WYDZIAŁ PODSTAWOWYCH PROBLEMÓW TECHNIKI

POLITECHNIKA WROCŁAWSKA

WIZUALIZACJA POŁĄCZEŃ

SIECI INTERNETOWEJ

MARCIN ADAMCZYK

NR INDEKSU: 221 429

Praca inżynierska napisana

pod kierunkiem

Dr. Przemysława Kobylańskiego



WROCŁAW 2017

Spis treści

Wstęp

1. Analiza wymagań
2. Projekt systemu
   1. Grupy użytkowników i założenia
   2. Przypadki użycia i scenariusze
   3. Diagramy aktywności
   4. Diagramy stanów
   5. Opis protokołów
3. Implementacja systemu
   1. Opis technologii
   2. Omówienie metodologii
   3. Omówienie kodów źródłowych
   4. Opis algorytmów
   5. Synchronizacja wątków
   6. Wizualizacja
4. Podsumowanie

Bibliografia

Dodatek

Instalacja i wdrożenie

Zawartość płyty

Wstęp

Celem pracy jest stworzenie oprogramowania, które poprzez nasłuchiwanie ruchu sieciowego oraz jego analizę, naniesie na mapę świata punkty, w których znajdują się urządzenia biorące udział w transmisji pakietów użytkownika. W efekcie powstanie wizualizacja, mająca pomóc osobom niezaznajomionym z sieciami komputerowymi w zrozumieniu pewnych podstawowych zasad oraz metod dotyczących działania Internetu. System będzie działał w czasie rzeczywistym, dzięki czemu użytkownik będzie mógł na bieżąco obserwować wpływ jego działań na urządzenia rozlokowane na świecie.

Swoim zakresem praca obejmuje szereg zagadnień powiązanych z Internetem oraz sieciami komputerowymi. Spośród nich szczególnie ważnym jest *routing*, czyli wyznaczanie trasy, za pomocą której pakiet ma dotrzeć do celu. Do analizy sieci wykorzystane zostały narzędzia takie jak *Wireshark* i *traceroute.* Dodatkowo system wykorzystuje usługę geolokalizacji na podstawie adresu IP.

Obecnie nie ma oprogramowania udostępniającego podobną funkcjonalność. Istnieje serwis internetowy pokazujący na mapie świata trasę pojedynczego pakietu, jednak nie ma to zastosowania w przypadku chęci analizowania ruchu w czasie rzeczywistym. Samo połączenie programów *Wireshark* i *traceroute* może dać efekty w postaci wydruku listy adresów IP - nieczytelnego dla zakładanego użytkownika oprogramowania.

Praca zawiera:

* omówienie zagadnień związanych z Internetem oraz sieciami komputerowymi poruszonych podczas tworzenia oprogramowania,
* informacje na temat dodatkowych narzędzi wykorzystywanych do działania systemu,
* informacje na temat stworzonego oprogramowania: dokumentację, zasadę działania, schematy UML, fragmenty kodów źródłowych,
* szczegółową instrukcję instalacji opisywanego oprogramowania, jak również całego środowiska niezbędnego do prawidłowego jego działania.

1. Analiza wymagań

W niniejszym rozdziale przedstawiono środowisko pracy systemu oraz omówiono podstawowe pojęcia z tym środowiskiem powiązane. Następnie wyjaśniono procesy zachodzące w programie oraz powiązania pomiędzy tymi procesami. Dalsza część zawiera opis założeń funkcjonalnych i niefunkcjonalnych przedstawianego systemu. Na koniec przeprowadzono analizę oprogramowania trzeciego, używanego przez system, jak również porównanie do innych programów dostępnych dla użytkownika, realizujących podobne zadania.

1.1 Opis środowiska

Zadaniem opisywanego oprogramowania jest wizualizacja połączeń między urządzeniami wchodzącymi w skład Internetu, biorących udział w transmisji pakietów użytkownika. Zanim przedstawione zostanie jak dokonuje się sama wizualizacja, warto omówić co tak naprawdę jest jej poddawane. Jak można przeczytać w książce Tannenbauma:

*„Aby osiągnąć swoje cele, warstwa sieciowa musi znać topologię sieci (tzn. zbiór wszystkich routerów i połączeń między nimi) i wybierać odpowiednią trasę poprzez tę sieć, niezależnie od jej rozmiarów. Musi też tak dobierać trasy, aby unikać przeciążenia części linii komunikacyjnych i routerów, pozostawiając inne bez pracy.”*

Proces ten nosi nazwę routingu. Mając topologię sieci w postaci skierowanego grafu z wagami, warstwa sieciowa dla każdego routera (wierzchołka grafu) musi wyznaczyć najkrótsze ścieżki do każdego innego routera w sieci, tworząc dla niego tak zwane „drzewo ujścia” [pojęcie z Tannenbauma]. Z owego drzewa konstruowana jest (również dla każdego routera osobno) tablica routingów, mówiącą do którego z sąsiednich routerów należy przekazać pakiet, aby dotarł do celu.

Na kształt wyznaczonego drzewa poza fizycznym czasem przesyłu pakietu pomiędzy wierzchołkami, wpływ ma kilka dodatkowych czynników. Globalność Internetu wiąże się z problemami natury politycznej. Niektóre kraje nie życzą sobie, aby infrastruktura znajdująca się na ich terenie służyła do przesyłu danych z innego kraju. Podobnie sytuacja wygląda z wielkimi korporacjami, do których należą owe sieci. Mogą one żądać wysokich opłat przesyłowych, przez co bardziej opłacalne może być korzystanie z nieoptymalnej pod względem czasu przesyłu, ale bardziej opłacalnej finansowo trasy.

Wspomniane do tej pory czynniki są niezmienne (ewentualnie ulegają zmianom stosunkowo rzadko). Zdecydowanie częściej zmienia się sama topologia sieci. Urządzenia nie są niezawodne, a kable transmisyjne nie mają nieograniczonej przepustowości. Algorytmy routingowe na muszą bieżąco uwzględniać zmiany topologii i aktualizować tablice routingów.

