POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRONIKI I TECHNIK INFORMACYJNYCH

Michał Dziekoński (lider)

Daniel Obrębski

Jakub Skrętowski

Paweł Szymczyk

Dokumentacja Końcowa

System zbierania statystyk częstości zapytań do serwerów FTP

Pod przewodnictwem
dr inż. Jacka Wytrębowicza

Maj 2015

Spis Treści

1. Projekt Wstępny

1.1 Treść zadania

System zbierania statystyk częstości zapytań do serwerów FTP (np. z użyciem Netfilter).

System obserwuje predefiniowany zbiór maszyn, w którym inicjuje i zatrzymuje pomiar a następnie zbiera dane. Na każdej obserwowanej maszynie instalowany jest agent, z którym komunikuje się serwer zarządzający. Operator współpracuje z serwerem zarządzającym poprzez: 1) skrypty powłoki, 2) przeglądarkę WWW. Ponadto należy zaprojektować moduł do Wireshark umożliwiający wyświetlanie i analizę zdefiniowanych komunikatów. System może działać w sieci IPv6 i IPv4.

1.2. Nazwa Systemu

FTP Stattr

1.3 Przyjęte założenia funkcjonalne i niefunkcjonalne

1.3.1 Założenia funkcjonalne

1.3.1.1 Aplikacja Agenta

- Zbieranie danych o ruchu FTP (1)
- Komunikacja z aplikacją serwerową, bazowana na gniazdach BSD (1)
- Reakcje na przychodzące żądania z aplikacji serwerowej (1)
- Zapisywanie dużych ilości zbuforowanych danych do pliku (3)
- Zbieranie danych o ruchu w określonych porach dnia, w określonych przedziałach czasu (4)
- Porzucanie danych starszych niż określony okres czasu (4)

1.3.1.2 Aplkacja Serwerowa

- Komunikacja międzyprocesowa z interfejsem terminalowym (1)
- Dodawanie, modyfikowanie i usuwanie wpisów z listy obserwowanych serwerów z agentami (1)
- Komunikacja z aplikacjami agentowymi, bazowana na gniazdach BSD (1)
- Wysyłanie poleceń i odbieranie danych z serwerów agentowych (1)
- Analiza zebranych danych z serwerów agentowych (3)
- Komunikacja z interfejsem przeglądarkowym (2)
- Zarządzanie periodycznymi i automatycznymi procesami zbierania danych z agentów, bez udziału komend od użytkownika (4)
- Zapis zebranych danych do pliku (4)
- Modyfikowanie przedziałów czasu zbierania danych dla określonych maszyn agentowych (4)
- Modyfikowanie ważności danych dla określonych maszyn agentowych (4)

1.3.1.3 Komunikacja przez gniazda/wtyczka wireshark

- Przygotowanie specyfikacji wiadomości komunikacyjnych (0)

- Przygotowanie wtyczki do Wiresharka, testującej poprawność komunikatów (0)
- Opracowanie scenariuszy testowych/testów automatycznych sprawdzających poprawność komunikacji (1)

1.3.1.4 Interfejs konsolowy

- Komunikacja międzyprocesowa z aplikacją serwerową (1)
- Wysyłanie prostych komend do aplikacji serwerowej (1)
- Odbieranie i wyświetlanie wyników prostych komend z aplikacji serwerowej (1)
- Rozszerzenie listy komend o pobieranie danych statystycznych w prostym formacie tekstowym (3)

1.3.1.5 Interfejs przeglądarkowy

- Komunikacja z aplikacją serwerową (2)
- Wysyłanie prostych komend do aplikacji serwerowej (2)
- Odbieranie i wyświetlanie wyników prostych komend z aplikacji serwerowej (2)
- Rozszerzenie listy komend o pobieranie danych statystycznych (3)
- Wyświetlanie danych statystycznych w prostej i klarownej reprezentacji listowej, tabelarycznej lub wykresowej (3)
- Eksport danych do pliku (4)
- Zapis pobranych danych do pamięci lokalnej przeglądarki (4)

1.3.1.6 Priorytety

- 0 - natychmiastowy, wymagany do rozpoczęcia właściwych prac implementacyjnych
- 1 - ważny, pierwszy etap implementacyjny
- 2 - średni, drugi etap implementacyjny, rozwojowy
- 3 - mało ważny, trzeci etap, wykończeniowy w stosunku do właściwych funkcjonalności
- 4 - opcjonalne, dodanie mile widzianych funkcji, jeśli wystarczy na to czasu

1.3.2 Założenia niefunkcjonalne

1.3.2.1 Łatwość konfiguracji

Program będzie udostępniał interfejsy konfiguracyjne w postaci ustawień w graficznym panelu sterowania oraz API do skryptowania. Użytkownik powinien być zaznajomiony z tematyką programu przed jego użyciem. Opis API oraz sposób obsługi panelu sterowania dostarczone zostaną w instrukcji obsługi. Szacowana trudność: ⅖.

