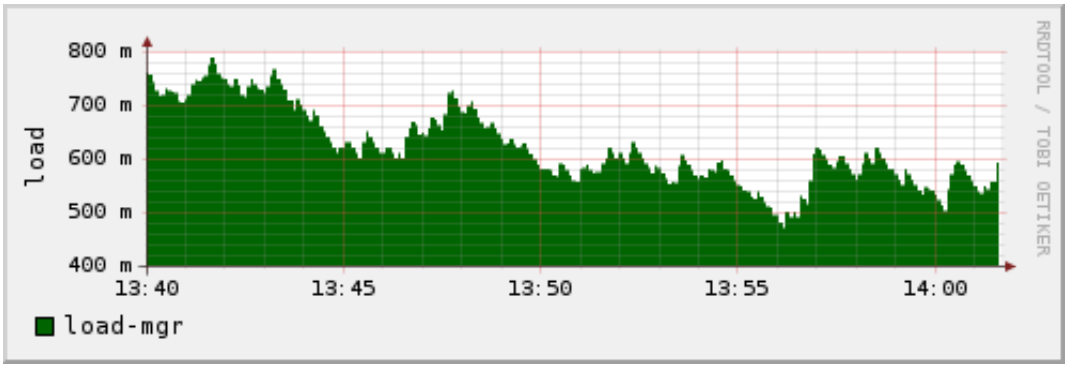
Rysunek 6.1: Wykres obciążenia procesora podczas pracy RTG



Rysunek 6.2: Wykres obciążenia procesora podczas pracy MRTG

Jak widać na rysunkach 6.1 i 6.3 obciążenie podczas działania MGR okazało się cztery razy większe niż w przypadku RTG. Przypuszczalnie głównym powodem takiego stanu rzeczy jest fakt, że RTG tylko wysyła pytania i wyniki zapisuje do bazy, a w przypadku MGR, pomijając większy narzut związany z otwieraniem plików, wykonywana jest jeszcze konsolidacja danych.

Porównanie rysunków 6.2 i 6.3 pokazuje, że obciążenie w przypadku MRTG jest 35 razy większe, jednakże ciężko określić jaki wpływ na ten wynik miał fakt, że podczas wykonywania testu obciążenie komputera było tak wysokie, że nawet zatrzymanie testu stanowiło poważny problem (na każdą komendę komputer reagował z kilkunastu sekundowym opóźnieniem).



Rysunek 6.3: Wykres obciążenia procesora podczas pracy MGR

Można przypuszczać, że na dużo mocniejszej maszynie (np. kilku procesorowej z kilkoma GB pamięci RAM) różnica wydajności byłaby mniejsza. Niestety sprawdzenie tego w praktyce nie było możliwe z powodu braku takiej maszyny.

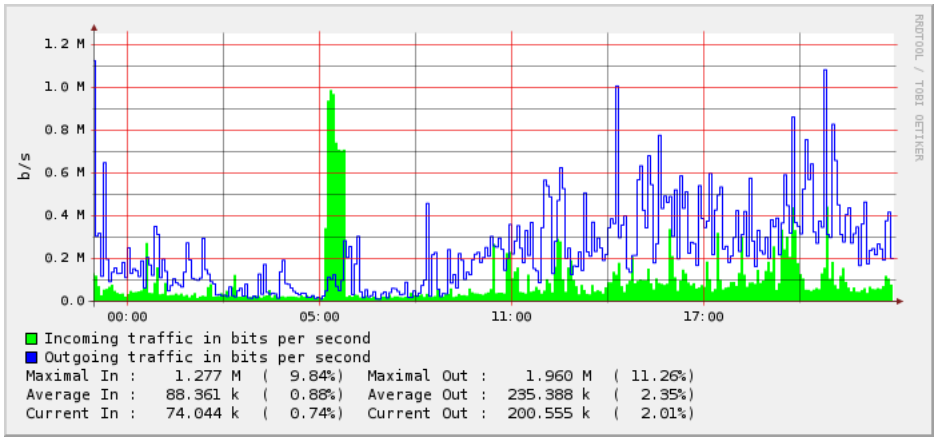
6.1.3. Podsumowanie

Wyniki testów potwierdzają że główny cel stawiany MGR (zdolność do efektywnego odpytywania dużej liczby urządzeń) udało się zrealizować. Wprawdzie w bezpośrednim porównaniu wypada gorzej od RTG, ale rekompensuje to dużo większą funkcjonalnością, dzięki której trzeba poświęcać mniej czasu na zarządzanie konfiguracją.

