Programowanie Sieciowe

System konwersji TCP/UDP - “SensNet”

Magdalena Dudek (lider), Kinga Świderek, Marcin Kowalczyk, Jakub Kowalczyk
20.12.2023 r.

Treść zadania

Zaprojektuj i zaimplementuj system bramy komunikacyjnej dla podsieci do 1024 urządzeń sensorycznych. Urządzenia obsługują jedynie protokół UDP. Użytkownicy łączą się z urządzeniami poprzez bramę używając protokołu TCP. Zbiór aktywnych urządzeń i użytkowników może się zmieniać. Zaproponuj sposób ich autoryzacji. Brama komunikacyjna powinna obsługiwać użytkowników o adresach IPv4 i IPv6.

Założenia funkcjonalne i niefunkcjonalne

Założenia funkcjonalne

1. System będzie obsługiwał do 1024 urządzeń sensorycznych.
2. Urządzenia sensoryczne będą wysyłać dane tylko za pomocą protokołu UDP.
3. Użytkownicy będą łączyć się z urządzeniami za pośrednictwem bramy, używając protokołu TCP. Górny limit liczby obsługiwanych użytkowników jest równy 256.
4. Użytkownik otrzymuje pomiary z jednego wybranego urządzenia.
5. System będzie obsługiwał użytkowników o adresach IPv4 i IPv6.
6. Urządzenia sensoryczne będą adresowane za pomocą protokołu IPv4.
7. System będzie obsługiwał dynamiczne zmiany w zbiorze aktywnych urządzeń i użytkowników.
8. System będzie zawierał mechanizm uwierzytelniania użytkowników i urządzeń, na podstawie porównywania adresów IP z listą zaufanych urządzeń i użytkowników.
9. Częstość wysyłania danych pomiarowych będzie zdefiniowana w kodzie urządzenia sensorycznego.

Założenia niefunkcjonalne

1. System będzie skalowalny, aby obsługiwać do 1024 urządzeń sensorycznych.
2. System będzie odporny na błędy, z mechanizmami do wykrywania i obsługi sytuacji błędnych.
3. System będzie zapewniał niskie opóźnienia w komunikacji TCP/UDP.

Podstawowe przypadki użycia

1. **Konfiguracja systemu:** tworzenie i konfigurowanie serwera, definiowanie uprawnień użytkowników i urządzeń.

2. **Użytkownik pobiera pomiar:** użytkownik nawiązuje połączenie TCP z bramą, która następnie odsyła pomiar z zarządcy pamięci.
3. **Urządzenie wysyła dane pomiarowe do bramy:** urządzenie wysyła dane do bramy za pomocą UDP, która następnie przekazuje je do zarządcy pamięci razem z identyfikatorem urządzenia oraz dokładnym czasem zebrania pomiarów.
4. **Autoryzacja użytkownika:** użytkownik musi być autoryzowany przez system przed nawiązaniem połączenia TCP.
5. **Dodawanie i usuwanie autoryzowanych użytkowników:** lista użytkowników, którzy mają dostęp do danych pomiarowych będzie się dynamicznie zmieniać.
6. **Autoryzacja urządzenia:** urządzenie musi zostać autoryzowane przez bramę, jeżeli komunikat pochodzi z nieznanego urządzenia zostaje zignorowany. W przeciwnym przypadku dane przekazywane zostają do zarządcy pamięci.
7. **Dodawanie i usuwanie autoryzowanych urządzeń:** lista urządzeń, które mogą komunikować się z bramą będzie się dynamicznie zmieniać.
8. **Przechowywanie danych:** zarządca pamięci przechowuje jeden ostatni pomiar dla każdego urządzenia. Pomiary na podstawie id urządzenia zostają każdorazowo nadpisywane.

Wybrane środowisko sprzętowo-programowe i narzędziowe

System będzie zaimplementowany w języku Python na systemie operacyjnym Linux. Wykorzystamy bibliotekę *socket* do obsługi połączeń TCP i UDP. Do debugowania i testowania użyjemy standardowych narzędzi dostępnych w środowisku Python, takich jak *pdb* (Python Debugger) i *unittest*. Wszelkie testy akceptacyjne będą wykonane w środowisku Docker na serwerze *bigubu*, lecz produkcyjnie system będzie dostosowany do działania na fizycznie różnych maszynach.

