Zespół: Z46
Opiekun: Jacek Wytrębowicz
Data: 16/01/2023 r.

Autorzy: Marcin Grabysz (lider),
Jan Jędrzejewski, Patrycja Wysocka,
Szymon Wysocki

PSI - System agregacji dokumentów

Dokumentacja końcowa

Treść zadania

Zaprojektuj i zaimplementuj system bramy komunikacyjnej dla podsieci do 1024 urządzeń sensorycznych. Brama odbiera od urządzeń krótkie binarne paczki danych, agreguje je i rozsyła do predefiniowanego zbioru serwerów (do 32). Urządzenia wysyłają dane w określonych interwałach czasowych. Brama opatruje zagregowane paczki stemplem czasowym i podpisem cyfrowym. Komunikacja odbywa się po UDP. Co określony czas brama wysyła do urządzeń komunikat synchronizujący czas i wariację opóźnienia (jitter) dla wysyłanych paczek danych.

Przyjęte założenia

Tworzymy bramę do agregacji danych z urządzeń sensorycznych. W określonych interwałach czasowych urządzenia wysyłają paczki danych. Brama odbiera dane od urządzeń, łączy je do jednego formatu i przesyła do każdego z predefiniowanych serwerów. Dane od każdego z aktualnie pracujących urządzeń sensorycznych są wysyłane do każdego z serwerów.

1. Parametrami w systemie są
    a. interwał czasowy komunikacji T_K
    b. interwał czasowy synchronizacji T_S
    c. zbiór adresów serwerów nadrzędnych.
2. Urządzenia komunikacyjne mogą być rejestrowane i wyrejestrowywane w dowolnym momencie działania bramy.
3. Brama w zbiorze zapisuje wszystkie zarejestrowane urządzenia sensoryczne.
4. Dane wysyłane są w formacie JSON.
5. Co określony czas brama wysyła do urządzeń komunikat synchronizujący czas i wariację opóźnienia.

Przypadki użycia

Zarejestrowanie urządzenia komunikacyjnego

Urządzenia komunikacyjne rejestrują się w bramie, wysyłając pakiet z prośbą o zarejestrowanie. Brama przekazuje wiadomość modułowi “rejestracja”. Jeżeli klient o podanym id lub adresie ip już istnieje, moduł nie wykonuje rejestracji i loguje odpowiedni komunikat w dzienniku zdarzeń. W przeciwnym razie id klienta zostaje dodane do listy zarejestrowanych urządzeń.

Wyrejestrowanie urządzenia komunikacyjnego

Urządzenia komunikacyjne wyrejestrowują się w bramie, wysyłając pakiet z prośbą o wyrejestrowanie. Brama przekazuje wiadomość modułowi “rejestracja”. Jeżeli klient o danym id jest zarejestrowany, zostaje usunięty z listy zarejestrowanych urządzeń. W przeciwnym wypadku zdarzenie zostaje zalogowane w dzienniku zdarzeń.

Transmisja danych użytkowych z klientów do bramy

Urządzenia komunikacyjne wysyłają paczki danych, w których “ładunkiem” jest pewna liczba zmiennoprzecinkowa. Pakiety od niezarejestrowanych urządzeń są ignorowane. Odebranie wiadomości od niezarejestrowanego urządzenia jest logowana w dzienniku zdarzeń, podobnie jak odebranie wiadomości o nieprawidłowym formacie. Zawartość odebranych pakietów jest agregowana.

Transmisja danych z bramy do serwerów

Co pewien interwał czasu zdefiniowany w momencie tworzenia bramy komunikacyjnej, brama wysyła zagregowane dane każdemu z serwerów. Przed wysłaniem danych, wiadomość jest podpisywana. Podpis cyfrowy zajmuje pierwsze 256 bajtów wiadomości do serwerów. Za wysyłanie zagregowanych danych odpowiedzialny jest moduł “transmisja”.

Rozesłanie wariacji opóźnienia zarejestrowanym klientom

Co pewien interwał czasu zdefiniowany w momencie tworzenia bramy komunikacyjnej, brama wysyła każdemu z klientów wariację opóźnienia (jitter). Klienci dostosowują opóźnienie wysyłania wiadomości. Za wysyłanie wariacji opóźnienia odpowiedzialny jest moduł “synchronizacja”.