Obciążenie systemu podczas pracy MRTG wyraźnie pokazuje jak istotne było zastąpienie go innym programem. Testowy komputer nie był w stanie monitorować 80 urządzeń, co dla pozostałych dwóch programów nie stanowiło najmniejszego problemu.

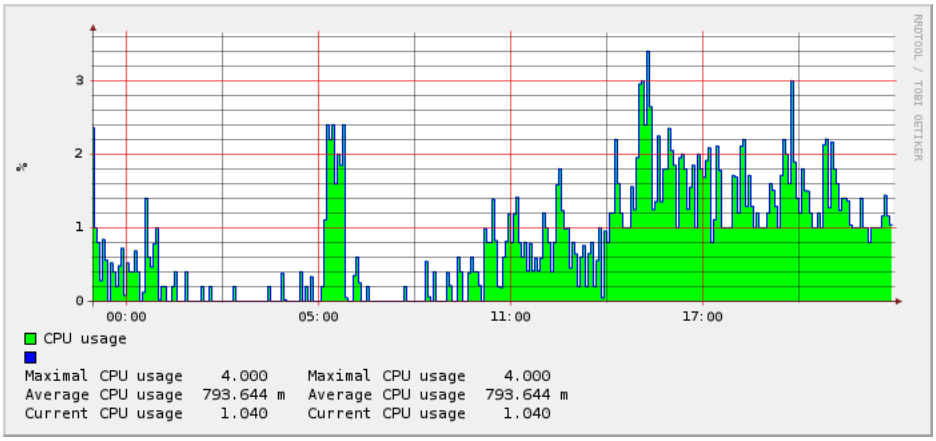
6.2. Testy działania

W tym punkcie przedstawiono przykłady praktycznego wykorzystania narzędzia. Załączono różne wykresy wygenerowane na podstawie zebranych danych.



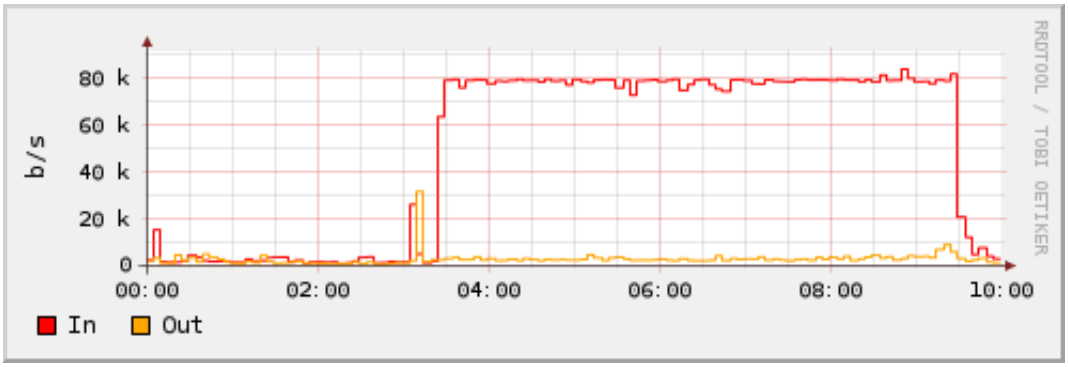
Rysunek 6.4: Wykres przesłanych danych przez interfejs urządzenia

Na rysunku 6.4 widzimy wykres wykorzystania łącza, na rysunku 6.5 widzimy zaś wykres obciążenia procesora na tym samym urządzeniu. Jak łatwo można zauważyć obciążenie



Rysunek 6.5: Wykres obciążenia procesora urządzenia

procesora jest zależne od wykorzystania łącza.