1.3.2.2 Wydajność

Program musi w stanie sprawnie reagować na komendy użytkownika oraz być w stanie płynnie przekazywać informacje do serwera zarządzającego oraz sprawnie komunikować się z serwerami FTP, w razie potrzeby musi sygnalizować błędy oraz o ile to możliwe spróbować wznowić połączenie.

1.3.2.3 Bezpieczeństwo

Program powinien chronić użytkownika przed ewentualnymi szkodami wynikającymi z nieprawidłowej konfiguracji. Potrzebna jest więc walidacja danych wprowadzanych do programu.

1.3.2.4 Możliwość rozwoju systemu

Program będzie dostępny jako zamknięta i gotowa aplikacja przeznaczona na wiele stanowisk komputerowych. Aplikacje będzie można modyfikować skryptami w zależności od potrzeb.

1.4 Wybrane środowisko sprzętowo, programowe i narzędziowe (systemy operacyjne, języki programowania)

- Do prowadzenia komunikacji pomiędzy programem a maszynami na których zbierane są informacje o ruchu zostanie zastosowana aplikacja napisana w języku C++ korzystająca gniazd BSD.
- Do utworzenia interfejsu przeglądarkowego zastosowana zostanie aplikacja napisana w JavaScript używająca Backbone.js, Twitter Bootstrapa, jQuery oraz API WebSockets.
- Skryptowanie (przesyłanie komend z konsoli) odbywać się będzie w języku powłoki systemowej, jako wywołanie aplikacji z odpowiednimi parametrami, bądź jako sekwencja wprowadzanych danych po uruchomieniu programu komunikacyjnego z serwerem.
- Skrypty testujące działanie programu napisane zostaną w języku Python
- Rozwiązanie będzie projektowane dla systemów operacyjnych Linux i Mac OS X

1.5 Architekturę rozwiązania, tj. ilustrację i opis struktury logicznej systemu (koncepcyjnych bloków funkcjonalnych).

- Architektura zostanie podzielona na pięć części składowych:
- Aplikacji agenta działającego w trybie demona, nieustannie zbierającego dane do analizy
- Aplikacji serwerowej, działającej również w trybie “aktywnego” demona, zbierającego dane z agentów i wykonującego lokalne analizy
- Wtyczki do programu Wireshark, która posłuży do sprawdzania poprawności komunikacji między agentami a serwerem
- Interfejsu przeglądarkowego udostępniającego pełny zestaw konfiguracji i poleceń dla aplikacji serwerowej
- Interfejsu konsolowego pełniącego bardziej ubogą i “niegraficzną” wersję interfejsu przeglądarkowego

Serwer i agent zostaną oparte o moduły połączone ze sobą w jedną całość za pomocą kontrolerów. Zarówno agent jak i serwer będą dzieliły między sobą wspólny kod do komunikacji poprzez gniazda, będący modułem łączności. Będzie to swego rodzaju biblioteka wykorzystywana w komunikacji.

1.5.1 Agent

Agent zostanie wyposażony w moduł pomiarowy, zbierający dane o ruchu FTP na maszynie docelowej. Dane będą przechowywane w pamięci do czasu komunikacji z serwerem, reakcje na żądania serwera zapewni moduł usługowy.

1.5.2 Serwer

Serwer zostanie wyposażony w moduł analityczny, wykonujący obliczenia na potrzeby utworzenia statystyk pomiarów; moduł komend, nasłuchujący przychodzących poleceń z terminalu i przesyłający wyniki poleceń na terminal; moduł webowy, nasłuchujący przychodzących poleceń z aplikacji webowej, i przesyłający wyniki do użytkownika; moduł zbiorczy, wywołujący łączność z agentami i zbierający otrzymane dane dla modułu analitycznego.

1.5.3 Wtyczka do programu Wireshark

Wtyczka do programu Wireshark zdefiniuje sposób komunikacji między aplikacją główną a aplikacją Wireshark i określi, jak przeprowadzić obróbkę danych do czytelnego formatu.

1.5.4 Interejs przeglądarkowy

Interfejs przeglądarkowy, zbudowany jako “Single Page Application”, udostępni prosty lecz rozbudowany dostęp do wszelkich komend i statystyk zebranych przez serwer. Komunikacja między interfejsem a serwerem będzie odbywać się za pomocą protokołu WebSockets wykorzystującego JSON do przekazywania danych (w czasie rzeczywistym, asynchronicznie; nie mylić WebSockets z gniazdami). Wszystkie dane i statystyki będą prezentowane w przejrzystym, przyjemnym dla oka formacie listowym, tabelarycznym lub wykresowym (o ile użytkownik nie zarząda “czystych danych”, w formacie JSON).