System będzie składał się z następujących bloków funkcjonalnych:

1. **Nadawcy komunikatów** (urządzenia)
2. **Brama komunikacyjna**: zbiór modułów odpowiadających za komunikację, autoryzację i walidację danych. Odbiera dane z urządzeń komunikujących się po UDP oraz zarządza modułami pomocniczymi:
    - **walidacja danych**: sprawdza dane pod kątem kompletności i prawidłowości
    - **zarządca pamięci**: przechowuje dane z urządzeń sensorycznych, które nadpisywane są każdorazowo po przesłaniu pomiarów
    - **autoryzacja**: sprawdza, czy użytkownik lub urządzenie sensoryczne, komunikujące się z bramą jest dozwolone.
3. **Odbiorcy komunikatów** (użytkownicy).
4. **Test**: moduł zawierający testy jednostkowe i integracyjne
5. **Dziennik zdarzeń**: do zapisywania logów zdarzeń. Będzie zapisywał informacje o nawiązanych połączeniach, odebranych komunikatach oraz o sytuacjach błędnych.

API modułów stanowiących główne bloki funkcjonalne

Użytkownicy

1. API:
    - ask_sensor(sensor_id) - zapytanie o dane i odebranie danych
    - check_status(response) - sprawdza kod statusu otrzymanego w odpowiedzi na ask_sensor. W zależności od odebranego statusu, podjęte zostaną odpowiednie akcje (np. retransmisja danych)
    - register() - prośba o dodanie do listy autoryzowanych użytkowników
    - logout() - prośba o wyrejestrownaie z listy autoryzowanych użytkowników
2. Struktura wysyłanych komunikatów:
    - Prośba o rejestrację użytkownika jako autoryzowanego:
      { "**action**": "register", "**ip_address**": IPv4/IPv6 }
    - Zapytanie o dane określonego urządzenia - sensor_id może być zarówno intem (ostatni bajt adresu ip) lub stringiem (cały adres ip):
      {“**action**”: “ask”, “**sensor_id**”: int/string}
    - Prośba o usunięcie z listy autoryzowanych użytkowników:
      {“**action**”: “logout”, “**ip_address**”: IPv4/IPv6}

Brama komunikacyjna

1. API:
    - start() - wczytuje klucz do autoryzacji, zaufane urządzenia/użytkowników oraz startuje wątki dla użytkowników i sensorów
    - handle_sensor_data() - nawiazuje połączenie z sensorami i wczytuje od nich dane
    - handle_user_requests() - nawiązuje połaczenie z użytkownikami
    - handle_user_connection(conn, addr) - podtrzymuje połączenie z użytkownikiem i przekazuje mu dane
    - save_data_to_file(filename) - zapisuje dane przesłane przez urządzenie w pliku
    - retrive_sensor_data(sensor_id) -> response - odnajduje zapisane dane powiązane z określonym urządzeniem sensorycznym, na podstawie identyfikatora
    - authorize_user(user_adress) -> bool - sprawdza, czy adres IP użytkownika, próbującego nawiązać połączenie, znajduje się na liście autoryzowanych
    - authorize_device(ip_address) -> bool - sprawdza czy adres IP urządzenia sensorycznego, przesyłającego dane, znajduje się na liście urządzeń autoryzowanych

    Dołączony moduł z klasy LogBook:
    - logger.info(event) - zapisuje wydarzenie do dziennika zdarzeń dostępne formy komunikatów to info, error, warning, debug, critical
2. Struktura wysyłanych komunikatów:
    - Odpowiedź na prośbę nawiązania połączenia przez użytkownika:
      {“**status_code**”: int}
    - Wysyłanie danych pomiarowych urządzenia:
      {“**status_code**”: int, “**sensor_ip**”: IPv4/IPv6, **“data”** : { “**humidity**”: int, “**timestamp**”: datetime} }

- Zarejestrowanie lub usunięcie autoryzowanego użytkownika/urządzenia: {“**status_code**”: int}

Wartości **status_codes** mogą być następujące:

- 200 - poprawnie wysłano dane
- 201 - zarejestrowano klienta jako autoryzowanego
- 202 - pomyślne wylogowanie
- 202 - poprawnie nawiązano połączenie
- 400 - niepoprawny/nieobsługiwany format komunikatu
- 401 - nieautoryzowane połączenie (brak klienta w bazie autoryzowanych użytkowników/sensorów)
- 404 - brak urządzenia
- 500 - błąd po stronie serwera
- 507 - nie da się dodać nowego autoryzowanego urządzenia, gdyż przekroczony został limit 1024 urządzeń

Urządzenie sensoryczne

1. API:
   - message_data() - wysyła dane pomiarowe do bramy komunikacyjnej
   - register() - prośba o dodanie do listy autoryzowanych urządzeń
   - logout() - prośba o wyrejestrownaie z listy autoryzowanych urządzeń
   - collect_data() - generuje przykładowe dane zebrane przez sensor
   - increase_packet_id() - obsługuje retransmisję zgubionych pakietów
2. Struktura wysyłanych komunikatów:
   - Wysłanie danych urządzenia:
     {“**action**”: “data”, “**data**”: {“**humidity**”: int, “**timestamp**”: datetime}}
   - Prośba o rejestrację urządzenia jako autoryzowanego:
     { "**action**": "register", "**ip_address**": IPv4/IPv6}
   - Prośba o usunięcie z listy autoryzowanych urządzeń:
     {“**action**”: “logout”, “**ip_address**”: IPv4/IPv6}

Sposób testowania

Testowanie systemu będzie obejmować testy jednostkowe dla poszczególnych bloków funkcjonalnych i testy integracyjne dla sprawdzenia poprawności współpracy między modułami. Wykorzystamy do tego standardowe narzędzia dostępne w środowisku Python, takie jak unittest.

