Sauron

Mateusz Forc,
Grzegorz Staniszewski,
Wiktor Franus,
Przemysław Kopański

29 maja 2017

Spis treści

1 Treść zadania

W pierścieniu połączone są serwer i N czujników. Serwer jest stale aktywny. Czujniki periodycznie budzą się i wykonują pomiar. Każdy czujnik próbuje odebrać komunikat z pomiarami od jednego sąsiada i wysłać drugiemu sąsiadowi. Czujniki oszczędzają energię, zatem należy minimalizować czas ich pracy i wolumin transmitowanych danych. Sieć ma tolerować pojedyncze uszkodzenie węzła lub połączenia. Należy zaprojektować i wykonać system komunikacji dla tej sieci. Przeanalizować wybór protokołu IPv4 i IPv6. Ponadto należy zaprojektować moduł do Wireshark umożliwiający wyświetlanie i analizę zdefiniowanych komunikatów.

2 Założenia funkcjonalne

1. Serwer
   (a) inicjalizuje sieć czujników
   (b) ustala tryb pomiaru wspólny dla wszystkich czujników
   (c) uruchamia pomiar
   (d) przechowuje informacje o czujnikach oraz wykonywanych pomiarach
   (e) rozpoznaje miejsca awarii sieci i wypisuje komunikaty
   (f) w przypadku awarii dowolnego czujnika przystosowuje sieć do możliwie poprawnego funkcjonowania
2. Czujnik
   (a) periodycznie budzi się, wykonuje pomiar i przesyła go do serwera
   (b) przekazuje odbierane wiadomości w stronę docelowego urządzenia
3. Aplikacja wspiera IPv4 i IPv6.
4. Konfiguracja sieci pobierana jest z pliku.

3 Założenia niefunkcjonalne

1. Serwer jest stale aktywny.
2. Sieć ma tolerować uszkodzenia węzłów lub połączeń.
3. Minimalizacja czasu pracy czujników i ilości transmitowanych danych.

4 Przypadki użycia

4.1 Włączenie systemu

Aktorzy: użytkownik
Scenariusz główny:

1. Użytkownik wprowadza do pliku konfiguracyjnego adresy czujników i tryb pomiaru
2. Użytkownik uruchamia aplikację (serwer)
3. Serwer inicjalizuje czujniki
4. Serwer wypisuje komunikat o sukcesie
5. Serwer rozpoczyna wykonywanie pomiarów przez czujniki
6. Serwer wypisuje na ekran pomiary otrzymane z czujników

Scenariusz alternatywny 1: (Jeden czujnik nie działa na etapie inicjalizacji)

1-3. Jak w scenariuszu głównym

4. Serwer wypisuje ostrzeżenie z informacją o niedziałającym czujniku

5-6. Jak w scenariuszu głównym

Scenariusz alternatywny 2: (Jeden czujnik przestaje działać po etapie inicjalizacji)

1-5. Jak w scenariuszu głównym

6. Serwer wypisuje ostrzeżenie z informacją o niedziałającym czujniku
7. Serwer wypisuje na ekran pomiary otrzymane z czujników

Scenariusz alternatywny 3: (Więcej niż jeden czujnik przestaje działać po etapie inicjalizacji)

1-5. jak w scenariuszu głównym

6. Serwer wypisuje komunikat o awarii systemu
7. Serwer usypia działające czujniki
8. Serwer kończy działanie jak w PU 4.2

4.2 Wyłączenie systemu

Scenariusz główny:

1. Serwer usypia działające czujniki
2. Serwer kończy działanie

4.3 Wyłączenie systemu przez użytkownika

Scenariusz główny:

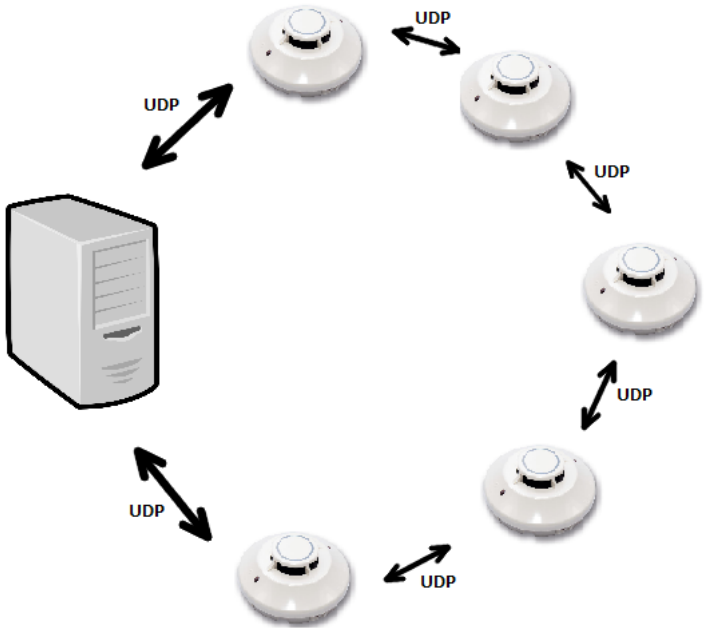
1. Użytkownik zleca serwerowi zakończenie pracy
2. Serwer kończy działanie jak w PU 4.2

5 Wybranie środowisko sprzętowo-programowe i narzędziowe

1. System operacyjny: GNU Linux (dystrybucje: ArchLinux, Ubuntu)
2. Języki programowanie: C, C++ (aplikacja), Lua (wtyczka do Wireshark)
3. Kompilatory: GNU C++ Compiler, Clang
4. Biblioteki: API gniazd BSD, Boost, YAML-CPP
5. Narzędzia pomocnicze: Git, Gerrit, Github, gdb, CMake, Wireshark, Jenkins, Docker

6 Architektura rozwiązania

Architektura rozwiązania jest hybrydą architektur pierścieniowej oraz klient-serwer. W sieci pierścieniowej połączone są ze sobą czujniki - dokonujące pomiaru, wraz z serwerem głównym - zbierającym wyniki oraz zestawiającym całą sieć do poprawnej transmisji. Dodatkowo, czujniki pełnią rolę serwera pośredniczącego; ze względu na budowę sieci muszą one odbierać i przekazywać wyniki pomiarów innych czujników oraz informacje organizacyjne sieci.