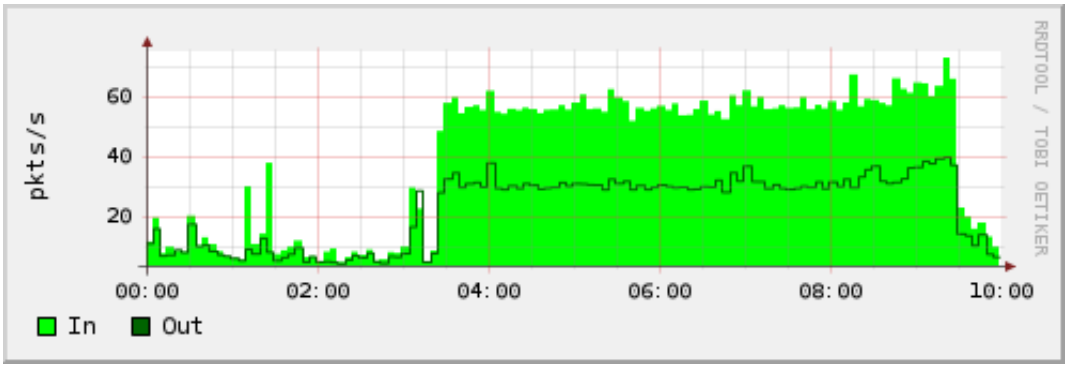


Rysunek 6.6: Wykres ilości przesyłanych danych

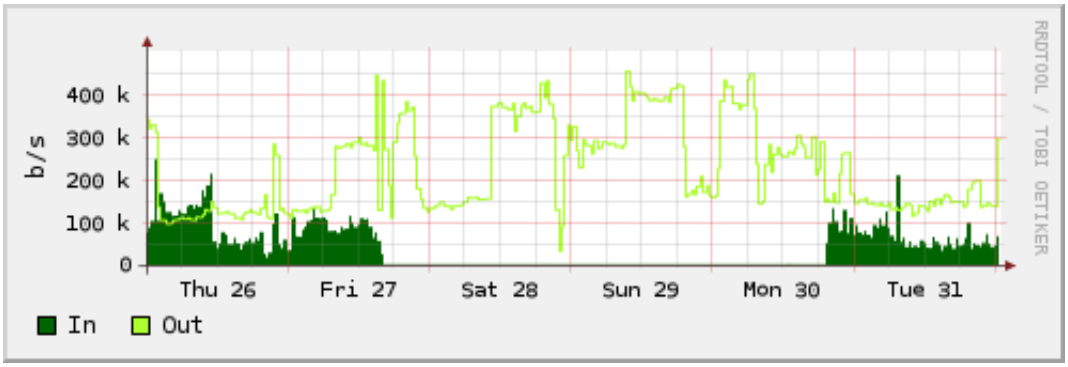
Rysunek 6.6 przedstawia wykres ilości przesyłanych danych, rysunek 6.7 przedstawia wykres liczby przesłanych pakietów, oba wykresy pochodzą z tego samego urządzenia, dotyczą tego samego interfejsu i przedstawiają ten sam okres czasu. Porównanie liczby odbieranych pakietów z ilością odbieranych danych pokazuje, że pakiety są duże. Podobne porównanie dotyczące wysyłanych pakietów pokazuje że ich rozmiar jest niewielki. Z dużej różnicy między liczbą przesyłanych i odbieranych danych wynika, że wysyłane są jedynie pakiety z potwierdzeniami dostarczenia danych. Jak pokazuje wykres, średnio wysyłany jest jeden pakiet potwierdzenia na dwa pakiety danych, co prowadzi do kolejnego wniosku, że łącze działa dobrze i nie ma na nim strat pakietów.

Na rysunku 6.8 przestawiony został wykres liczby przesyłanych danych. Jak można zauważyć w pewnym momencie wykres liczby odbieranych danych spada do zera. Dane dalej są normalnie wysyłane, co może sugerować jakiś asymetryczny problem z łączem. Ponieważ dane są wysyłane przez cały czas, sugeruje to, że dane przychodzące (w tym pakiety z potwierdzeniami odebrania wysłanych pakietów) są przesyłane innym, zapasowym łączem.

Rysunek 6.9 przestawia wykres temperatury, mierzonej przy wlocie i wylocie powietrza z urządzenia. Jak można łatwo zaobserwować w połowie czerwca temperatura wyraźnie wzrosła, o prawie 10 stopni. Powodem okazała się awaria klimatyzacji w serwerowni, w której pracuje monitorowane urządzenie. Jak się potem okazało w przypadku, gdy klimatyzator ustawiony był na temperaturę, której utrzymanie stanowiło problem, wyłączał się całkowicie. Kiedy na