W przypadku Internetu, wielkość sieci sprawia, że niemożliwe jest, aby każdy router przechowywał i przetwarzał dane na temat całej topologii. Problem ten rozwiązano wykorzystując istniejący podział Internetu na tzw. „sieci autonomiczne”. Pozwala on na użycie dwóch oddzielnych algorytmów routingowych. Pierwszy z nich (*Algorytm bram wewnętrznych* [Tannenbaum]) działa wewnątrz sieci autonomicznej, należącej do jednej korporacji, znajdującej się na terenie jednego państwa, dzięki czemu nie musi on uwzględniać ograniczeń natury politycznej i może się skupić na wyznaczeniu najszybszych i najbardziej niezawodnych połączeń. Sama liczba urządzeń jest mocno ograniczona (w stosunku do całego Internetu), więc obliczanie i aktualizowanie tablic routingów nie jest tak wymagające pod względem zasobów. Drugi algorytm (*Algorytm bram zewnętrznych* [Tannenbaum]) patrzy na topologię sieci z wyższego poziomu – rolę wierzchołków grafu pełnią wspomniane wcześniej sieci autonomiczne. Ponieważ w tym wypadku linie transmisyjne na trasie mogą należeć do różnych korporacji, algorytm musi uwzględniać takie czynniki jak koszty wykorzystania łącz. Dodatkowo, w zależności od sytuacji politycznej niektóre połączenia międzynarodowe mogą być blokowane.

Połączenie tych dwóch algorytmów daje możliwość kierowania pakietów Internetowych odpowiednimi trasami i to właśnie ich działanie poddawane jest wizualizacji w opisywanym systemie.

1.2 Opis procesów

System składa się z trzech głównych procesów: **wczytywanie**, **analiza**, **wyświetlanie** (w dalszych częściach pracy komponenty realizujące te zadania nazywane są serwisami). Ogólny sposób działania procesów został opisany poniżej.

1. Wczytywanie danych

Bazą tego procesu jest *Wireshark* - program potrafiący nasłuchiwać ruch sieciowy. Jego zadaniem jest przechwytywanie pakietów TCP wychodzących z urządzenia użytkownika i wyłuskanie z nich docelowych adresów IP. Dane te przekazywane są przy pomocy unixowego pipe’a na wejście głównego programu. Ten po sprawdzeniu poprawności danych pod względem strukturalnym, dokonuje odfiltrowania tak zwanych adresów zastrzeżonych. Na koniec gotowe dane przekazuje do następnego procesu.

1. Analiza danych

Proces ten odpowiada za przechowywanie oraz przetwarzanie danych na temat tworzonego grafu połączeń sieciowych. Dla każdego wczytanego z wejścia adresu IP uruchamiany jest program *traceroute*, co pozwala otrzymać listę adresów IP urządzeń pośredniczących w komunikacji. Dane na temat nawiązanych połączeń (krawędzi grafu) przechowywane są w drzewiastej strukturze. Zawierają one informacje na temat łącznej ilości pakietów przesłanych przy pomocy tego połączenia oraz ilości pakietów przesłanych w określonym odcinku czasu. Statystyki przechowywane są jako obiekt osobnej klasy, dzięki czemu uzyskano izolację logiki związanej z przetwarzaniem danych statystycznych, a co za tym idzie, swobodną rozszerzalność typów informacji jakie mają być gromadzone i przetwarzane.

Oprócz danych na temat połączeń (krawędzi grafu), przechowywane są też dane na temat konkretnych adresów IP (wierzchołków grafu). Przy każdym pojawieniu się nowego (nie zarejestrowanego do tej pory) adresu IP, uzyskiwana jest informacja na temat lokalizacji geograficznej danego adresu. W tym celu wykorzystano usługę geolokalizacyjną w postaci API serwisu *ip-api.com*.Jeśli otrzymane dane są poprawne, proces zapisuje je w mapie.

Jako ostatni krok, wszystkie dane, które powinny zostać uwzględnione w wizualizacji zostają opakowane w odpowiednią strukturę i przekazane do następnego procesu.

1. Wyświetlanie obrazu wynikowego

Zadaniem tej części systemu jest przetworzenie danych odebranych z wejścia celem wyświetlenia ich w odpowiedniej formie na ekranie. Proces ten nie przeprowadza żadnej analizy – zakłada, że to co odebrał, jest informacją prawidłową i gotową do wyświetlenia. Dokonywana jest tu przede wszystkim konwersja współrzędnych geograficznych na położenie na ekranie (piksele). Dodatkowo na podstawie opcjonalnych danych przekazanych przez analizator, mogą zostać nałożone dodatkowe efekty wizualne (przykładowo: pogrubienie lub zmiana koloru wyświetlanej linii w przypadku połączenia o wysokim natężeniu ruchu).

1.3 Założenia systemu

Podstawowym założeniem dotyczącym funkcjonalności systemu, jest jego działanie   
w czasie rzeczywistym. Ma na celu to zapewnienie użytkownikowi wrażenia wyższej responsywności używanego oprogramowania. Aby uniknąć wstrzymywania pracy programu w przypadku nieprzewidzianych opóźnień związanych z korzystaniem z zewnętrznego API oraz analizą sieci, oprogramowanie wykorzystuje wielowątkowość. Konkretnie, każdy z opisanych w poprzednim podrozdziale serwisów ma działać jako osobny wątek. Komunikacja pomiędzy nimi, odbywa się przez umieszczanie wyjścia jednego serwisu w buforze będącym jednocześnie wejściem dla drugiego serwisu. Dla zachowania bezpieczeństwa, jako bufory wykorzystano nieblokujące kolejki FIFO. Uproszczony schemat przedstawiający komunikację między procesami przedstawiony jest na *rysunku 1.1*.

F

I

F

O

F

I

F

O

ANALIZA

Wątek 2

WYŚWIETLANIE

Wątek 3

WCZYTYWANIE

Wątek 1

*Rysunek 1.1 – komunikacja między procesami*

1.4 Porównanie do istniejącego oprogramowania

Każdy z wymienionych wcześniej programów wykorzystanych do stworzenia projektu sam z siebie realizuje jakąś część jego funkcjonalności. Jednak w przypadku analizatorów (*Wireshark, traceroute*), ich wyjście samo z siebie jest zupełnie nieczytelne dla przeciętnego użytkownika.