1.5.5 Interfejs konsolowy

Interfejs konsolowy, zbudowany jako osobny proces, wykorzystując IPC będzie komunikował się z działającą w tle aplikacją serwerową, asynchronicznie wywołując zadane polecenia i oczekując na dane. Rezultatem wywołania polecenia w tym interfejsie będzie wyświetlenie danych na konsoli uzytkownika i oczekiwanie na dalsze komendy do przekazania. Ten interfejs będzie miał jedynie możliwość listowego wyświetlenia danych.

1.6 Sposób testowania

- Skrypty w Python - zautomatyzowane testy wykonujące zadane scenariusze testowe i sprawdzające poprawność ich wyników.
- Testy jednostkowe w C++ - użyte zostaną do sprawdzenia najważniejszych modułów aplikacji serwerowej i agenckiej - modułu łączności, modułu pomiarowego, modułu usługowego czy kontrolerów.
- Empiryczne testowanie - przygotowane odpowiednie zestawy scenariuszy testowych, które można odtworzyć w dowolnym późniejszym momencie w celu weryfikacji poprawności działania.

1.7 Sposób demonstracji rezultatów, scenariusze testów akceptacyjnych

- Poprawność działania wtyczki do programu Wireshark sprawdzimy poprzez włączenie aplikacji Wireshark, uaktywnienie odpowiedniego, zdefiniowanego ruchu sieciowego oraz prostą analizę otrzymanych danych. Powstanie do tego osobny skrypt lub aplikacja.
- Aby, sprawdzić czy system dobrze wykonuje pomiar, możemy wykorzystać aplikację Wireshark, która na podstawie wysłanych/odebranych pakietów, komunikatów potwierdzi poprawność danych ujawnionych w programie i/lub prostego wykresu.
- Aby zademonstrować obsługę błędów w połączeniu, spróbujemy podłączyć się do nieistniejącego serwera podając niepoprawny adres IP lub będąc odłączonym od sieci Internet.
- Zaprezentujemy dane historyczne zebrane przez program i udowodnimy ich poprawność.
- Aby pokazać dostępne opcje w programie, przeprowadzimy krótką miniprezentację. Pomyślne wykonanie będzie świadczyło o poprawnej obsłudze różnych przypadków.
- W celu zaprzentowania możliwości wykrywania anomalii w obserwowanym ruchu przygotujemy specjalny skrypt zlecający wykonanie wielu żądań jednocześnie do jednej z maszyn obserwowanych. Po wykonaniu skryptu serwer powinien zebrać dane z agenta na tej maszynie, poprawnie rozpoznać atak i wyświetlić dane na jego temat.