6.1 Inicjalizacja

Serwer trzyma listę adresów IP należących do pierścienia. Wyznacza nastepnie podziałkę w ich połowie - od tej pory komunikacja będzie odbywać się tylko w dwóch połowach pierścienia niezależnie (oszczędzamy wolumin danych i ułatwiamy zarządzanie nimi w normalnym trybie pracy). Do wyznaczonych połówek wysyłane są adresy określające połowy pierścienia wraz z kierunkiem przesyłania pomiarów. Serwer oczekuje na informację zwrotną z obu połówek świadczącą o braku błędów podczas inicjalizacji. Każdy czujnik po wysłaniu wiadomości oczekuje na odpowiedź, co umożliwia wykrycie awarii sieci.

Czujnik pracuje w dwóch trybach naprzemiennie:

- w trybie aktywnym (okienko czasowe), w którym dokonuje pomiaru i przekazuje w kierunku serwera pomiary dokonane przez inne wezły sieci.
- w trybie nieaktywnym, w którym nasłuchuje tylko na wybrane komunikaty

6.1.1 AWARIA 1: węzeł nie odpowiada podczas inicjalizacji

Węzeł, który nie otrzymał odpowiedzi zwrotnej od sąsiada podczas inicjalizacji, zwraca w przeciwną stronę informację o błędzie zawierającą adres IP węzła, który nie odpowiedział. Informacja z błędem trafia ostatecznie do serwera. Serwer rozpoczyna inicjalizację czujników od początku, dostosowując listy adresów czujników w obu połówkach pierścienia, w sposób umożliwiający poprawną komunikację z jednym niedziałającym czujnikiem.

6.2 Tryb pomiaru

Pomiary startują po otrzymaniu od serwera wiadomosci RUN. Węzły wysyłają wynik pomiaru w stronę serwera; każdy z nich, jeżeli otrzyma od sąsiada pomiar, przekazuje go dalej w odpowiednią stronę. Po każdej wysłanej wiadomości węzeł oczekuje na jej potwierdzenie od sąsiedniego węzła, brak potwierdzenia oznacza awarię węzła.

6.2.1 AWARIA 2: węzeł nie odpowiada podczas zbierania wyników pomiarów

Węzeł, który nie otrzymał potwierdzenia od adresata wiadomości zawierającej pomiaru, wysyła w przeciwnym kierunku tą samą wiadomość dodając do niej informację o adresie wadliwego węzła. Czujnik znajdujący się po drugiej stronie wadliwego węzła, nie otrzymawszy w dopuszczalnym czasie od poprzednika żadnej wiadomości, przesyła do kolejnego węzła swój pomiar dołączając do wiadomości adres wadliwego węzła. Serwer z dwóch stron otrzymuje informację o pomiarach i wadliwym czujniku. Jeśli adresy otrzymane z dwóch stron są różne to zaszła awaria więcej niż jednego czujnika. Serwer przystępuje wtedy do wyłączania czujników. Jeśli serwer otrzymał od obu sąsiadów ten sam adres wadliwego czujnika, oznacza to, że tylko jeden węzeł doznał awarii. W tym przypadku serwer dokonuje ponownej inicjalizacji sieci dostosowując odpowiednio listę adresów IP w obu połówkach pierścienia.