Istnieją serwisy internetowe oferujące możliwość przeprowadzenia wizualizacji trasy pojedynczego pakietu. Jednak nie ma to zastosowania chcąc uzyskać podgląd w czasie rzeczywistym.

2. Projekt systemu

W tym rozdziale przedstawiono opis systemu używając schematów blokowych oraz diagramów UML.

2.1 Grupy użytkowników i założenia

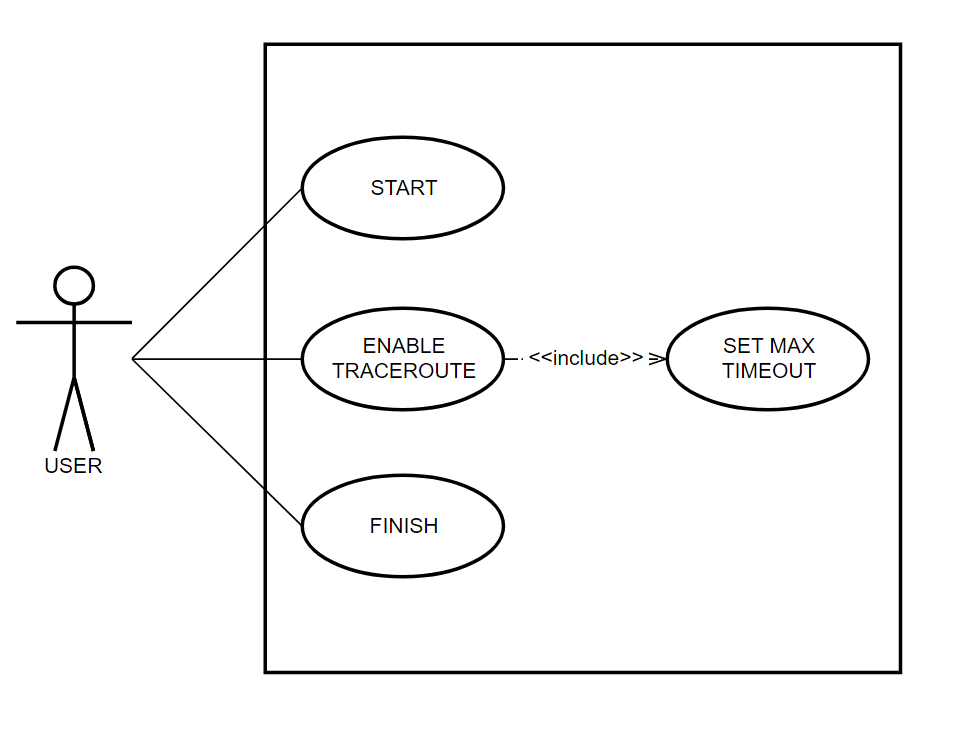
System ma pełnić rolę narzędzia edukacyjnego, a jako grupę odbiorców przyjmuje się osoby chcące poznać architekturę oraz zasady działania sieci komputerowych. System ma spełniać następujące założenia:

* Działający w czasie rzeczywistym,
* Wielowątkowy,
* Podzielony na odseparowane komponenty, komunikujące się za pomocą ustalonych protokołów.

Sposób realizacji wymienionych założeń, został dogłębniej przedstawiony w dalszych podrozdziałach.

* 1. Przypadki użycia i scenariusze

Mimo, iż program sam w sobie zaprojektowany jest do działania „w tle”, bez interakcji użytkownika, ten ma możliwość wybrania trybu w jakim będzie pracować. Dokładnie, użytkownik może (w zależności od tego jaki efekt chce uzyskać) włączyć obsługę programu *traceroute* (domyślnie jest on wyłączony). Jeśli tak zrobi, może on także ustawić parametr mówiący o maksymalnym czasie oczekiwania na odpowiedź przez program *traceroute*. W przypadku gdy użytkownikiem jest osoba nie mająca wiedzy wystarczającej, by określić jaką wartość powinien mieć dany parametr, oprogramowanie wykorzysta własną, domyślną wartość. Ostatecznie do użytkownika należą również dwie dodatkowe akcje: uruchomienie oraz zakończenie pracy systemu.