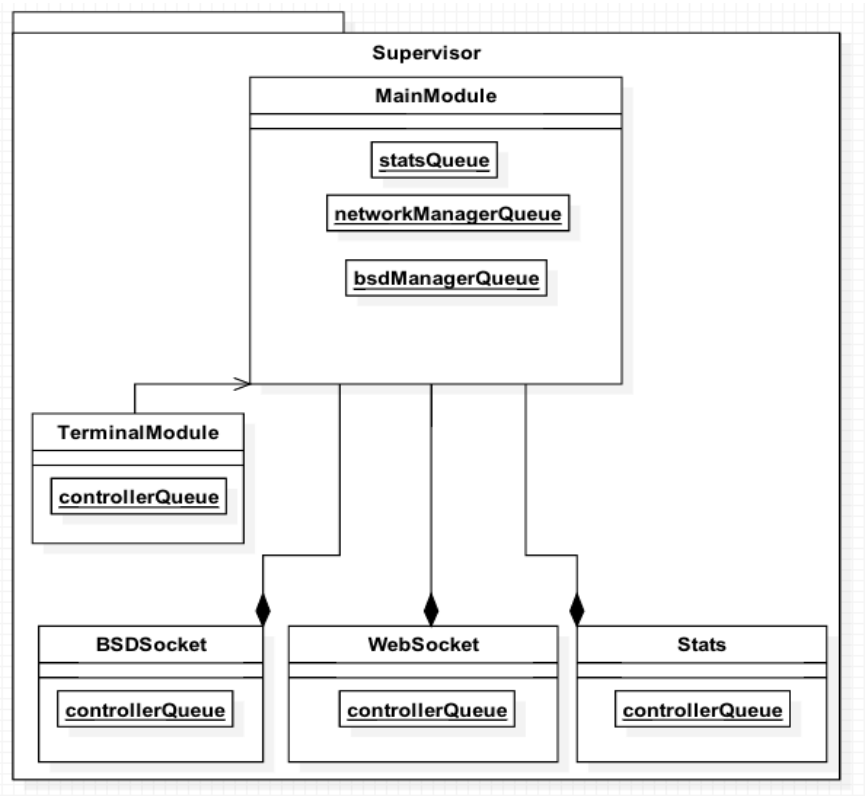
1.8 Podział prac w zespole

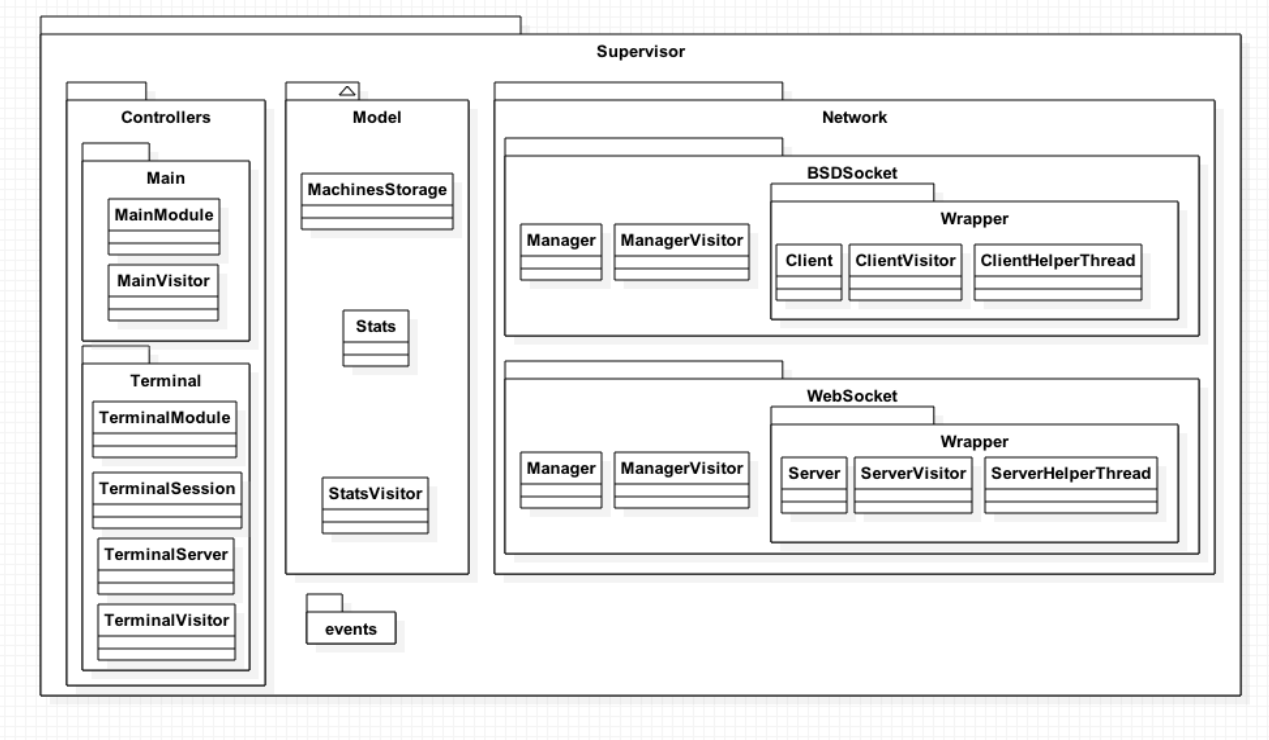
- **Michał Dziekoński** - przygotowanie szkieletu kodu dla Agenta i Supervisora, interfejs przeglądarkowy
- **Daniel Obrębski -** komunikacja na bazie gniazd, przygotowanie wtyczki Libpcap, moduł zbiorczy dla Serwera
- **Jakub Skrętowski** - moduł analityczny i komend dla Supervisora, wtyczka do Wiresharka
- **Paweł Szymczyk** - moduł pomiarowy i usługowy dla Agenta, interfejs konsolowy