Analiza błędnych sytuacji

Rejestracja urządzenia, które jest już zarejestrowane

Brama nie pozwala na rejestrację urządzenia.

Wyrejestrowanie urządzenia, które nie jest zarejestrowane

Brama nie pozwala na wyrejestrowanie urządzenia.

Nieodebranie danych z zarejestrowanego urządzenia w określonym interwale

W końcowej, zagregowanej paczce wysłanej do serwerów brama wypełnia pole payload wartościami Null. Urządzenia od których nie odnotowano danych są rejestrowane.

Odebranie danych od niezarejestrowanego urządzenia

Są traktowane jako błędny pakiet i dane nie są zapisywane w bramie.

Odebranie niepoprawnie sformatowanej wiadomości

Wiadomości w formacie innym niż w dokumentacji są odrzucane przez bramę.

Wszystkie wystąpienia błędnych sytuacji są wypisywane na konsolę. Do pliku gateway.log są zapisywane logi wraz z czasem wystąpienia, poziomem ważności i wiadomością o powodzie wystąpienia błędu.

Środowisko sprzętowo-programowe

Program będzie napisany w Pythonie przy pomocy następujących bibliotek:

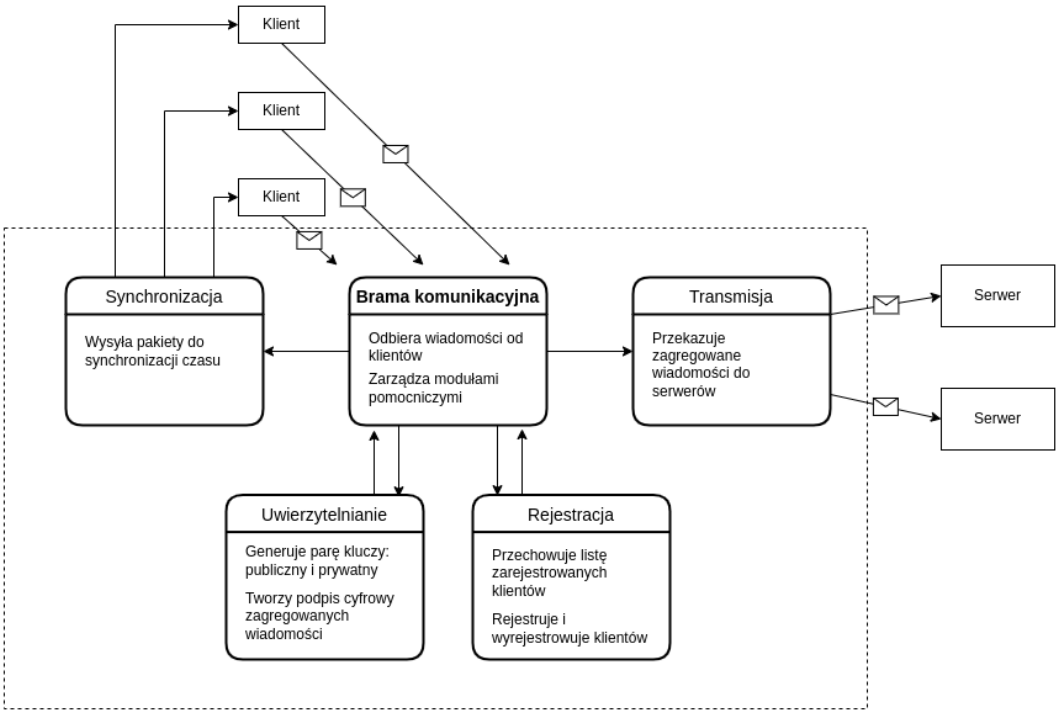
- socket - do zaimplementowania komunikacji sieciowej
- cryptography - do wygenerowania podpisu cyfrowego dla przesyłanego pakietu danych
- unittest - do tworzenia i uruchamiania testów jednostkowych

Program będzie działał na systemie operacyjnym Linux. W początkowych fazach projektu moduły będą rozwijane i testowane na komputerach lokalnych z wykorzystaniem adresów loopback, w końcowych - na maszynie wirtualnej *Bigubu* z wykorzystaniem konteneryzacji.