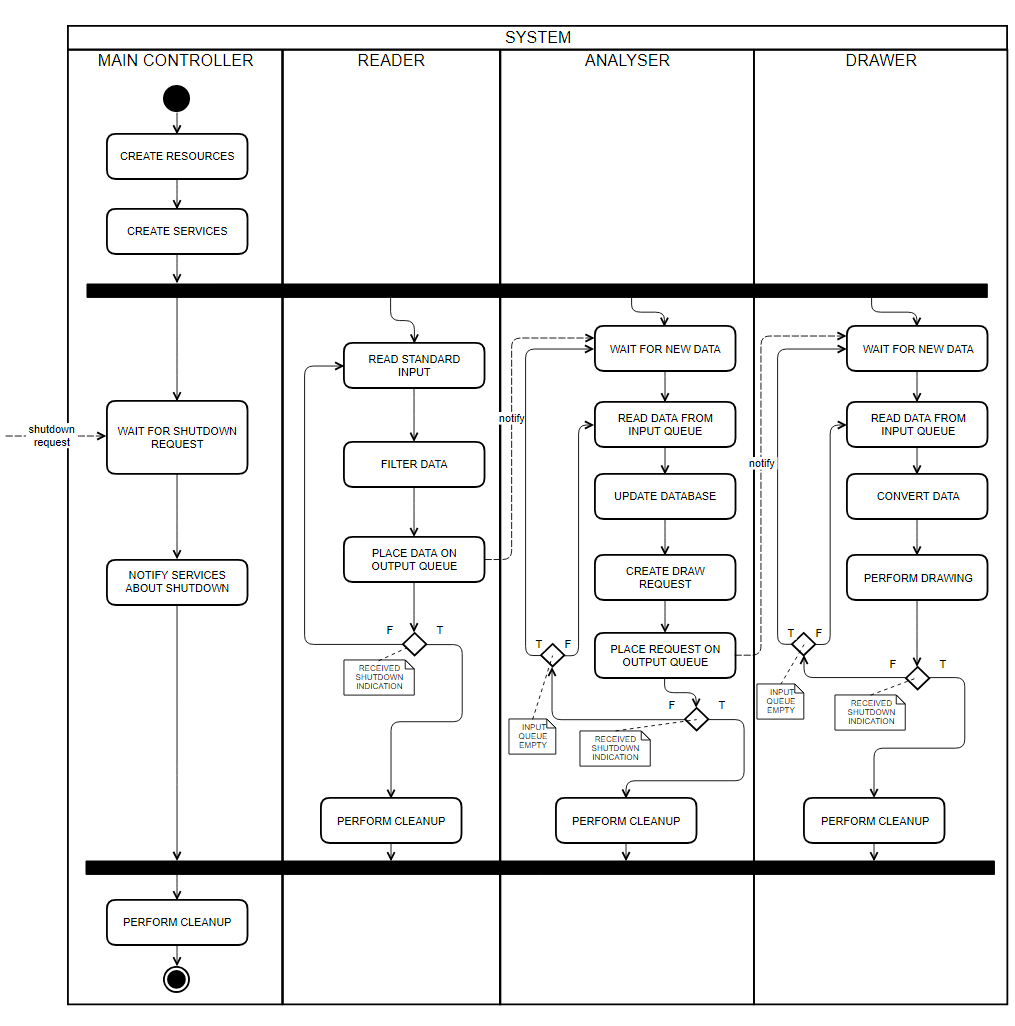


*Rysunek 2.1 – diagram przypadków użycia*

* 1. Diagram aktywności

Diagram na *rysunku 2.2* przedstawia przepływ sterowania oraz wysokopoziomowy podgląd na sposób realizacji zadań przez poszczególne komponenty.

Całością pracy systemu zarządza kontroler. Do jego zadań należą utworzenie środowiska pracy systemu (stworzenie zasobów w postaci buforów komunikacyjnych oraz serwisów działających jako osobne wątki), jak również zapewnienie, że w momencie otrzymania od użytkownika polecenia zakończenia pracy systemu, wszystkie zasoby zostaną poprawnie zwolnione. Każdy z serwisów wykonuje swoją pracę do momentu otrzymania wspomnianego polecenia zakończenia pracy. Komunikacja między serwisami została ograniczona do minimum w celu zmaksymalizowania korzyści wynikających z wielowątkowości.

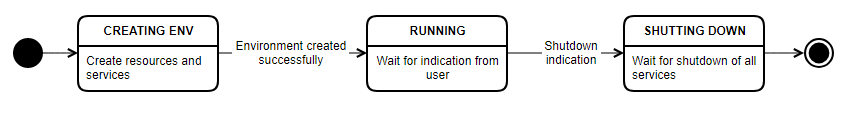


*Rysunek 2.2 – diagram aktywności*

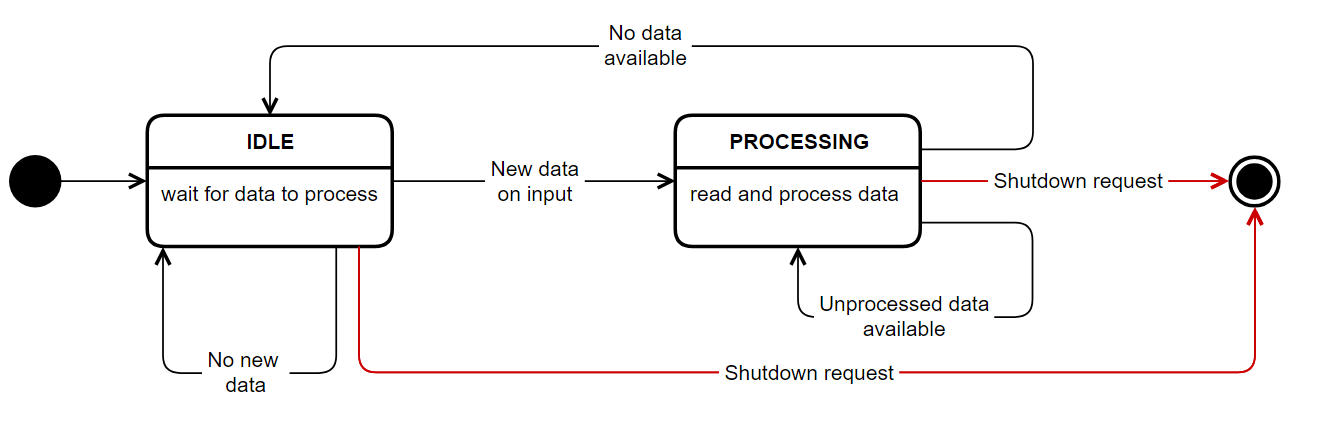
2.4 Diagramy stanów

Poniżej przedstawiono diagramy stanów dla poszczególnych komponentów systemu. Komponenty zostały podzielone na dwie kategorie: serwisy i kontroler.

*Rysunek 2.3* przedstawia diagram stanów w jakich znajdować się może komponent kontrolera systemu. Po stworzeniu środowiska, przechodzi on w stan oczekiwania na polecenie użytkownika. Po otrzymaniu polecenia, wykonuje operacje zakończenia działania systemu.