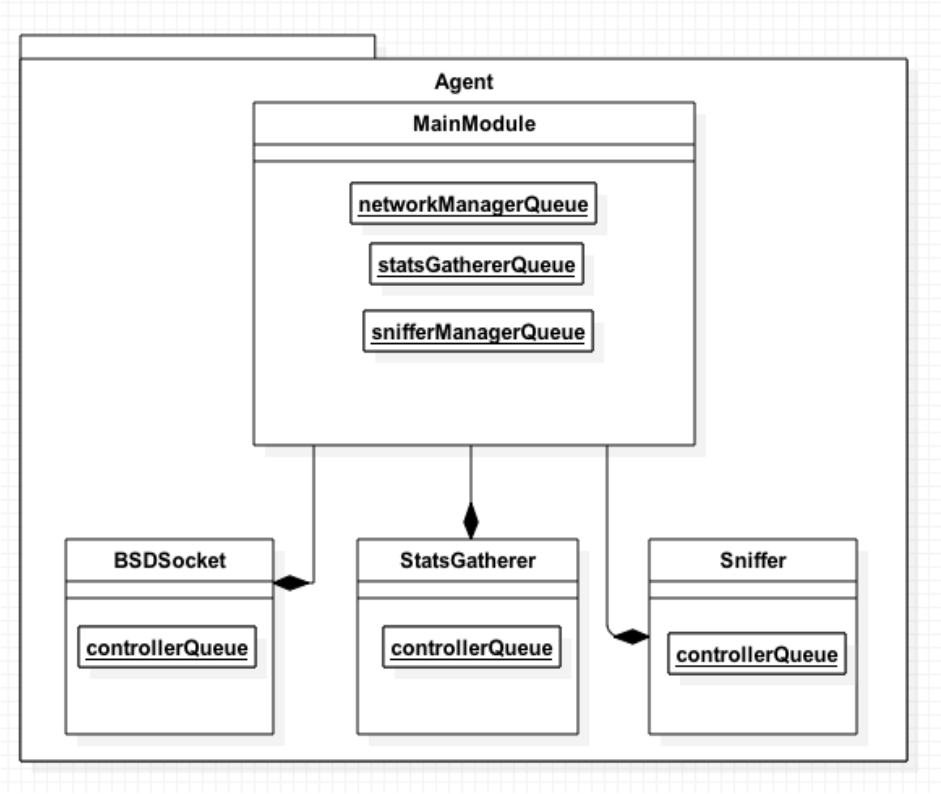
Projekt Końcowy

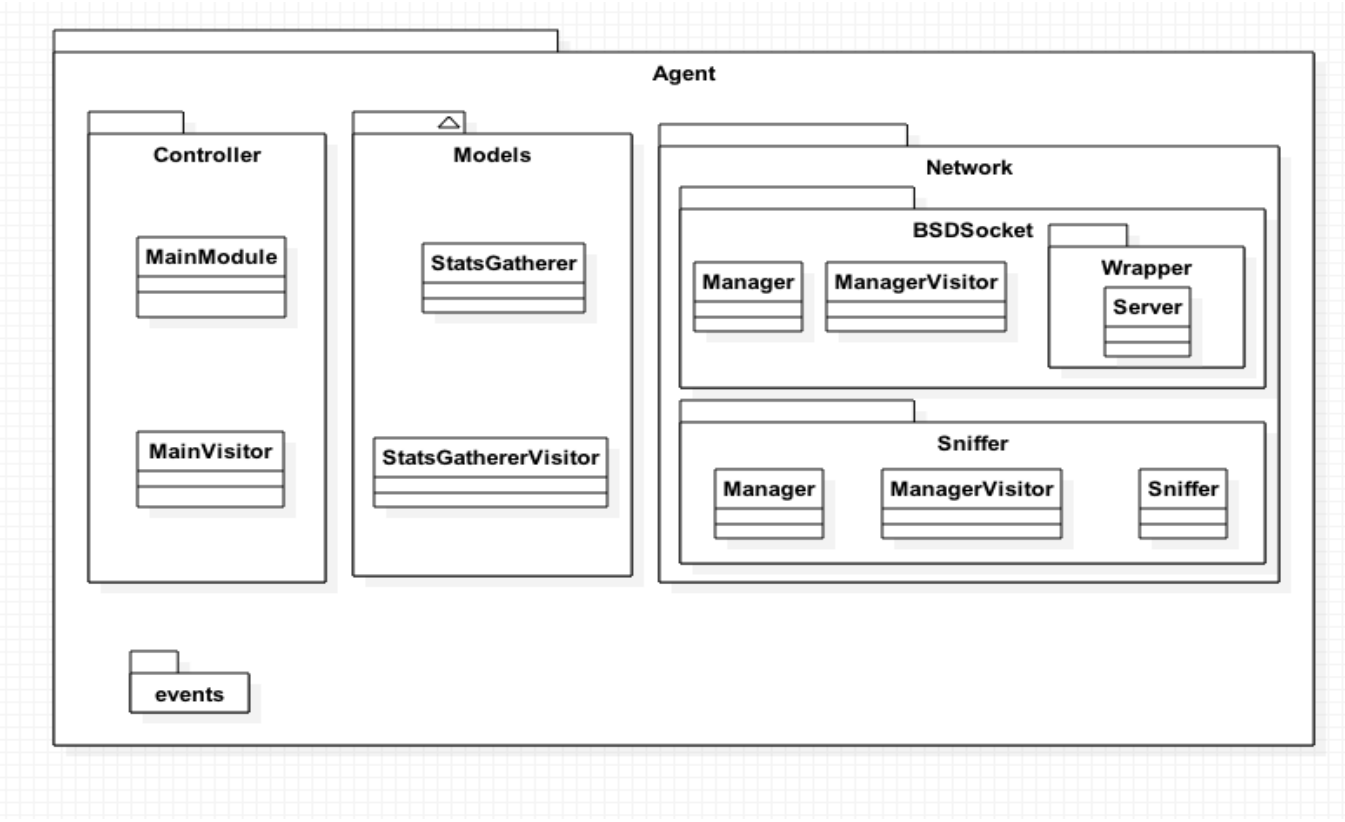
2. Diagram klas

Są to tylko rysunki poglądowe połączeń miedzy modułami, nie przedstawiają one faktyczne diagramu klas.