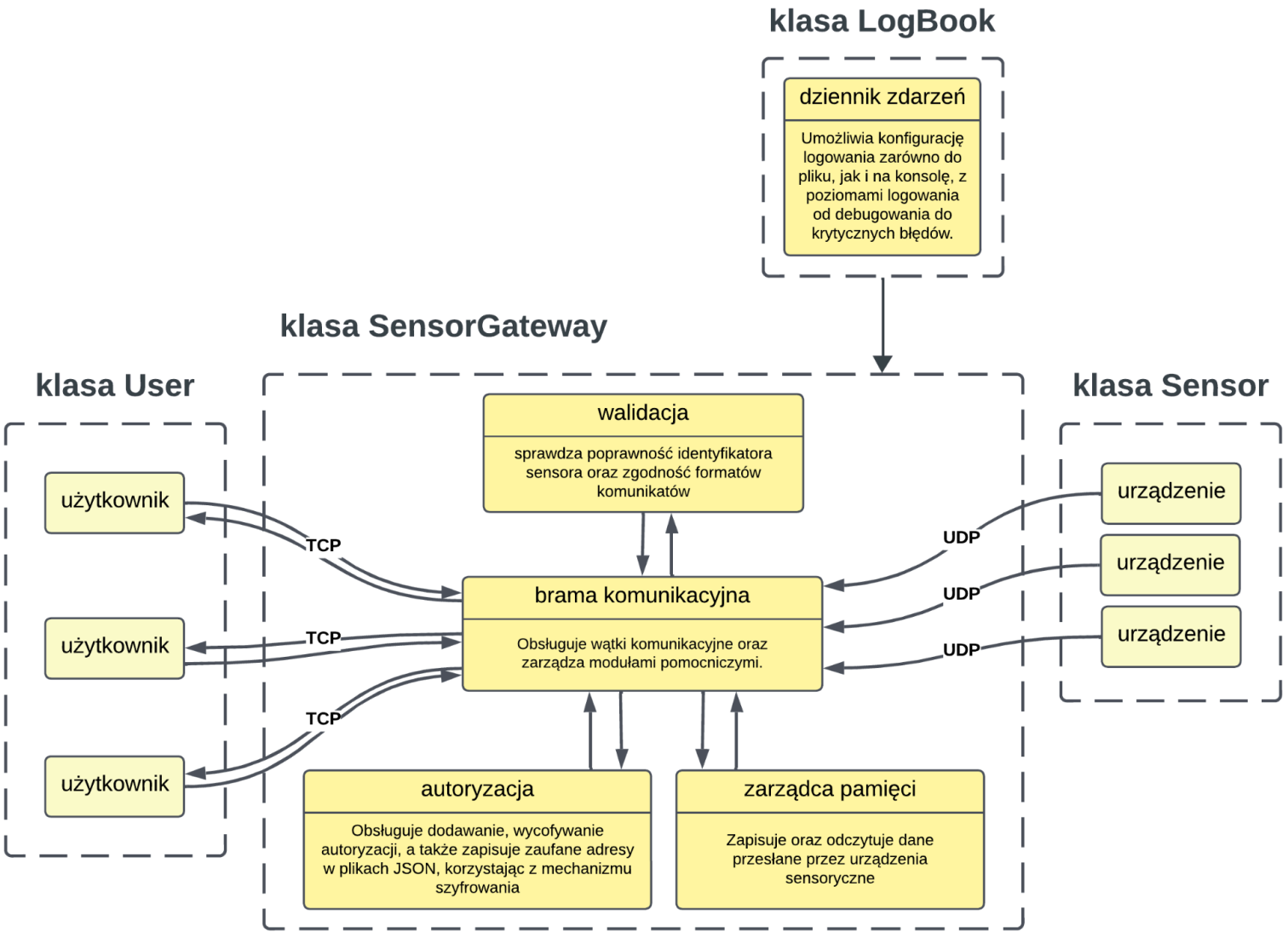
Wstępny podział prac w zespole

Magdalena	Kinga	Marcin	Jakub
urządzenia sensoryczne	użytkownicy	brama - wydobycie i przetwarzanie danych	brama - autoryzacja i logowanie wydarzeń

Przewidywane funkcje do zademonstrowania w ramach odbioru częściowego

- Wysyłka danych przez urządzenia i ich autoryzacja.
- Wyświetlenie odebranych danych w bramie.
- Wysyłka żądań o dane z urządzeń
- Autoryzacja użytkowników

Architektura systemu wraz z podziałem na klasy i moduły



Opis obsługi sytuacji błędnych

1. **Błąd transmisji danych:** jeśli wystąpi błąd podczas transmisji danych urządzenie, które nie otrzyma potwierdzenia odbioru pomiaru, będzie dokonywać transmisji komunikatu (domyślnie 5 razy, jeżeli się nie uda zwraca status_code -1) .

```
Options:
1. Logout sensor
2. Start sending data
3. Quit
Choose an option (1-3): 2
Message sent
[1] Timeout. Resending data.
Message sent
[2] Timeout. Resending data.
Message sent
[3] Timeout. Resending data.
Message sent
[4] Timeout. Resending data.
Message sent
[5] Timeout. Resending data.
No connection with server.
Response:  {'status_code': -1}

Server error.
```

2. **Błąd uwierzytelniania:** jeśli autoryzacja użytkownika lub urządzenia nie powiedzie się, system rejestruje błąd jako log i powiadamia użytkownika lub urządzenie. System następnie blokuje dalsze próby połączenia do momentu poprawnego uwierzytelnienia.

```
2024-01-15 22:36:27,900 - WARNING - Attempted unauthorised connection from user ::ffff:127.0.0.1
2024-01-15 22:36:27,907 - INFO - Closing connection...
```

```
jakubkow1@KubaMateBook:~/psi-23z-z46/projekt$ python3 user/user.py localhos
t 8082
Trying to connect with server...
Connected to gate.
Trying to authorize...
Unauthorized connection.
Unable to communicate with server.
```

3. **Przeciążenie systemu:** jeśli system osiągnie limit 1024 urządzeń lub 256 użytkowników, przy próbie dodania nowego zarejestruje incydent w dzienniku zdarzeń, a następnie będzie odrzucać nowe próby dodania autoryzowanych urządzeń/użytkowników do momentu zwolnienia zasobów.

```python
def add_authorize_device(self, device_address):
    if len(self.active_devices.keys()) < self.MAX_SENSORS:
        self.trusted_devices.add(device_address)
        self.save_trusted_devices()
        self.logger.info(f"Added sensor {device_address} to trusted devices")
    else:
        self.logger.error(
            f"Cannot add sensor {device_address}: too many authorized"
        )
```

Wnioski z wykonanego testowania

Testy jednostkowe przeprowadzone za pomocą frameworka unittest potwierdziły, że system poprawnie autoryzuje użytkowników i urządzenia zarówno dla adresów IPv4, jak i IPv6. Nowy użytkownik może zostać zarejestrowany jako autoryzowany, wysyłać zapytania do sensorów oraz wylogowywać się z systemu. Podobnie, urządzenia sensoryczne mogą zostać zarejestrowane i wysyłać dane pomiarowe, które są poprawnie przetwarzane przez bramę komunikacyjną, a także wylogować się.