Architektura i opisy zachowania poszczególnych podmiotów komunikacji

W projekcie wyszczególniamy następujące moduły logiczne:

- nadawcy komunikatów (klienci)
- Brama komunikacyjna - zbiór modułów odpowiadających za odbieranie, agregowanie i wysyłanie danych do serwerów:
    - gateway (brama komunikacyjna)
        - odbiera wiadomości od klientów
        - zarządza modułami pomocniczymi
    - transmission (transmisja)
        - przechowuje zagregowane wiadomości
        - przekazuje zagregowane wiadomości do serwerów
    - sync (synchronizacja)
        - wysyła do klientów pakiety służące synchronizacji czasu
    - authentication (uwierzytelnianie)
        - generuje parę kluczy: publiczny i prywatny
        - tworzy podpis cyfrowy pakietów zagregowanych wiadomości
    - registration (rejestracja)
        - przechowuje listę zarejestrowanych klientów
        - rejestruje i wyrejestrowuje klientów
- serwery odbierające zagregowane dane
- Obsługa dziennika zdarzeń - wszelkie nietypowe zdarzenia będa odnotowywane i zapisywane w pliku .json
- test - moduł zawierający testy jednostkowe



Definicja komunikatów:

Prośba klienta o rejestrację (klient -> brama)

```
{
    "action": "register",
    "device_id": int,
    "timestamp": float
}
```

Prośba klienta o wyrejestrowanie (klient -> brama)

```
{
     "action": "unregister",
     "device_id": int
}
```

Transmisja danych użytkowych przez klienta (klient -> brama)

```
{
     "action": "transmit",
     "device_id": int,
     "timestamp": float,
     "payload": float
}
```

Przesłanie wariacji opóźnienia (brama -> klient)

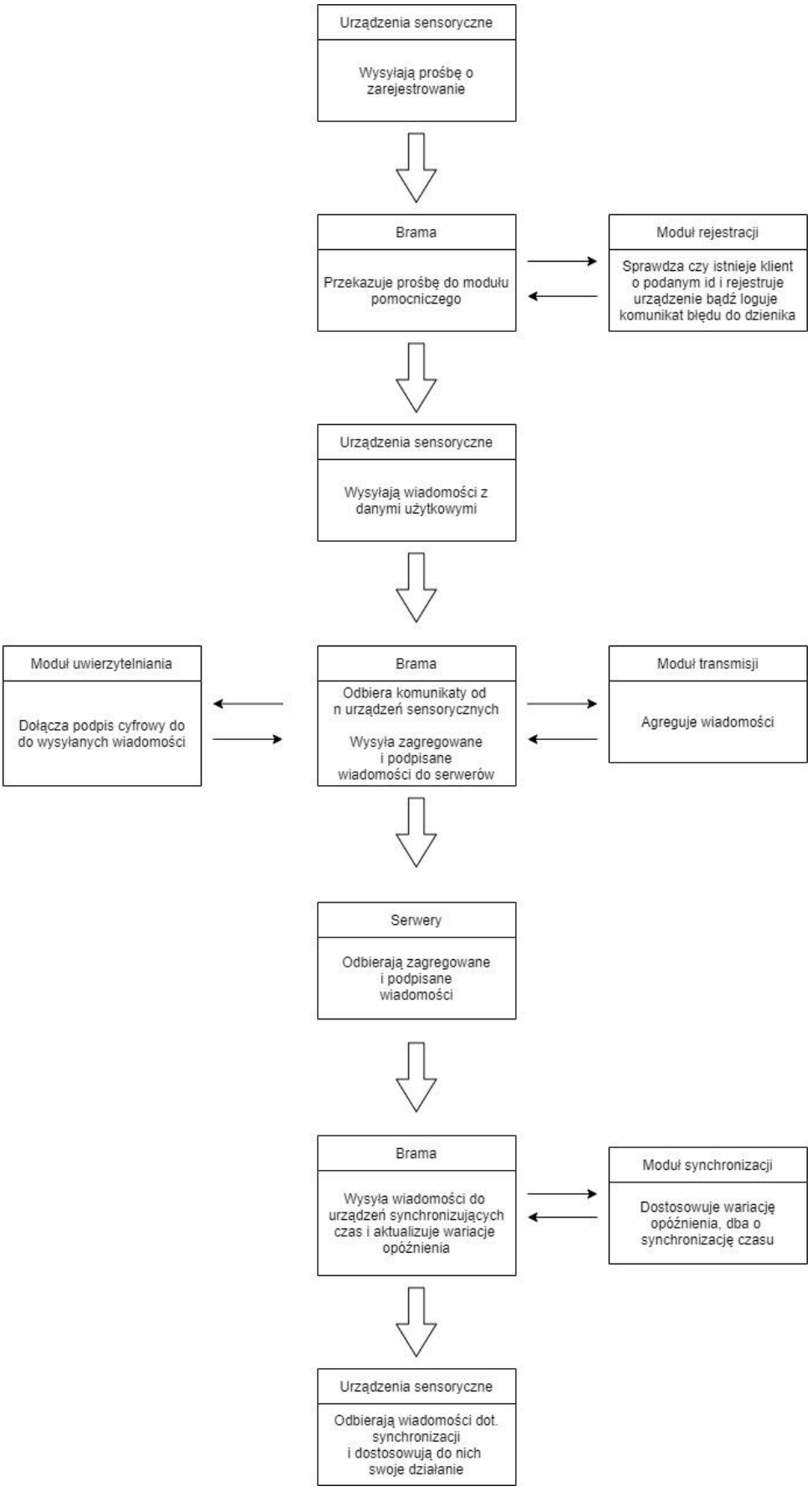
```
{
     "reference_time": float,
     "jitter": float
}
```