Agent
MainModule
networkManagerQueue
statsGathererQueue
snifferManagerQueue
BSDSocket
controllerQueue
StatsGatherer
controllerQueue
Sniffer
controllerQueue


Agent
Controller
MainModule
MainVisitor
Models
StatsGatherer
StatsGathererVisitor
Network
BSDSocket
Manager
ManagerVisitor
Wrapper
Server
Sniffer
Manager
ManagerVisitor
Sniffer
events

3. Definicja komunikatów

3.1 Webclient – Supervisor

Typ komunikacji: WebSockets + JSON

Webclient wysyła JSONy z polami “Route”, “Type”. Zostaje przekazane zdarzenie do supervisora, który w zależności od zawartości tych pól odpowiedno przetwarza zapytanie. W odpowiedzi ponownie wysyła JSONa z odpowiednimi danymi na temat maszyn.
“**Route**” = **machine** | **machine**/**ID**/**action**, gdzie ID to identyfikator maszyny, a action, podejmowana akcja np. "sync".
“**Type**” = “**GET**” | “**POST**”

3.2 Terminal – Supervisor

Typ komunikacji: Boost::asio Sockets + JSON

Terminal łączy się z serwerem terminala (działającego jako wątek Supervisor’a) na hoście lokalnym (port 4321) i pozwala na wysyłanie do niego komend:
-**add** <name> <ip> <port> - dodanie maszyny
-**remove** <id> - usunięcie maszyny
-**get** <id> - pobranie informacji o maszynie
Komunikaty są parsowane do obiektu JSON i wysyłane poprzez socket do serwera. Następinie komenda dodawana jest do kolejki oczekujących na przetworzenie zdarzeń w Supervisorze.

3.3 Agent – Supervisor

Typ komunikacji: BSDSocket + JSON

Agent w tej parze jest serwerem, to on nasłuchuje, supervisor jest klientem. I serwer i klient wysłają obiekt typu JSON, który jest zapisywany do gniazda w postaci std::string. Supervisor wysyła do agenta wiadomości typu { “**cmd**” : **action** }, gdzie **action** to {“**sync**”, “**ping**”, “**startsniffing**”, “**stopsniffing**”, “**change_filter**”}. Agent wysyła odpowiedzi do nadzorcy, informując go o tym jaką wykonał operację i/lub czy zakończyła się powodzeniem).

3.4 Agent – Sniffer

Typ komunikacji: Queue + JSON

Ponieważ sniffer jest częścią agenta, to komunikacja zachodzi za pośrednictwem dwóch kolejek. Agent wysyła odpowiednie zdarzenie - w zależności od tego czego oczekuje od Sniffera - są to zdarzenia - “ChangeFilter” - w tym zdarzeniu wysłany jest również JSON - z nowym filtrem, “StopSniffing” - zatrzymuje działanie sniffera, “StartSniffing” - uaktywnia działanie Sniffera, “IsSniffing” - pyta się o obecny stan Sniffera. W odpowiedzi na te zdarzenia do Agenta jest wysyłana wiadomość z aktualnym stanem programu węszącego.

4. Testowanie

4.1 Testowanie analizatora (Scenariusz Testowy)

1. Wysyłanie danych na serwera
   - Wyślij 5 plików o sumarycznym rozmiarze 5MB do serwera B.
   - Sprawdź poprawność przeanalizowanych danych.
2. Pobieranie danych z serwera
   - Wyślij żądanie pobrania danego pliku.
   - Sprawdź poprawność przeanalizowanych danych.
3. Wysyłanie na serwer z różnych źródeł
   - Wyślij po 2 pliki o różnych rozmiarach z maszyn o różnych IP
   - Filtruj wyniki po IP
   - Sprawdź poprawność danych
4. Pobranie danych z serwera
   - Pobierz po 2 pliki o różnych rozmiarach z maszyn o różnych IP
   - Filtruj wyniki po IP
   - Sprawdź poprawność danych

Wniosek: Na podstawie wskazań w programie wireshark oraz empirycznym porównaniu rozmiaru pobranych/wysłanych danych. Wyniki okazują się być zgodne z rzeczywistością, świadczy to o dobrej implementacji programu nasłuchującego (sniffera), a także o poprawnym działaniu analizatora ruchu.

4.2 Testowanie połączenia (Scenariusz Testowy)

1. Dodawanie maszyny
2. Edycja Statusu Maszyny (włączanie/wyłączanie) podsłuchiwania
3. Pobranie danych na temat maszyny
4. Pobranie maszyn
5. Edycja Maszyny

Wniosek: W przypadku poprawnego wykonania scenariusza w przypadku klienta sieciowego jak i terminala, mamy pewność, że komunikacja między klientem a serwerem z oparciem gniazd BSD została zrealizowana poprawnie.