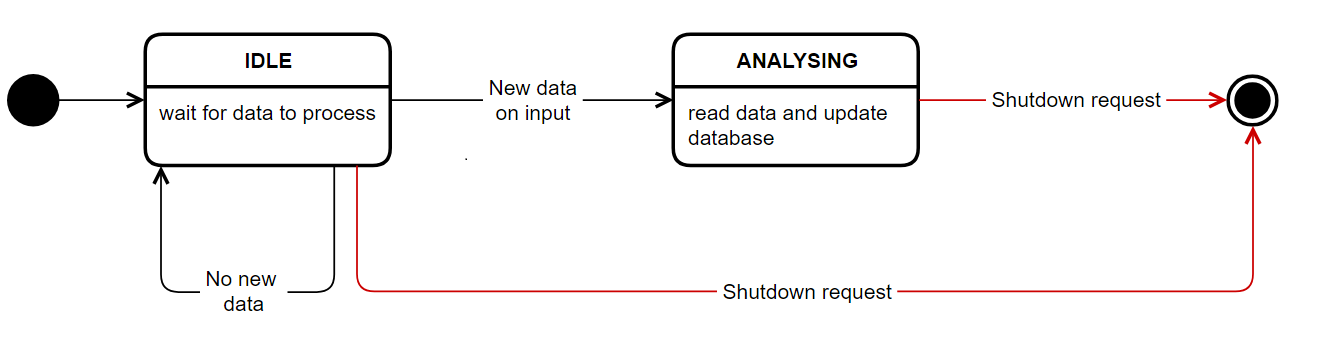


*Rysunek 2.3 – diagram stanów dla kontrolera*

*Rysunek 2.4* przedstawia diagram stanów w jakich mogą znajdować się serwis wczytujący dane oraz serwis rysujący. Pracują one tylko wtedy gdy otrzymują nowe dane do przetworzenia. W przeciwnym wypadku przechodzą w stan bezczynności, oczekując na nowe dane. 

*Rysunek 2.4 – diagram stanów dla serwisu wczytującego oraz rysującego*

*Rysunek 2.5* przedstawia diagram stanów w jakich może się serwis analizujący dane. W porównaniu do pozostałych serwisów, ten nie przechodzi w stan bezczynności – stale sprawdza dane na temat czasów wysłania pakietów, przesyłając do serwisu rysującego zaktualizowane dane.



*Rysunek 2.5 – diagram stanów dla serwisu analizującego*

2.5 Opis protokołów

W opisywanym systemie komunikacja pomiędzy komponentami odbywa się poprzez przekazanie przez jeden serwis odpowiednio skonstruowanej wiadomości do bufora i odebranie tej wiadomości przez inny serwis. Można wyróżnić dwa protokoły komunikacji:

1. Serwis wczytujący → Serwis analizujący

Wiadomość ma postać łańcucha znaków (std::string) zawierającego adres IP.

1. Serwis analizujący → Serwis rysujący

Wiadomość zawiera wskaźnik do obiektu przechowującego listę danych na temat krawędzi grafu, które mają być zwizualizowane przez serwis rysujący. Pojedynczy element listy zawiera współrzędne (długość geograficzną oraz szerokość geograficzną) punktów, które są połączone przez daną krawędź, a także kolor jaki ma być nadany tej krawędzi zakodowany przy pomocy trzech wartości (model RGB). Strukturę takiej wiadomości przedstawiono w kodzie źródłowym 2.1.

Kod źródłowy 2.1: struktura wiadomości protokołu komunikacyjnego pomiędzy serwisem analizującym i rysującym

DrawingPackage.hpp

1. ...
2. struct CurveData
3. {
4. // coordinates of first node
5. float lat1;
6. float lon1;
7. // coordinates of second node
8. float lat2;
9. float lon2;
10. // RGB coded colour
11. short r = 0;
12. short g = 0;
13. short b = 0;
14. };
15. ...

3. Implementacja systemu

W tym rozdziale przedstawiono sposób, w jaki system został zaimplementowany. Na podstawie kodów źródłowych omówiono nowoczesne techniki programistyczne oraz metodyki mające na celu poprawić niezawodność oraz czytelność kodu. Dla poprawienia przejrzystości omawianych fragmentów, pominięto nieistotne części kodu. Brakujące fragmenty oznaczono przy pomocy trzech kropek („...”).

3.1 Opis technologii

Do implementacji systemu użyto języka C++ w wersji określanej jako C++17. Zdecydowano się na ten właśnie język ze względu na możliwość uzyskania wysokiej wydajności przy jednoczesnym zachowaniu czytelności kodu. Oprócz standardowej biblioteki języka C++ wykorzystano także dwie zewnętrzne biblioteki:

1. Boost: biblioteka zawierająca algorytmy i struktury danych. Wykorzystano następujące elementy:
   * Nieblokująca kolejka FIFO, użyta do stworzenia buforów komunikacyjnych pomiędzy serwisami (boost::lockfree::spsc\_queue),
   * Wartość opcjonalną (boost::optional) używaną do uzyskania wartości, gdy nie jest pewne czy jakakolwiek wartość zostanie zwrócona.
2. Qt: biblioteka graficzna, użyta do ostatecznej wizualizacji. Wykorzystano następujące elementy:
   * QWidget
   * QPixmap,
   * QTimer,
   * QPainter,
   * QPainterPath.

3.2 Omówienie metodologii

W celu uniknięcia niebezpieczeństw związanych z operowaniem na pamięci w języku C++ (ręczna alokacja, przekazywanie wskaźników), podczas implementowania systemu, zastosowano technikę RAII (Resource Acquisition Is Initialization). Jak możemy wyczytać w dokumentacji języka C++:

*„RAII jest techniką programowania wiążącą cykl życia zasobu, który musi być uzyskany przed użyciem (zaalokowana pamięć na stercie, wykonywany wątek, otwarty port, otwarty plik, zablokowany mutex, przestrzeń na dysku, połączenie z bazą danych – cokolwiek, co istnieje w ograniczonej ilości), z czasem istnienia obiektu.”*

W praktyce sprowadza się to do „opakowania” zasobu, czyli stworzenia klasy, której polem będzie sam uchwyt do zasobu, a sam zasób [3x zasób] będzie tworzony w konstruktorze i niszczony w destruktorze klasy. Dodatkowe zaimplementowanie konstruktora kopiującego oraz konstruktora przenoszącego pozwalają na swobodne zarówno współdzielenie zasobu, jak i przekazywanie jego własności. Przykłady użycia techniki RAII w opisywanym systemie przedstawione będą w sekcji następnej.