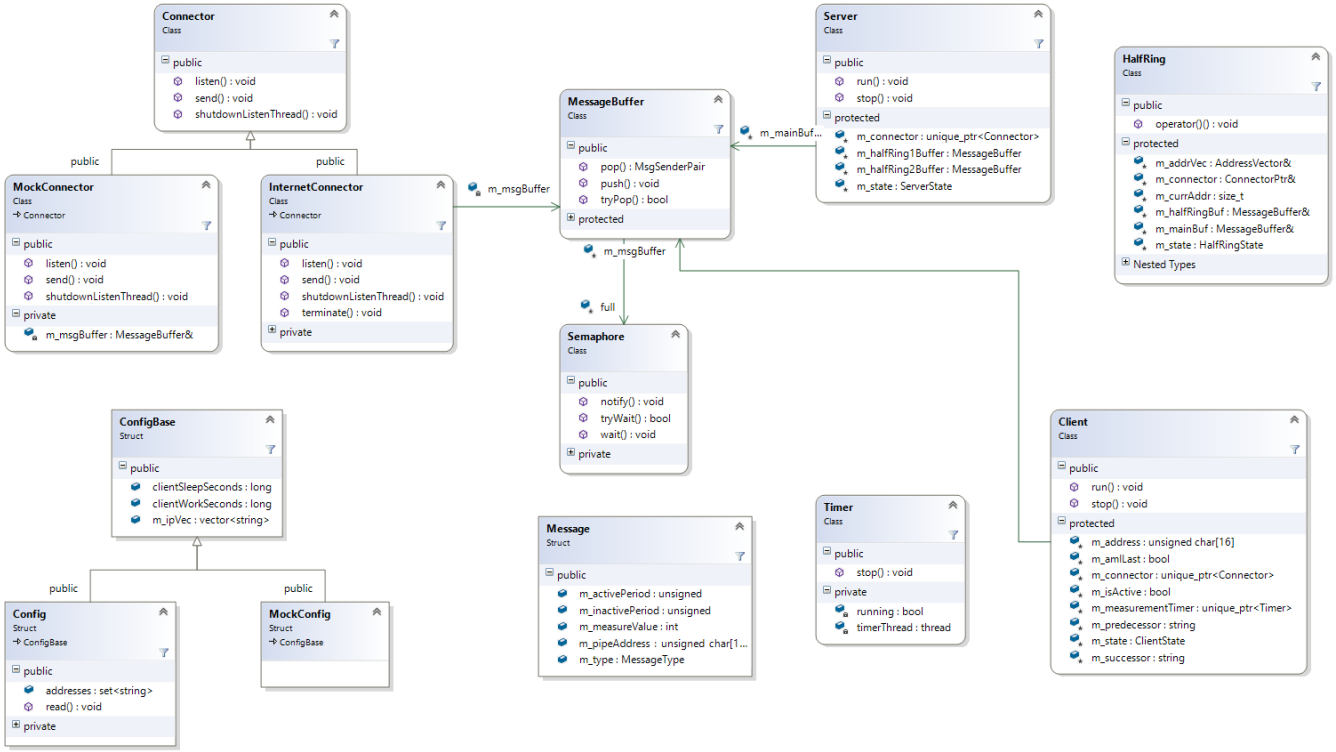
7 Sposób testowania

- Testy jednostkowe dla wszystkich modułów napisane w Boost Test Unit
- Testy integracyjne
- Możliwość uruchomienia wszystkich rodzajów testów za pomocą jednego polecenia

8 Sposób demonstracji rezultatów

Działanie systemu zostanie zaprezentowane przy pomocy narzędzia Docker, przy pomomcy którego zostanie stworzona wirtualna sieć pierścieniowa, w której dokona się poprawna innicjalizacja, a następnie rozpocznie się wykonywanie pomiarów.

9 Opis struktury implementacyjnej systemu



Rysunek 1: Diagram klas wykorzystywanych w aplikacji

10 Definicja komunikatów

Komunikacja między węzłami sieci odbywa się przy użyciu zdefiniowanej przez nas struktury Message:

```
struct Message {
    // wyliczeniowy typ wiadomości
    MessageType m_type;

    // adres IP wykorzystywany przy inicjalizacji i podczas wykonywania pomiarów
    unsigned char m_pipeAddress[16];

    // wynik pomiaru
    int m_measureValue;

    // czas pracy czujnika (w sek.)
    unsigned m_activePeriod;

    // czas "spania" czujnika (w sek.)
    unsigned m_inactivePeriod;
};
```

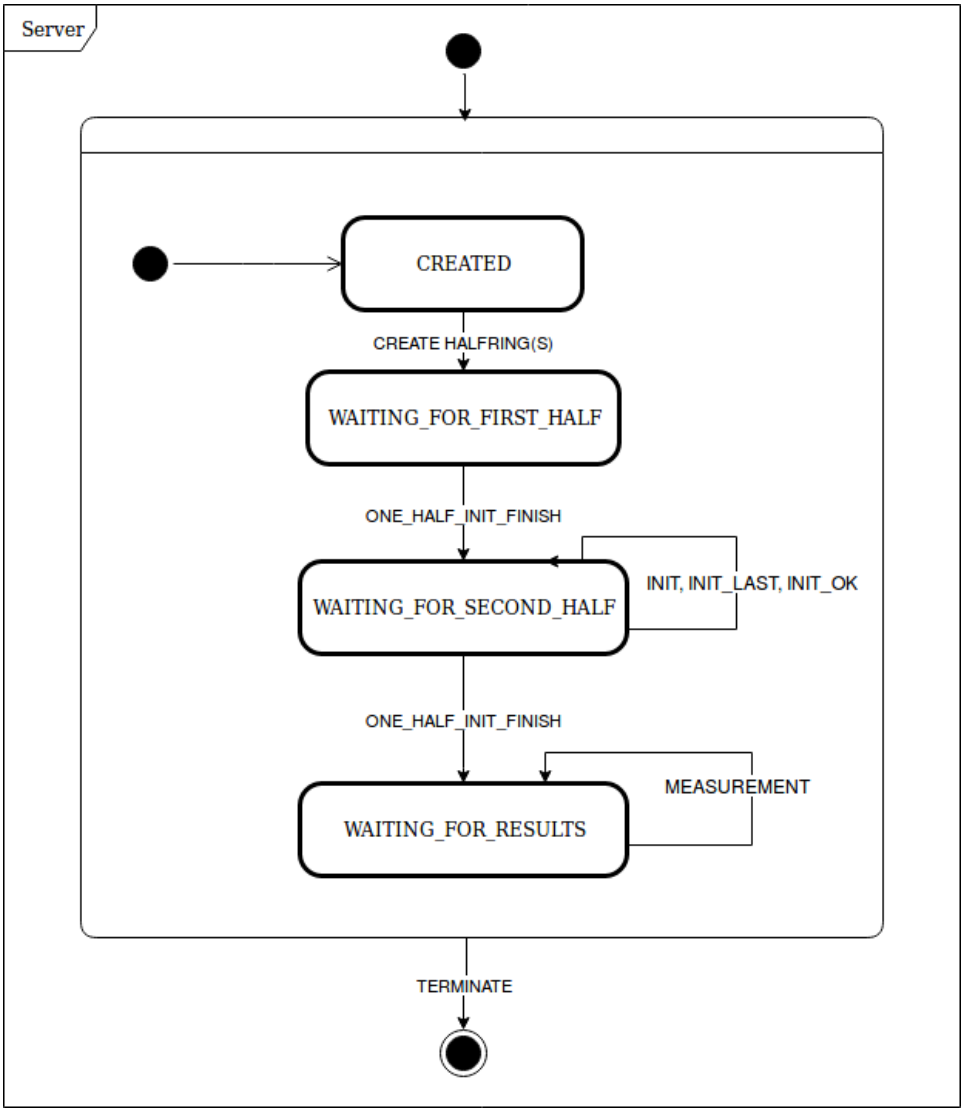
Pole m_type w strukturze Message może przyjmować jedną z wartości typu wyliczeniowego MessageType:

- **Init** - wysyłane przez serwer na etapie inicjalizacji w celu dołączenia do sieci kolejnego węzła. W polu m_pipeAddress znajduje się adres IP dodawanego węzła. Czujniki otrzymując wiadomość tego typu po raz pierwszy, dołączają się do sieci (czujnik zapamiętuje adres poprzednika) i wysyłają w stronę serwera wiadomość typu InitOk. Kolejne odebranie wiadomości tego typu skutkuje zapamiętaniem adresu następnego węzła w pierścieniu i przekazaniem mu tej wiadomości.
- **InitOk** - wysyłane przez świeżo dołączonego do sieci klienta w stronę serwera
- **InitLast** - wysyłane dwukrotnie przez serwer do ostatniego klienta w półpierścieniu. Za pierwszym razem pole m_pipeAdress zawiera adres ostatniego węzła należącego do półpierścienia. Za drugim - adres ostatniego węzła z drugiej połowy pierścienia. Niezainicjalizowany do tej pory klient (ostatni) otrzymawszy kolejno dwie takie wiadości zapamiętuje swój adres, awaryjny adres sąsiada z drugiej połowy pierścienia oraz informację o swoim krańcowym położeniu w półpierścieniu.
- **Run** - wysyłane przez serwer do pierwszych klientów w półpierścieniach. Klient przekazuje tą wiadomość następnikowi, sam rozpoczynając pierwszy pomiar.

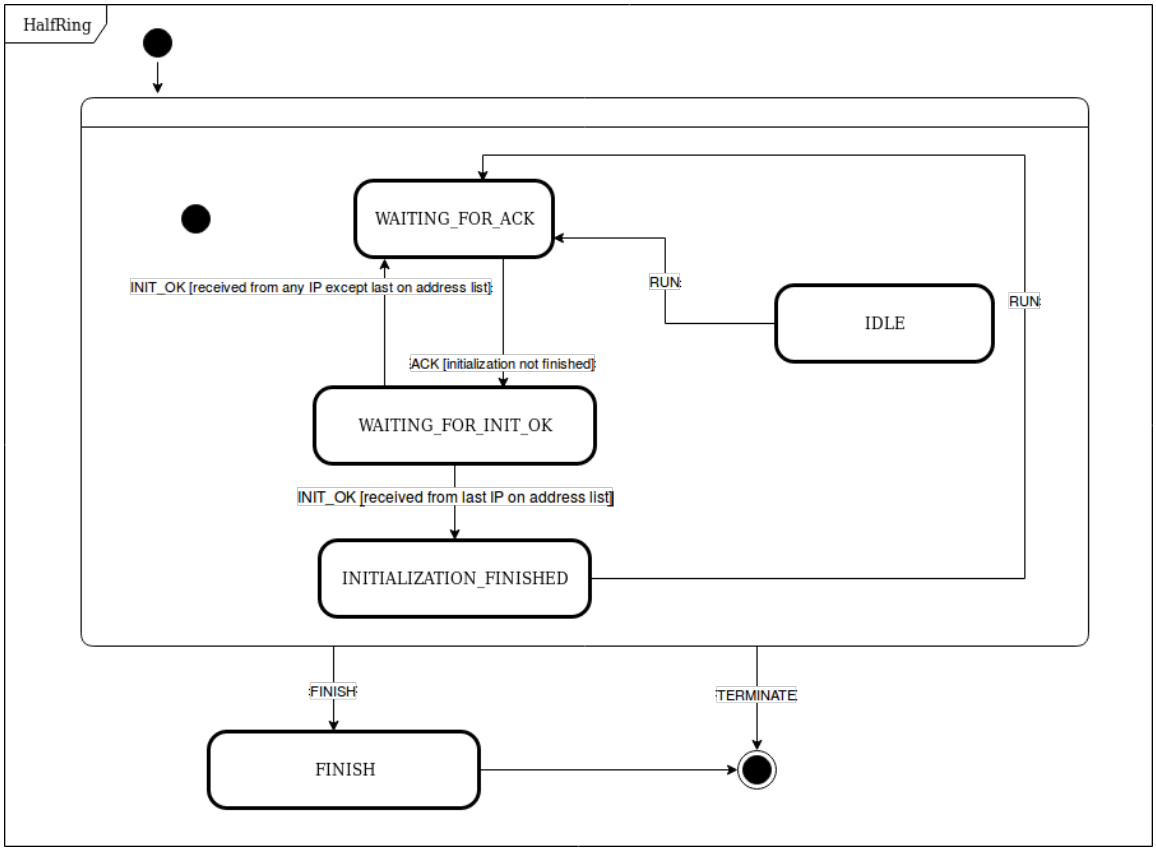
- **Measurement** - wysyłane przez klienta w stronę serwera. W polu m_pipeAddress umieszczony jest adres czujnika, który wykonał pomiar. Wynik pomiaru znajduje się w polu m_measureValue.
- **Finish** - wysyłane przez serwer do klienta podczas wyłączania systemu. Klient przekazuje wiadomość do następnika i kończy działanie. Używane również wewnątrz modułu serwera do informowania wątków obługujących półpierścienie o zakończeniu działania aplikacji.
- **Ack** - wysyłane przez węzeł sieci (serwer lub klient) do nadawcy poprzednio odebranej przez węzeł wiadomości (potwierdzenie odebrania).
- **Terminate** - pomocniczy typ używany wewnątrz modułów serwera i klienta (nie przesyłany przez sieć).
- **OneHalfInitFinish** - pomocniczy typ używany wewnątrz modułu serwera. Wiadomość tego typu umieszczana jest w buforze wątku głownego przez wątek obłsugujący półpierścień po pomyślnej inicjalizacji obsługiwanych czujników.