Przesłanie zagregowanych danych do serwerów (brama -> serwer)

Bajty 0 - 255: podpis cyfrowy
Pozostała część wiadomości: dane w formacie JSON:

```
{
     “payload”: {
          “devices”: {
               “id_1”: List[float],
               “id_2”: List[float], ...
          },
          "timestamp": float,
}}
```

Przykładowe scenariusze (diagramy komunikatów)



Sposób testowania

W celu łatwiejszego testowania, dodano możliwość uruchomienia kilku klientów w oddzielnych wątkach. Podobna funkcja została wprowadzona w serwerze.
Ręcznie przetestowaliśmy zachowanie uruchomionych programów oraz działanie bramy komunikacyjnej poprzez sprawdzenie poprawności komunikatu wysłanego do serwerów.

Podstawowe funkcje bramy są testowane za pomocą testów jednostkowych.

Do testów akceptacyjnych można użyć parametrów przekazywanych do poszczególnych komponentów, opisanych w następnej sekcji.

Parametry programów

Brama

`--port <PORT>`	Ustawia port, na którym serwer nasłuchuje na wiadomości od klientów.
`--server <IP:PORT>`	Dodaje serwer, do którego mają być wysyłane zagregowane dane. Argument może być użyty wielokrotnie.
`--interval <INTERVAL>`	Interwał komunikacji T_K, w sekundach
`--sync_interval <INTERVAL>`	Interwał synchronizacji T_S, w sekundach
`--jitter <JITTER>`	Stała wartość wariacji opóźnienia, jako część T_K. Jeśli wartość nie zostanie podana, zostanie ona wyznaczona automatycznie na podstawie liczby serwerów
`--private_key <PATH>`	Ścieżka do klucza prywatnego (jeśli nie istnieje, zostanie wygenerowany)
`--public_key <PATH>`	Ścieżka, pod którą zostanie zapisany wygenerowany klucz publiczny
`--key_password <PASSWORD>`	Hasło do klucza prywatnego
`--verbose`	Wypisywanie całego podpisu cyfrowego przy wysyłaniu paczki do serwerów

Klient

`--address <SERVER_IP>`	Adres IP pod którym znajduje się brama
`--port <SERVER_PORT>`	Port na którym znajduje się brama
`--devices <NUM_OF_DEVICES>`	Liczba klientów (wątków), którzy mają zostać uruchomieni
`--messages <NUM_OF_MESSAGES>`	Liczba wiadomości, które mają zostać wysłane przez każdego klienta (z pominięciem prośby o zarejestrowanie i wyrejestrowanie)
`--interval <INTERVAL>`	Przerwa pomiędzy kolejnymi wiadomościami wysyłanymi przez jednego klienta w sekundach
`--random_start`	dodaje pewne losowe opóźnienie przed pierwszą transmisją
`--initial_jitter <JITTER>`	początkowa wartość wariacji opóźnienia
`--id <ID>`	numer ID pierwszego klienta (kolejne będą kolejnymi liczbami całkowitymi)