5. Instrukcja instalacji i deinstalacji

Instalacja wymaganych komponentów:

* Kompilacja aplikacji w C++
** sudo apt-get install scons libboost-dev clang
** scons (wywołanie z poziomu katalogu głównego z plikami)
* Kompilacja aplikacji Webclienta
** sudo apt-get install nodejs nodejs-legacy npm
** npm install bower grunt grunt-cli -g
** npm install (wywołane z poziomu katalogu "webclient")
** bower install (wywołane z poziomu katalogu "webclient")
** grunt server (wywołane z poziomu katalogu "webclient", musi być ciągle włączone, ponieważ stanowi również rolę serwera plików)

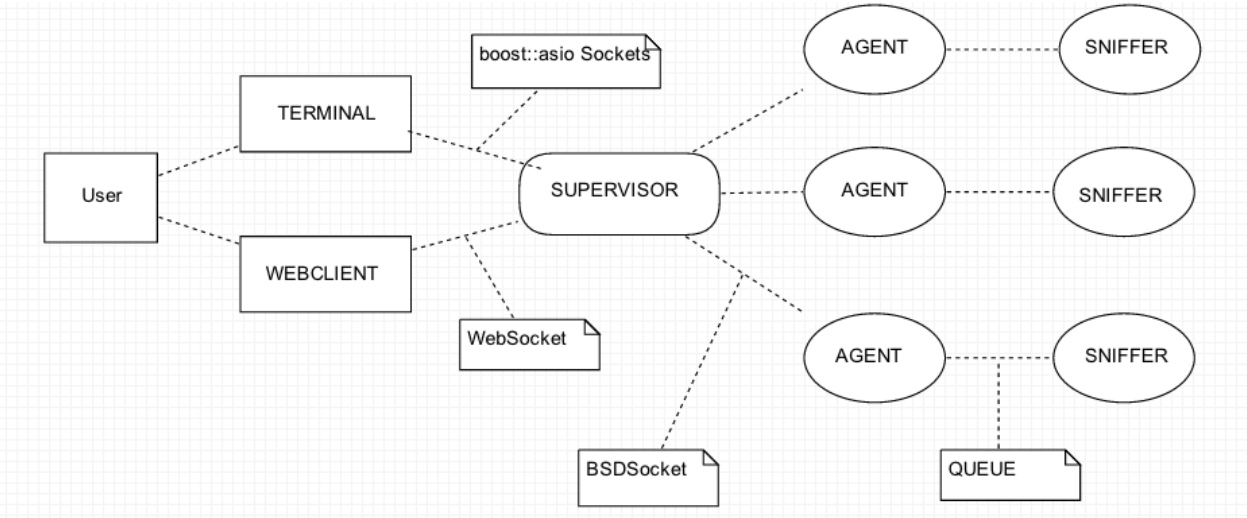
Deinstalacja:

* scons -c (wywołane z poziomu katalogu głównego z plikami)

Uruchamianie:

* Supervisor: ./build/debug/supervisor/supervisor
* Terminal: ./build/debug/terminal/terminal
* Agent: sudo ./build/debug/agent/agent
* Webclient: Uruchomienie stront localhost:9000 w przeglądarce (nalezy pamiętać o wczesniejszym uruchomieniu polecenia grunt server, patrz instrukcja)