11 Opis zachowania podmiotów komunikacji

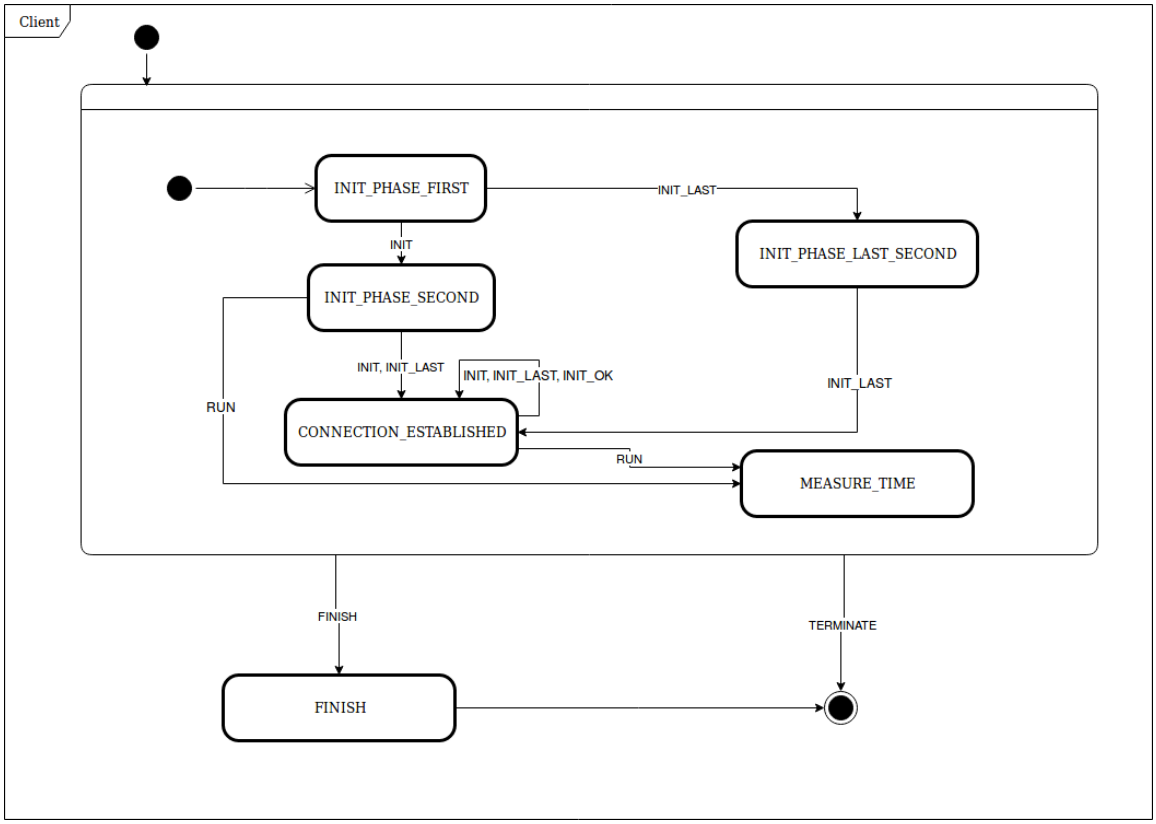
11.1 Diagramy stanów



Rysunek 2: Diagram stanów głównego wątku serwera.



Rysunek 3: Diagram stanów wątku obsługującego połowę pierścienia czujników.