3.3 Przegląd kodu źródłowego

Kod źródłowy [3.1](#kod41) pokazuje zastosowanie techniki RAII przy użyciu dostarczonych przez Bibliotekę Standardową języka C++ tzw. Inteligentnych wskaźników (ang. Smart Pointers). W momencie wyjścia z zakresu, obiekt wskaźnika zostanie zniszczony, a co za tym idzie zostanie wywołany jego destruktor, który jest odpowiedzialny za poprawne zwolnienie przechowywanego zasobu. Inteligentne wskaźniki, dzięki wspomnianym wcześniej, dobrze zaimplementowanym konstruktorom kopiującym i przenoszącym, pozwalają na kontrolę nad istnieniem zasobu także przy przekazywaniu i współdzieleniu własności. Dla przykładu, std::shared\_ptr (linie 4 – 5) posiada licznik referencji, dzięki czemu dopóki jakikolwiek obiekt korzysta z zasobu, można być pewnym, że ten nie zostanie zwolniony. W tym wypadku został on użyty do przechowania obiektu bufora komunikacyjnego, który jest współdzielony przez komponenty. Z kolei std::unique\_ptr (linie 7 – 9) zapewnia, że istnieje tylko jedno odwołanie do danego zasobu. W omawianym systemie użyto tej własności do przechowywania instancji serwisów.

Kod źródłowy 3.1: fragment kontrolera, przykład użycia techniki RAII

MainController.cpp

1. ...
2. void MainController::run(QApplication& app)
3. {
4. string\_queue\_ptr queue\_1 = make\_shared<string\_queue\_type>(1000);
5. drawings\_queue\_ptr queue\_2 = make\_shared<drawings\_queue\_type>(1000);
6. unique\_ptr<Reader> reader(new Reader(queue\_1));
7. unique\_ptr<Analyser> analyser(new Analyser(queue\_1, queue\_2));
8. unique\_ptr<Drawer> drawer(new Drawer(queue\_2, app));
9. }
10. ...

Kod źródłowy [3.2](#kod42) przedstawia bezpieczne zarządzanie zasobem, jakim jest wątek. Nie została tu wykorzystywana żadna pomocnicza klasa z biblioteki standardowej – jest to własna implementacja wykorzystująca technikę RAII. W konstruktorze klasy wykonywana jest operacja stworzenia i uruchomienia nowego wątku (linia 8 pliku Analyser.cpp). Destruktor odpowiada za poprawne zakończenie działania wątku (linia 15 pliku Analyser.cpp).

Kod źródłowy 3.2: fragment serwisu analizatora, przykład użycia techniki RAII

Analyser.hpp:

1. ...
2. class Analyser
3. {
4. public:
5. Analyser(
6. string\_queue\_ptr& inputQueue,
7. drawings\_queue\_ptr& outputQueue);
8. ~Analyser();
9. void run();
10. private:
11. std::thread internalThread\_;
12. ...
13. };
14. ...

Analyser.cpp:

1. ...
2. Analyser::Analyser(
3. string\_queue\_ptr& inputQueue,
4. drawings\_queue\_ptr& outputQueue)
5. : inputQueue\_(inputQueue)
6. , outputQueue\_(outputQueue)
7. , threadRunning\_(true)
8. , internalThread\_(&Analyser::run, this)
9. {
10. logger\_ = "\n[ANALYSER]\n ";
11. }
12. Analyser::~Analyser()
13. {
14. internalThread\_.join();
15. }
16. void Analyser::run()
17. {
18. ...
19. }
20. ...

3.4 Opis algorytmów

Serwis analizujący w przeciwieństwie do pozostałych dwóch serwisów, nie przechodzi w stan bezczynności gdy nie otrzymuje nowych danych. Aby kontrolować i poddawać wizualizacji obciążenie poszczególnych połączeń (krawędzi grafu), algorytm analizatora stale uaktualnia dane na temat ilości pakietów, które zostały wysłane przy pomocy każdego z połączeń. Za pomocą *Pseudokodu 3.1* przedstawiono przy pomocy pseudokodu sposób działania algorytmu.

while (true)

if (new\_data\_available)

run\_traceroute\_for\_new\_data()

for each (IP in traceroute\_result)

if (IP\_not\_present\_in\_locations\_map)

get\_IP\_location()

end\_if

end\_for

for each (consecutive\_IP\_pair in traceroute\_result)

add\_timestamp\_in\_link\_data()

end\_for

end\_if

for\_each (link\_data in link\_data\_map)

delete\_timed\_out\_timestamps()

end\_for

end\_while

*Pseudokod 3.1 – algorytm analizy danych*

W przypadku gdy na wejściu pojawią się dane (docelowy adres IP), analizator uruchamia dla tych danych program *traceroute*, po czym dla wszystkich uzyskanych połączeń (par kolejnych adresów IP), w mapie przechowującej informacje na temat połączeń, dodawana jest informacja na temat czasu, kiedy pakiet został wysłany. Dodanie następuje przez wstawienie na koniec listy czasów (w tym wypadku rolę listy pełni std::vector<std::chrono::system\_clock::time\_point>) punktu w czasie odnoszącego się do chwili wysłania pakietu.

Zawsze, niezależnie od tego czy na wejściu są dostępne dane, analizator uaktualnia informacje na temat zapisanych czasów wysyłania pakietów przez połączenia. Polega to na usuwaniu znaczników czasowych, które są starsze niż wartość ustalona parametrycznie.