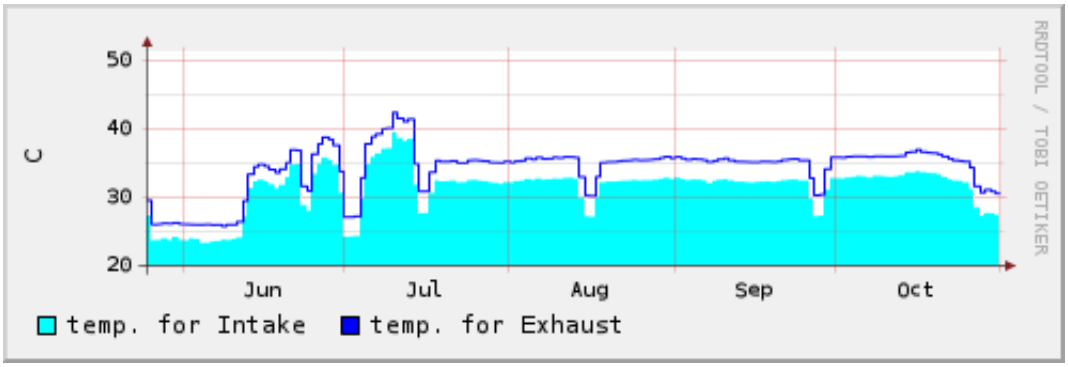


Rysunek 6.7: Wykres liczby przesyłanych pakietów



Rysunek 6.8: Tygodniowy wykres przesyłanych danych

początku lipca temperatura zaczęła zbliżać się do 40 stopni, ustawiono docelową temperaturę na poziomie, którego utrzymanie nie stanowiło problemu i temperatura obniżyła się do akceptowalnego poziomu. Dzięki monitorowaniu temperatury urządzeń, można łatwo wykryć awarie klimatyzacji i podjąć stosowne kroki zanim dojdzie do uszkodzenia sprzętu.



Rysunek 6.9: Wykres temperatury

Rozdział 7

Podsumowanie

Od czerwca 2005 roku napisany przeze mnie program wykorzystywany jest w Dziale Sieciowym ICM do monitorowania sieci szkieletowej Uniwersytetu Warszawskiego i obecnie realizuje prawie całą funkcjonalność opisaną w rozdziale 2. Jedyną funkcją, która nie została zaimplementowana, jest pobieranie danych przekazywanych przez skrypty. W trakcie ponad rocznego okresu użytkowania programu okazało się, że nie jest ona kluczowa i lepiej wprowadzać optymalizacje w funkcji odpytującej urządzenia, niż pisać tę praktycznie nie wykorzystywaną funkcję.

Mimo że program od dłuższego czasu jest wykorzystywany w środowisku produkcyjnym, to jego rozwój się jeszcze nie zakończył. Ciągle istnieją rzeczy, które można by dodać i poprawić np:

- szyfrowanie połączeń między modułem odpytującym a zdalnym modułem zapisującym. Umożliwi to bezpieczne korzystanie ze zdalnego modułu zapisującego w sieciach publicznych.
- obsługa nowych funkcji z wersji 1.2 RRDtool. W wersji 1.2 RRDtool dodanych zostało kilka ciekawych rozwiązań. Rozwiązaniem, z którym wiązane są największe nadzieje, ponieważ może przyczynić się do szybszego wykrywania awarii, jest detekcja anomalii w danych (np. dziwnie niskiego obciążenia na łączu, które zawsze jest mocno obciążone).
- pobieranie danych z lokalnych skryptów. Funkcja ta może okazać się przydatna do monitorowania parametrów pracy komputerów lub dodatkowych urządzeń (np. termometrów) podłączonych do komputera, bez konieczności pisania własnych modułów udostępniających dane przez SNMP.
- wczytywanie zmian w konfiguracji bez konieczności restartowania programu. Obecnie istnieje problem z aktualizowaniem konfiguracji poszczególnych procesów, dlatego może to wymagać poważniejszych zmian w komunikacji międzyprocesowej.
- wykorzystanie narzędzi z pakietu `autoconf` do sprawdzania dostępności bibliotek i automatyczne generowanie pliku `Makefile`. Pozwoli to uprościć instalację programu.

Dzięki mojemu programowi Dział Sieciowy ICM może monitorować wszystkie urządzenia, dla których istnieje taka potrzeba, co przy stosowanym wcześniej MRTG nie było możliwe. Pozwala to na dokładniejszą diagnostykę pojawiających się problemów i lepsze planowanie inwestycji.