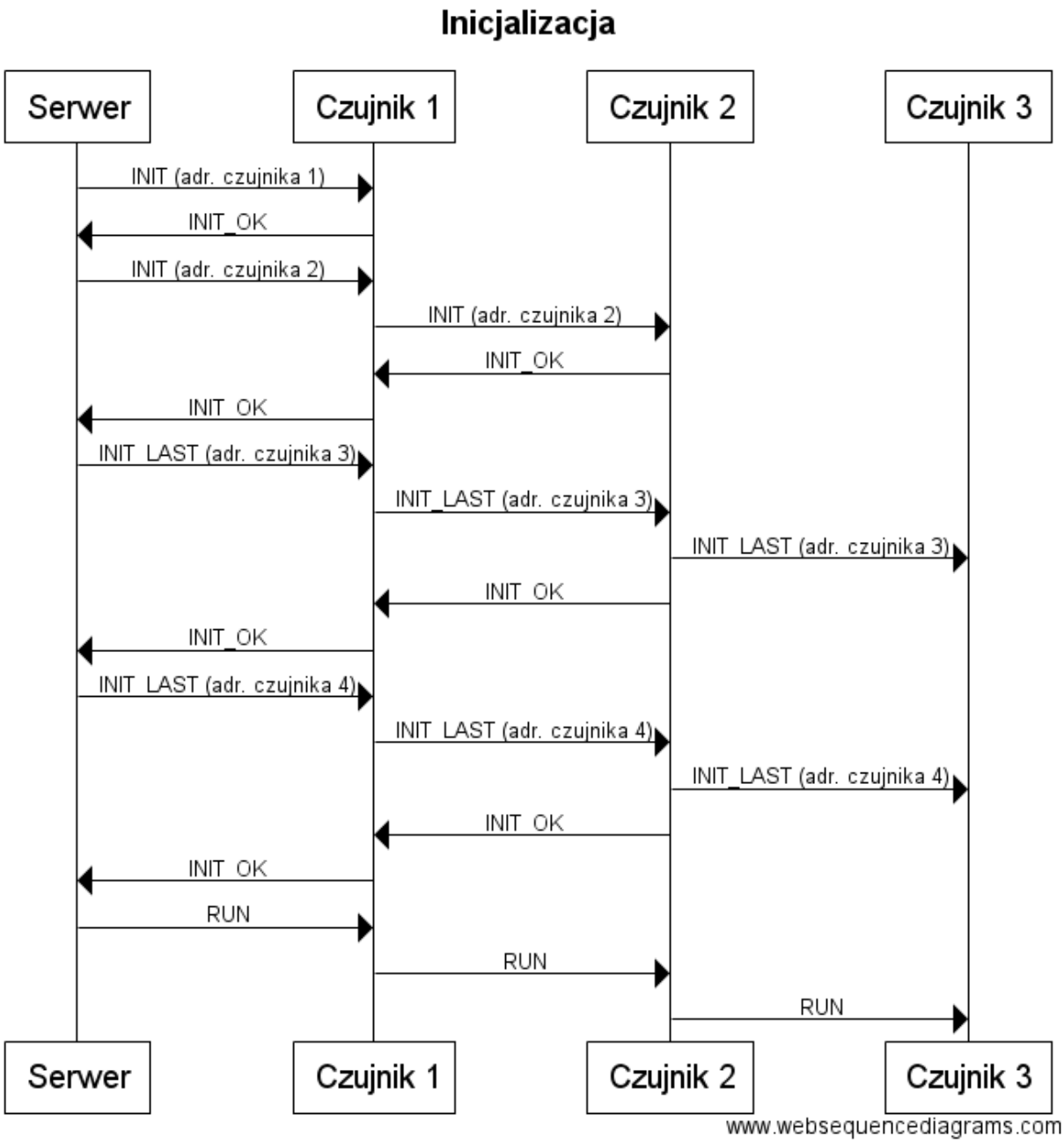


Rysunek 4: Diagram stanów klienta.

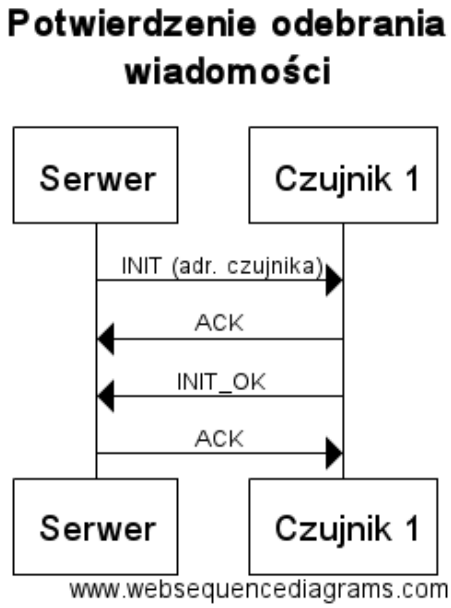
11.2 Opis zegarów klienta

Z każdym z czujników związany jest zegar, który wyzwala dokonanie pomiaru. Wykonany pomiar jest natychmiast wysyłany w stronę serwera. Przez pozostały czas trwania 'okienka czasowego' czujnik nasłuchuje na pomiary od sąsiadów i przekazuje je dalej. Po upłynięciu czasu pracy czujnik jest usypiany na okres trwania czasu nieaktywności. W tym czasie wszystkie odebrane komunikaty są ignorowane. Kolejne wyzwolenie zegara rozpoczyna nowy pomiar. Czas między kolejnymi pomiarami równy jest zatem sumie czasu pracy i czasu 'spania' czujników.