Testy wykazały również, że system prawidłowo obsługuje próby nieautoryzowanego dostępu, rejestrując takie zdarzenia i odpowiednio reagując na nie. Mechanizmy logowania zdarzeń działają zgodnie z oczekiwaniami, rejestrując wszystkie istotne informacje w dzienniku zdarzeń (gateway.log).

Przeprowadzone testy potwierdziły, że system "SensNet" działa poprawnie i jest zgodny z założeniami projektowymi. Testy te są kluczowym elementem procesu weryfikacji i walidacji systemu, zapewniającym, że wszystkie komponenty działają prawidłowo przed wdrożeniem systemu w środowisku produkcyjnym.

Podsumowanie

Opis wyniesionych doświadczeń z realizacji projektu

Realizacja projektu była dla nas cennym doświadczeniem, które pozwoliło nam na praktyczne zastosowanie wiedzy zdobytej podczas kursu programowania sieciowego. Praca nad systemem konwersji TCP/UDP pozwoliła nam dobrze zrozumieć, jak działają te protokoły komunikacyjne i jak można je wykorzystać do tworzenia nieco bardziej skomplikowanych systemów sieciowych od niskopoziomowej strony.

Zajęliśmy się również problemem autoryzacji użytkowników i urządzeń. Wykorzystaliśmy mechanizm szyfrowania kluczem adresów IP, zwiększając przy tym bezpieczeństwo systemu sieciowego. Wygenerowaliśmy jednorazowo klucz za pomocą biblioteki cryptography.fernet, który powinien być przechowywany jedynie lokalnie w secret.key. W ten sposób, znacznie utrudniliśmy dostęp niepowołanym użytkownikom / urządzeniem do systemu.

Praca nad projektem pozwoliła nam również zrozumieć, jak ważna jest wielowątkowość w systemach sieciowych. Łączyliśmy się wielowątkowo z użytkownikami i urządzeniami,

obsługując wiele z nich równocześnie. Użyliśmy także mechanizmu Lock do synchronizacji wątków przy jednoczesnym zapisie do pliku i odczycie z niego.

Dowiedzieliśmy się także, że tworzenie systemu komunikacyjnego wymaga wiele myślenia o tym co może źle zadziałać. Poświęciliśmy sporo czasu na rozpatrzenie sytuacji błędnych i testy systemu, zarówno za pomocą unittest, jak i manualne.

Statystyki określające rozmiar stworzonych plików

Plik	Liczba linii kodu
sensor.py	160
user.py	121
gateway.py	342
logbook.py	45
test.py	129
Suma	797

Oszacowanie czasu poświęconego na realizację projektu w godzinach

W ramach projektu spotkaliśmy się wszyscy 3 razy, co zajęło nam łącznie ok. 11 godzin.

Przygotowanie do projektu	Stworzenie projektu wstępnego	Stworzenie finalnej wersji projektu
Wypracowaliśmy ideę rozwiązania, podzieliliśmy pracę, przygotowaliśmy dokumentację wstępną a później sporządziliśmy poprawki do dokumentacji wstępnej	Wspólnie połączyliśmy stworzone przez nas prototypy urządzenia sensorycznego, użytkownika oraz bramy komunikacyjnej. Poprawa błędów w kodzie.	Przetestowaliśmy i uruchomiliśmy finalną wersję ze wszystkimi modułami w środowisku dockerowym. Przeprowadziliśmy wspólnie scenariusze testowe.
5h	3h	3h

Ponadto każdy z nas skupił się na swojej części kodu, pracując indywidualnie ok. 9 godzin na osobę. Razem daje nam to łącznie **ok. 20 godzin** pracy poświęconej na projekt **na osobę**.