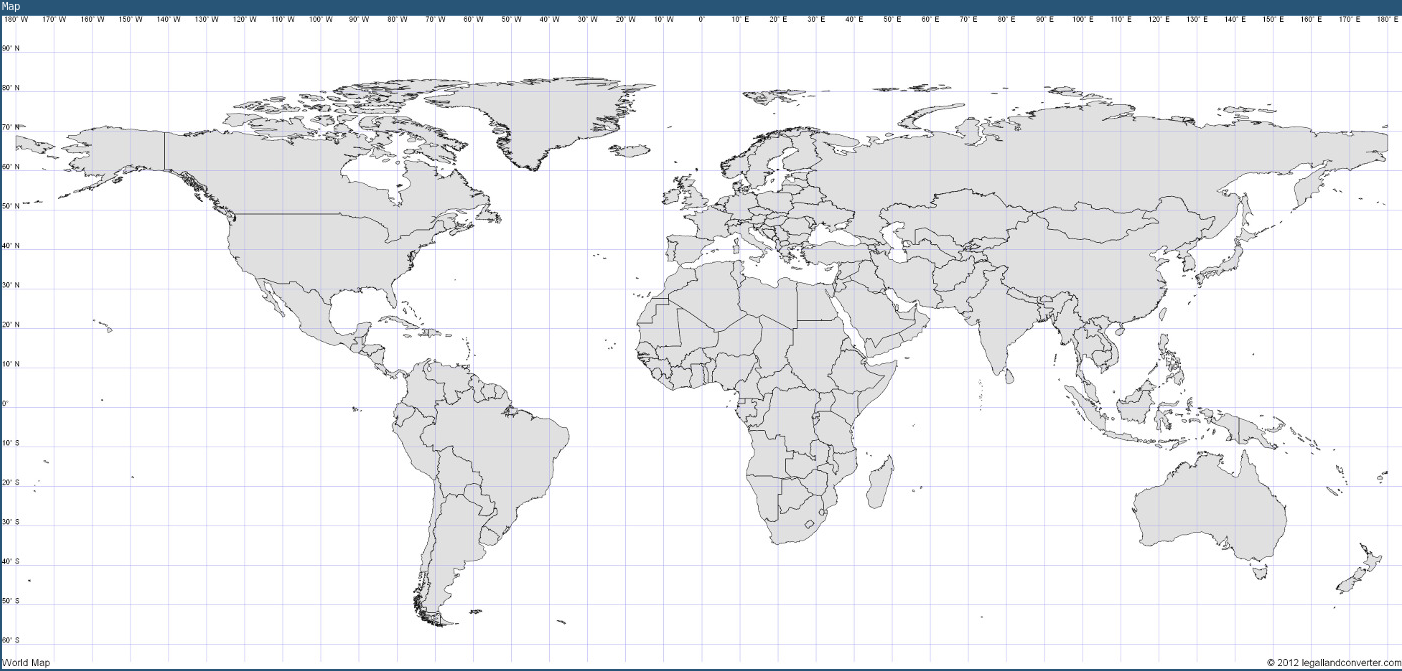
3.5 Synchronizacja wątków

W wielowątkowym kodzie komunikacja pomiędzy wątkami musi być synchronizowana. W omawianym systemie funkcja ta realizowana jest przez nieblokujące kolejki z biblioteki Boost. Wykorzystano kolejkę spsc\_queue (spsc: single producer single consumer), zaprojektowaną do obsługi jednego wątku dodającego dane do kolejki i jednego wątku te dane zdejmującego, co odpowiada modelowi opisywanego oprogramowania.

3.6 Wizualizacja

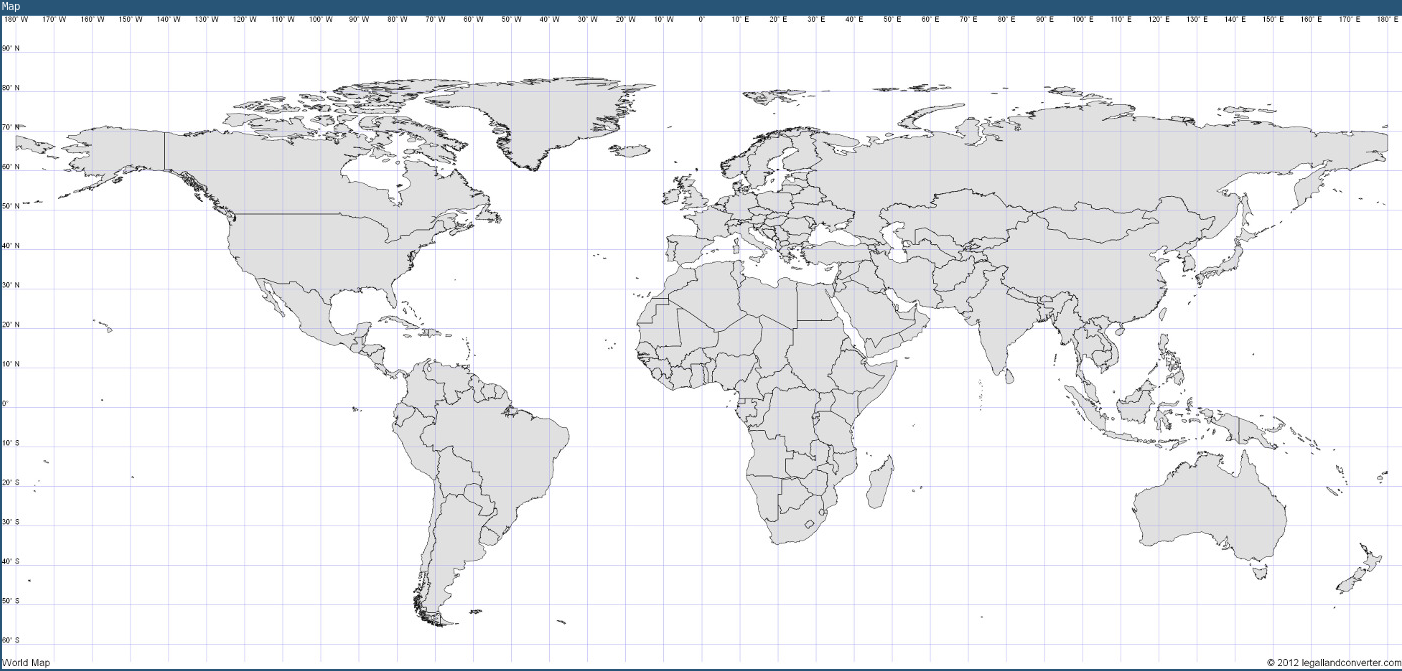
W tej sekcji zamieszczono serię zrzutów ekranu przedstawiających działanie stworzonego oprogramowania.

Zrzut 3.1 przedstawia stan początkowy wizualizacji – czystą konturową mapę świata.