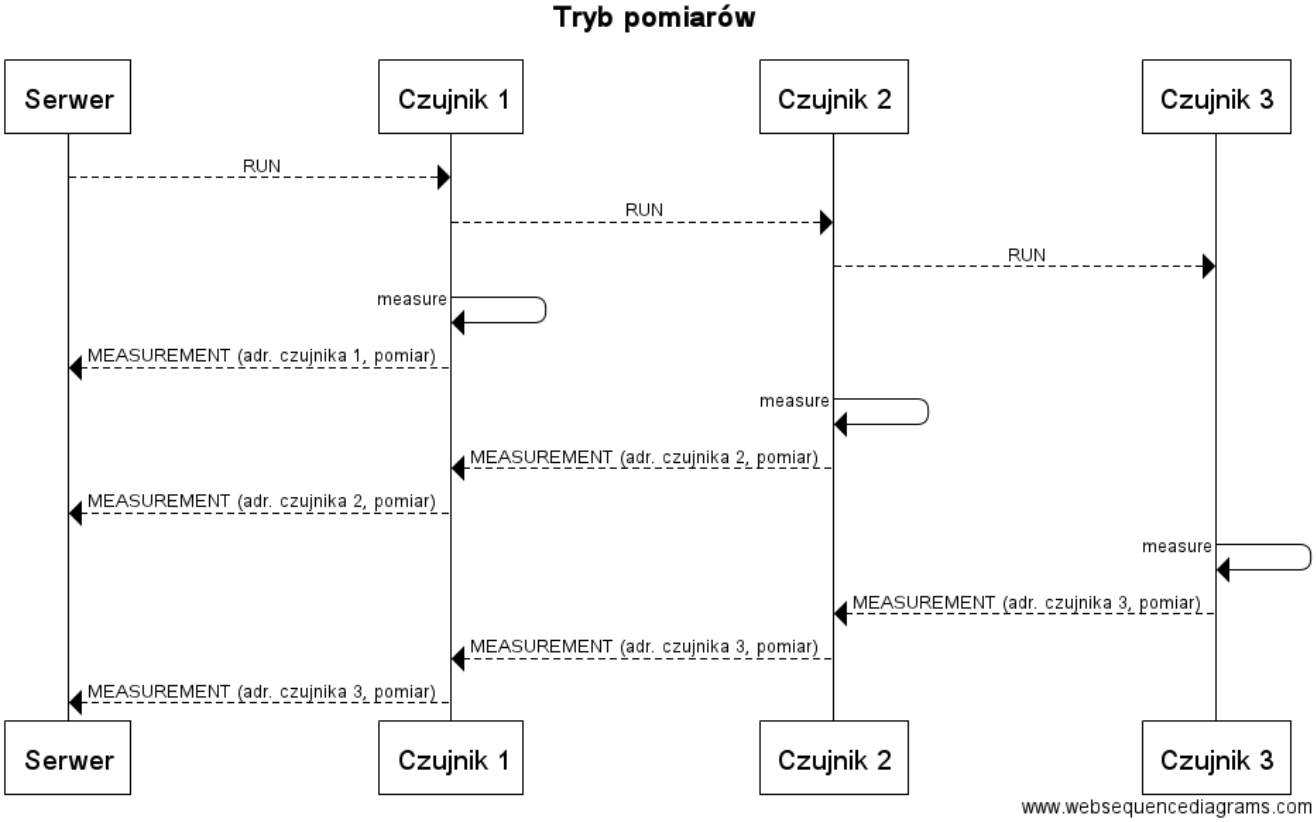
11.3 Scenariusze komunikacji



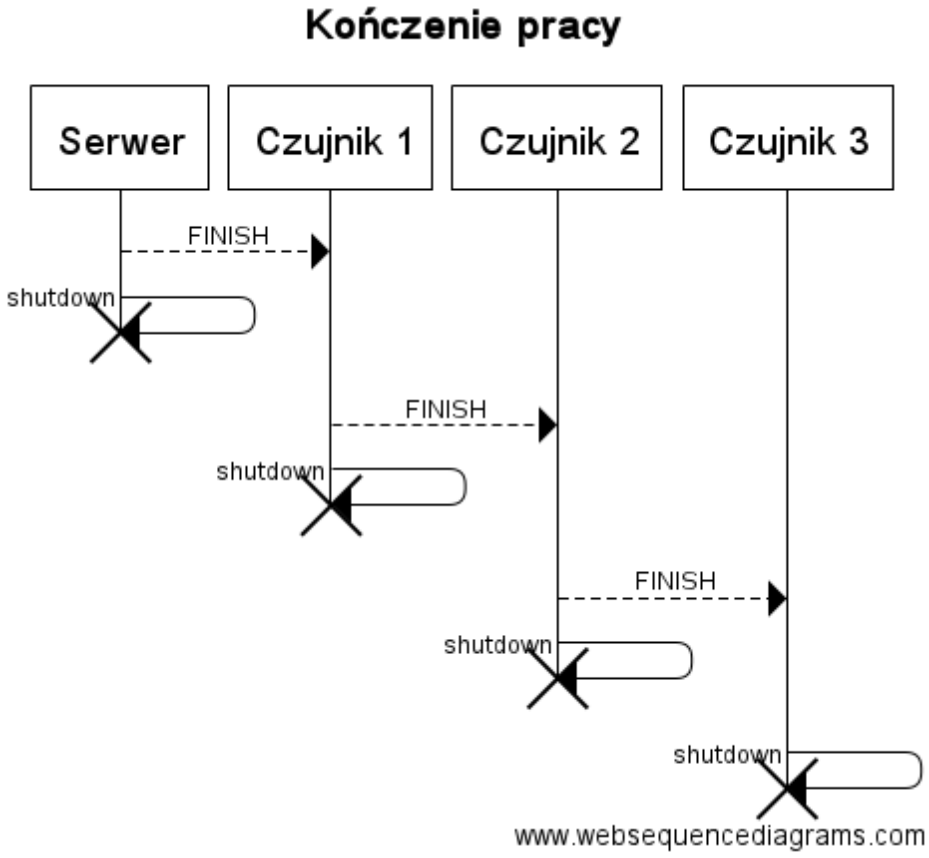
Rysunek 5: Sekwencja komunikatów podczas inicjalizacji sieci. Czujniki 1,2 i 3 należą do tego samego półpierścienia. Czujnik 4 (nie umieszczony na rysunku) jest ostatnim węzłem z drugiej połowy pierścienia.



Rysunek 6: Przykładowa sekwencja przedstawiająca potwierdzenie otrzymania wiadomości przez węzeł w sieci.



Rysunek 7: Sekwencja komunikatów podczas rozpoczynania etapu pomiarów.



Rysunek 8: Sekwencja komunikatów podczas kończenia pracy systemu.

12 Wnioski z wykonanego testowania

Aplikacja z racji na wielowątkowość, wykorzystywanie protokołu sieciowego oraz ścisłe powiązanie serwera/klienta z konkretnym portem jest problematyczna w testowaniu. Integracja serwera oraz kilku instancji klienta bez wykorzystania izolowanych wirtualnych środowisk jest w zasadzie niemożliwa. Testowanie jest procesem znacznie bardziej kosztownym czasowo niż implementacja, jednak jest to proces, w który warto inwestować czas i pieniądze gdyż zwiększa on jakość projektu oraz daje poczucie deweloperom, iż wprowadzane przez nich zmiany nie popsują działającego już kodu.