Dodatek A

Plik konfiguracyjny

W tym dodatku przedstawiono przykładowy plik konfiguracyjny. Zawiera on konfigurację jednego urządzenia, którym jest komputer, na którym uruchomiony będzie MGR (o adresie 127.0.0.1). Urządzenie będzie odpytywane co 5 minut (300 sekund), a wyniki zapisywane będą do katalogu `/var/lib/mgr/rrd/`. W pliku zdefiniowano jeden szablon `ifbytes` i dwa cele korzystające z tego szablonu, `localhost_eth0` i `localhost_lo`, które monitorują liczbę przesłanych bajtów na interfejsach `eth0` i `lo`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "mgr-graph.dtd">
<config version="1">

<global>
    <time>300</time>
    <community>public</community>
    <block>600</block>
    <defaults ds="ds_counter" rra="rra_mrtg5"/>
    <writer>/var/lib/mgr/rrd/</writer>
    <user>mgr</user>
    <group>mgr</group>
    <pidfile>/var/run/mgr.pid</pidfile>
</global>

<writer_conf/>

<rrd>
    <ds_templates>
<ds name="ds_counter" type="COUNTER" heartbeat="time,1.1,*"
    max="U" min="0"/>
    </ds_templates>

    <rra_templates>
<rra name="5avg5min" type="AVERAGE" xff="0" steps="1" rows="600"/>
<rra name="5avg30min" type="AVERAGE" xff="0.5" steps="6" rows="700"/>
<rra name="5avg2h" type="AVERAGE" xff="0.5" steps="24" rows="775"/>
<rra name="5avg1d" type="AVERAGE" xff="0.5" steps="288" rows="797"/>
<rra name="5max5min" type="MAX" xff="0.5" steps="1" rows="600"/>
```

```
<rra name="5max30min" type="MAX" xff="0.5" steps="6" rows="700"/>
<rra name="5max2h" type="MAX" xff="0.5" steps="24" rows="775"/>
<rra name="5max1d" type="MAX" xff="0.5" steps="288" rows="797"/>
    </rra_templates>

    <rraSets>
<rraSet name="rra_mrtg5">
    <useRRA use="5avg5min"/>
    <useRRA use="5avg30min"/>
    <useRRA use="5avg2h"/>
    <useRRA use="5avg1d"/>
    <useRRA use="5max5min"/>
    <useRRA use="5max30min"/>
    <useRRA use="5max2h"/>
    <useRRA use="5max1d"/>
</rraSet>
    </rraSets>
</rrd>

<templates>
    <tpl name="ifbytes" ds_use="ds_counter">
<oid>ifInOctets.<lookup/></oid>
<oid>ifOutOctets.<lookup/></oid>
<graph class="bytes" display="true"/>
    </tpl>
</templates>

<devices>
    <device name="localhost" host="127.0.0.1"/>
</devices>

<targets>
<tgt name="localhost_eth0" device="localhost" off="false">
<useTemplate use="ifbytes" ifdesc="eth0"/>
<useRRASet use="rra_mrtg5"/>
</tgt>
<tgt name="localhost_lo" device="localhost" off="false">
<useTemplate use="ifbytes" ifdesc="lo"/>
<useRRASet use="rra_mrtg5"/>
</tgt>

</targets>
</config>
```

Dodatek B

Opis zawartości płytki dołączonej do pracy

Na dołączonej płytce CD znajdują się:

- praca magisterska w formacie PDF,
- źródła programu MGR,
- przykładowe pliki konfiguracyjne,
- pliki na podstawie których przeprowadzono testy,
- plik zawierający reguły instalacji MGR w dystrybucji Gentoo Linux.

Bibliografia

[1] Dział Sieciowy ICM Uniwersytetu Warszawskiego, *http://www.net.icm.edu.pl*

[2] Extensible Markup Language (XML) 1.0 (Fourth Edition), *http://www.w3.org/TR/REC-xml/*

[3] LibXML, Strona główna, *http://xmlsoft.org*

[4] Lista obiektów należących do IF-MIB, *http://www.mibsearch.com/vendors/RFC/objview/IF-MIB*

[5] Multi Router Traffic Grapher, *http://oss.oetiker.ch/mrtg/*

[6] Namespaces in XML 1.0 (Second Edition), *http://www.w3.org/TR/REC-xml-names/*

[7] Net-SNMP, Strona główna, *http://net-snmp.sourceforge.net*

[8] Portage Overlay, *http://gentoo-wiki.com/Portage_Overlay*

[9] Router Traffic Grabber, *http://rtg.sourceforge.net/*

[10] RRDtool, Strona główna, *http://oss.oetiker.ch/rrdtool*

[11] Wikipedia, strona zawierająca informacje o SNMP, *http://en.wikipedia.org/wiki/SNMP*

[12] Środowisko GNOME, Strona główna, *http://www.gnome.org/*

[13] XML Base, *http://www.w3.org/TR/xmlbase/*