6. Diagram komunikacji



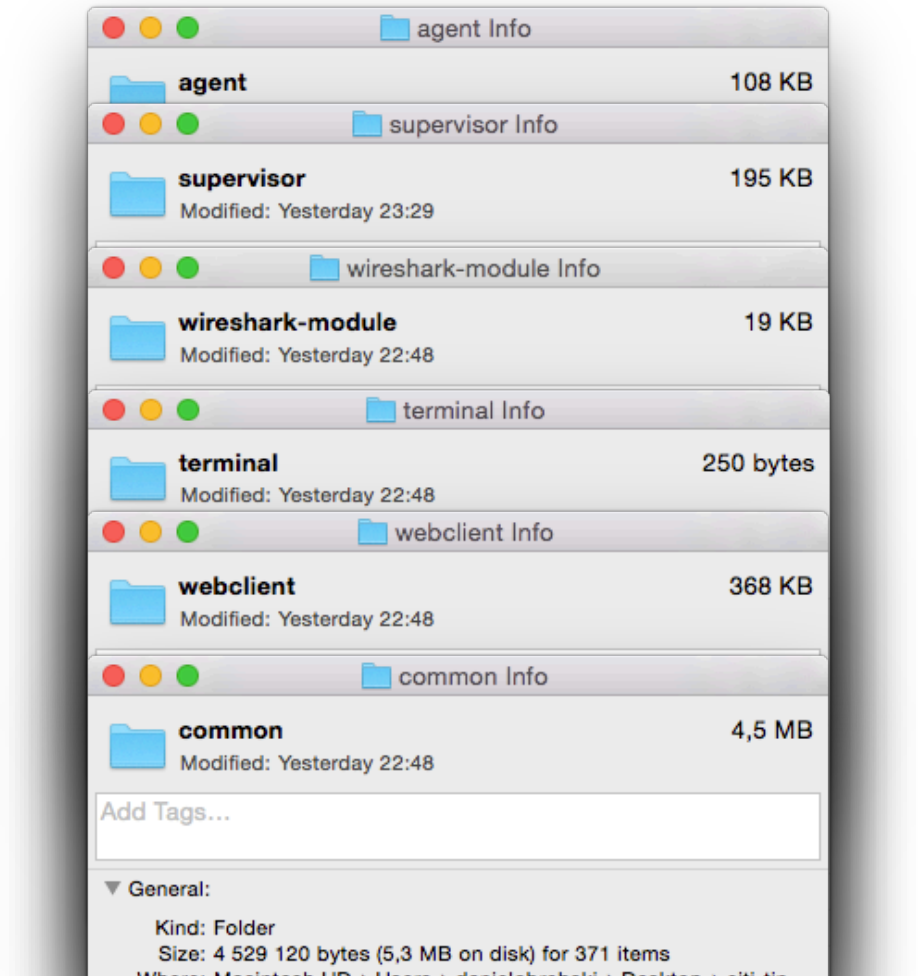
7. Podsumowanie

7.1 Wyciągnięte doświadczenia

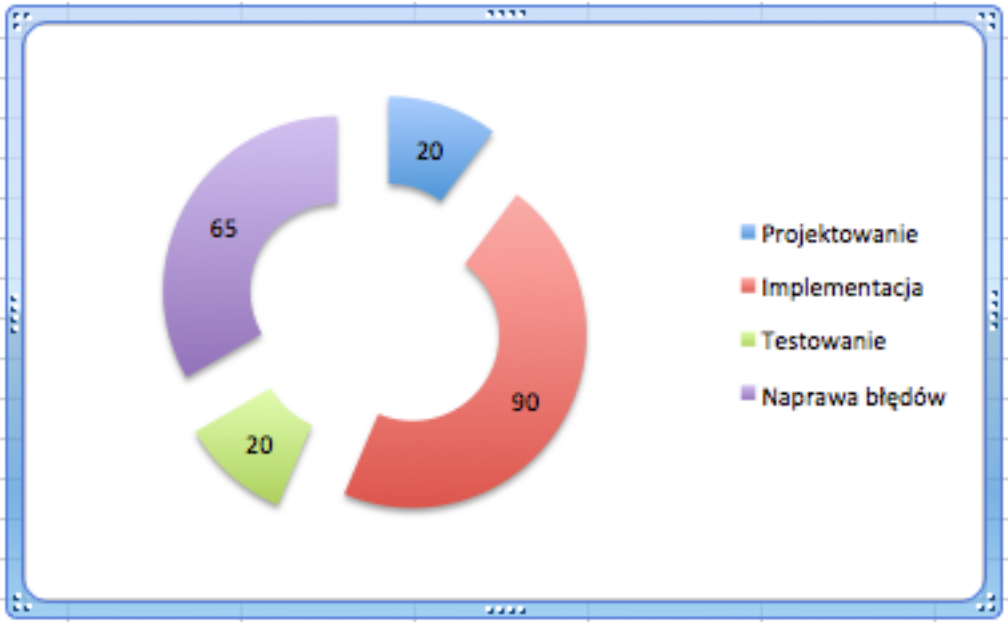
Bez systemu kontroli wersji git – projektu nie udałoby się zrealizować, zajmuje on dużo czasu. Warto zazanajomić wszystkich członków zespołów ze strukturą zdalnego repozytorium, oraz jego obsługą na początku projektu.
Dobra dokumentacja wstępna oraz w miarę równomierny podział obowiązków jest kluczem do sukcesu.
W przypadku odłożenia projektu na czas późniejszy, jego wykonanie nie byłoby możliwe.
Stworzona aplikacja pozwoliła nam nauczyć się wielu zagadnień w zakresie komunikacji, czy to sieciowej czy międzyprocesowej.

7.2 Rozmiar plików

Wylistowanie pojedyńczych plików z uwagi na ich ilość byłoby redundantne, dlatego umieszczona jest informacja na temat rozmiarów całych modułów.



7.3 Oszacowanie czasu pracy (dane podane w godzinach)



7.4 Użyte technologie i wykorzystany kod

- Do nasłuchiwania ruchu w sieci skorzystaliśmy z biblioteki libpcap.
- BSD Sockets
- JavaScript – Backbone.js, Twitter Bootstrap, jQuery, API WebSockets
- C++ Boost
- Skorzystaliśmy z ogólno dostępnej biblioteki do obsługi JSON w C++ : https://github.com/nlohmann/json
- Websocketpp https://github.com/zaphoyd/websocketpp
- Concurrent-queue https://github.com/juanchopanza/cppblog/blob/master/Concurrency/Queue/Queue.h
- Własny kod z zajęć ZPR