13 Podział prac w zespole

Prace nad projektem wykorzystują jedną z metod programowania zwinnego, tj. pair programming - każde zadanie przydzielane jest parze. Pary oceniają nawzajem stworzony przez siebie kod przy uzyciu narzędzia do rewizji kodu Gerrit. Istnieją zadania, które wykonuje cały zespół lub sam lider.
Przydział zadań:

1. Sprint 1 - dokumentacja wstępna i repozytorium
    - Stworzenie dokumentacji wstępnej - cały zespół
    - Utworzenie repozytorium na Github - Mateusz Forc (lider)
    - Integracja repozytorium z Gerrit - Mateusz Forc (lider)
    - Założenie kont dla programistów oraz prowadzącego - Mateusz Forc (lider)
    - Stworzenie instrukcji korzystania z Gerrit oraz ustalenie stylu kodowania - Mateusz Forc (lider)
2. Sprint 2 - implementacja podstawowych funkcjonalności
    - Projekt komunikatów - cały zespół
    - Wtyczka do Wireshark - Przemysław Kopański
    - Wrapper API gniazd BSD - Mateusz Forc (lider), Wiktor Franus
    - Testy jednostkowe wrappera w zakresie podstawowym - Mateusz Forc (lider), Wiktor Franus
    - Projekt i implementacja modułu serwera wraz z podstawowymi funkcjonalnościami - Mateusz Forc (lider), Wiktor Franus
    - Projekt i implementacja modułu klienta (czujnika) wraz z podstawowymi funkcjonalnościami - Grzegorz Staniszewski, Przemysław Kopański
    - Testy jednostkowe modułów serwera w zakresie podstawowym - Mateusz Forc (lider), Wiktor Franus
    - Testy jednostkowe modułów klienta w zakresie podstawowym - Grzegorz Staniszewski, Przemysław Kopański
    - Podstawowe testy integracyjne aplikacji - Grzegorz Staniszewski, Przemysław Kopański
    - Zaprojektowanie i implementacja klasy bufora wiadomości - Przemysław Kopański
    - Budowanie aplikacji z użyciem narzędzia CMake - cały zespół
    - Poprawa stylu kodowania - Grzegorz Staniszewski

- Zmiana struktury katalogów w projekcie - Mateusz Forc (lider), Wiktor Franus
- Utworzenie i skonfigurowanie zadań w narzędziu Jenkins (ciągła integracja) - cały zespół

3. Sprint 3 - kompletna implementacja wraz ze wszystkimi testami

    - Uzupełnienie testów jednostkowych wrappera - Mateusz Forc (lider), Wiktor Franus
    - Bezpieczne zamykanie aplikacji wraz z jej wszystkimi wątkami i zwalnianiem zasobów - cały zespół
    - Projekt i implementacja zegarów synchronizujących pracę czujników - Przemysław Kopański
    - Utworzenie i konfiguracja środowiska do testów akceptacyjnych - Przemysław Kopański
    - Implementacja pozostałych funkcjonalności modułu serwera - Mateusz Forc (lider), Wiktor Franus
    - Wczytywanie konfiguracji sieci z pliku - Grzegorz Staniszewski
    - Implementacja pozostałych funkcjonalności modułu klienta - Grzegorz Staniszewski, Przemysław Kopański
    - Pozostałe testy modułu serwera - Mateusz Forc (lider), Wiktor Franus
    - Pozostałe testy modułu klienta - Grzegorz Staniszewski, Przemysław Kopański
    - Uodpornienie programu na niepożądane parametry od użytkownika - Grzegorz Staniszewski
    - Aktualizacja zadań w sprincie 2 i 3 w ramach dokumentacji - Mateusz Forc (lider), Wiktor Franus
    - Aktualizacja dokumentacji przed oddaniem projektu - cały zespół
    - Poprawa nagłówków w plikach z kodem źródłowym - Mateusz Forc (lider), Wiktor Franus
    - Aktualizacja zadań na GitHubie w oparciu o narzędzie Gerrit oraz dokumentację - Mateusz Forc (lider)

14 Harmonogram prac

Sprint 2 - od 19.04 do 15.05
Sprint 3 - od 15.05 do 31.05

15 Podsumowanie

15.1 Opis wyniesionych doświadczeń z realizacji projektu

- praca z interfejsem gniazd BSD była uciążliwa, a popełnione błędy trudne do wykrycia podczas pisania. W przyszłych projektach wymagających użycia komunikacji sieciowej zdecydujemy się z pewnością na użycie gotowych, przetestowanych i bardziej abstrakcyjnych bibliotek
- poszerzenie znajomości języka C++
- praca w parach przyniosła wiele pozytywnych efektów, m.in. przyjmeniejszą pracę nad kodem, szybsze rozwiązywanie problemów, dzielenie się wiedzą z kolegami i pogłebianie stosunków międzyludzkich

15.2 Statystyki rozmiarów stworzonych plików

Liczba linii w plikach źródłowych: ok. 1800
Liczba linni w plikach z testami: ok. 1000
Wtyczka do Wireshark: ok. 60

15.3 Czas poświęcony na realizację projektu w godzinach

Realizacja projektu, w skład której oprócz projektowania i implementacji wchodziły także spotkania konsultacyjne i etapowe pochłonęła około 280 osobogodzin (średnio 70h na osobę).

16 Adres projektu

Repozytorium kodu dostępne jest w serwisie Github pod adresem:

https://github.com/formateu/sauron

Dostęp do repozytorium jest ograniczony do kont posiadających uprawnienia do jego odczytu.