Serwer

`--server_port`	Port, na którym serwer będzie nasłuchiwał na wiadomości od bramy
`--public_key`	Ścieżka do klucza publicznego bramy

Wnioski z wykonanego testowania

Uruchomienie bramy, serwerów i klientów w osobnych terminalach obrazuje przepływ danych i działanie systemu. Klienci rejestrują się do bramy przesyłają wiadomości, w bramie można obejrzeć otrzymane pakiety i regularne wysyłanie paczek do serwerów. W terminalu serwerów wyświetlane są zagregowane dane od urządzeń.

Synchronizacja czasu działa poprawnie. Po ustawieniu w klientach losowego opóźnienia (--random_start), najpierw klienci wysyłają pakiety w różnych odstępach. Po chwili brama wysyła wiadomość synchronizującą czas i jitter, a klienci zaczynają wysyłać pakiety w tym samym czasie.

W podobny sposób została przetestowana synchronizacja wariacji opóźnienia. W klientach użyto opcji --initial_jitter 0. Najpierw, klienci wysyłają pakiety w jednym momencie. Natomiast po odebraniu pakietu synchronizującego z jitter=0.1, zaczynają wysyłać pakiety w 2-sekundowych oknach czasu.

Podsumowanie

Opis wyniesionych doświadczeń z realizacji projektu

Przy realizacji projektu wykorzystaliśmy wiedzę i schematy przesyłania danych z zadań laboratoryjnych, zwłaszcza z zadania pierwszego, ponieważ w projekcie wszystkie dane przesyłamy protokołem UDP. Dodatkowo w projekcie poszerzyliśmy zakres działania o wielowątkowość, ponieważ klienci, serwery oraz brama pracują na wielu wątkach. Do tego dodaliśmy autoryzację poprzez podpisy cyfrowe oraz rozbudowaliśmy system synchronizacji i wariancji opóźnienia.

Każde z tych zadań stawiało przed nami nowe wyzwania, z którymi wcześniej na innych przedmiotach nigdy nie mieliśmy do czynienia. W obecnych czasach aplikacje wielowątkowe, komunikacja międzyprocesowa i komunikacja sieciowa jest nierozłączną częścią wytwarzania oprogramowania. W ramach projektu mogliśmy się nauczyć jak protokół UDP i przesyłanie danych jest realizowane niskopoziomowo (w porównaniu do języków takich jak java) od strony praktycznej.

Statystyki określające rozmiar stworzonych plików

Statystyki linii kodu dla bramy z podziałem na pliki

gateway.py	151
authentication.py	72
transmission.py	117
registration.py	68
sync.py	48
suma	456

client.py	190	server.py	123
messeges.py	51		
suma	241	test.py	50
łącznie:	870		

Oszacowanie czasu poświęconego na projekt w godzinach

Do śledzenia czasu poświęconego na projekt prowadziliśmy arkusz i zapisywaliśmy godziny. Oszacowaliśmy naszą łączną pracę na **80 godzin**.

Podział prac w zespole

Etap początkowy:

- klienci - Marcin Grabysz
- brama - rejestracja - Szymon Wysocki
- brama - dane
    - odbieranie paczek w interwałach, agregacje - Jan Jędrzejewski
    - wysyłanie do serwerów - Jan Jędrzejewski
- serwery - Patrycja Wysocka

Etap końcowy:

- Podpis cyfrowy - Marcin Grabysz
- Testy jednostkowe - Marcin Grabysz
- Synchronizacja czasu - Szymon Wysocki
- Interwał czasowy dla bramy inny niż dla urządzeń - zbieranie danych sensorycznych jako listę - Jan Jędrzejewski
- Logger do rejestracji błędów - Jan Jędrzejewski
- Jitter - Patrycja Wysocka
- Weryfikacja podpisu cyfrowego - Szymon Wysocki
- Ujednolicenie kodu i końcowy refaktoring - Patrycja Wysocka