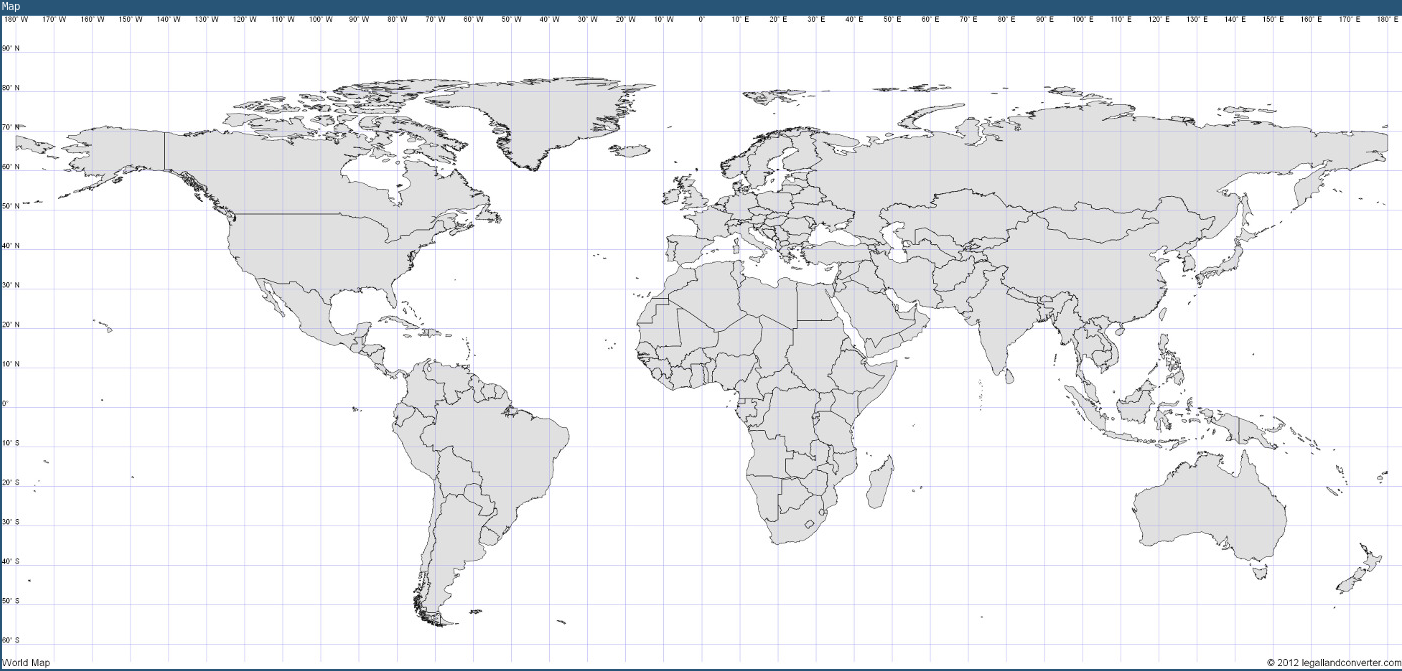
*Zrzut 3.1 – stan początkowy*

Zrzut 3.2 przedstawia wizualizację połączeń regularnym natężeniu - wszystkie linie są jednakowego, niebieskiego koloru.



*Zrzut 3.2 – uregulowany ruch*

Zrzut 3.3 przedstawia wizualizację połączeń o średnim i wysokim natężeniu. Są one oznaczone kolorami odpowiednio żółtym oraz czerwonym.



*Zrzut 3.3 – chwilowe wysokie obciążenie*

4. Podsumowanie

Zaimplementowane zostały wszystkie założone funkcjonalności związane z analizą sieci. Oprogramowanie wczytuje dane, przetwarza je i wyświetla w odpowiedniej formie. Nie zostały natomiast zaimplementowane opcje interakcji z użytkownikiem. Po uruchomieniu, program nie przyjmuje żadnych akcji z jego strony.

W przypadku rozwoju projektu, programy *Wireshark* oraz *traceroute* zastąpione zostaną własnymi implementacjami:

* Program *Wireshark* w pełni realizuje wyznaczone zadanie, jednak jako zaawansowany program do analizy sieci, zużywa więcej zasobów, niż program napisany własnoręcznie przy użyciu bibliotek *libnet oraz libpcap,* realizujący jedynie wymagane zadania.
* W przypadku programu *traceroute*, w sytuacjach gdy występują opóźnienia w otrzymaniu wyniku, brakuje informacji zwrotnej co je powoduje. Własna implementacja (ponownie przy użyciu bibliotek *libnet* i *libpcap*) pozwoliłaby na mniejsze straty danych, dzięki możliwości lepszego dobierania maksymalnego czasu oczekiwania na odpowiedź routerów. Opcjonalnym rozwiązaniem jest uwspółbieżnienie wywołań programu *traceroute*.

Dodatkowo, funkcjonalnością wymagającą rozwinięcia jest wyświetlanie danych. W obecnej formie, użytkownik nie ma możliwość przybliżenia obrazu (mapy świata), co w przypadku gęstego ruchu, obniża czytelność wyświetlanego obrazu.

Instalacja i wdrożenie

W tym rozdziale przedstawiono wymagania systemowe oraz poszczególne kroki konfiguracji systemu jakie należy wykonać w celu uruchomienia stworzonego systemu. Testowanie odbywało się na systemie Ubuntu w wersji 18.04, jak również Manjaro Linux w wersji 17.1.8. Kroki konfiguracyjne zostały opisane dla systemu Ubuntu 18.04.

4.1 Wymagania systemowe:

Stworzone oprogramowanie do prawidłowego działania wymaga instalacji trzech programów: *Wireshark, traceroute* oraz *Curl.* Wszystkie znajdują się w repozytorium systemu. W celu instalacji, w terminalu należy wpisać następujące polecenie dla każdego z pakietów, który ma być zainstalowany:

$: sudo apt install curl

$: sudo apt install traceroute

$: sudo apt install wireshark

Dodatkowym elementem potrzebnym do działania stworzonego systemu, są biblioteki *boost* oraz *Qt. [dopytać się o instalacje bibliotek]*

4.2 Budowanie

W celu zainstalowania systemu należy zbudować projekt używając programu *Make.* Aby to zrobić, należy uruchomić terminal w folderze zawierającym projekt i wpisać polecenie:

$: make build

Zbudowany program można uruchomić przy pomocy dołączonego skryptu wpisując polecenie:

$: sudo ./NVC [dodać obsługę argumentów]

W tym momencie program rozpoczyna swoje działanie i przechwytywanie pakietów. Na ekranie powinna wyświetlić się konturowa mapa świata.