TinyAd
Dokumentacja końcowa

Rafał Kulus (Lider),
Damian Kolaska,
Kamil Przybyła

https://bitbucket.org/BlueAlien99/tinyad

7 czerwca 2021

Spis treści

1 Treść zadania

W systemie pracuje stacja zarządzająca i zbiór (do kilkudziesięciu tysięcy) sterowników paneli reklamowych. Stacja multicastowo dystrybuuje treści i harmonogramy ich wyświetlania. Harmonogram określa okresy wyświetlania i czas przechowywania. Sterowniki unicastowo raportują swój stan i zgłaszają błędy. Błąd w transmisji multicastowej obsługiwany jest retransmisją multi- lub unicastową w zależności od liczby otrzymanych komunikatów NAK. System ma pracować w prywatnej sieci IPv6. Stacja zarządzająca może wysyłać unicastowo ważne treści informacyjne do natychmiastowego wyświetlenia przez wybrane sterowniki. Należy też zaprojektować moduł do Wireshark umożliwiający wyświetlanie i analizę zdefiniowanych komunikatów.

2 Format plików

2.1 Konfiguracja dla paneli

`panel_name` i `tags` są opcjonalne i służą do identyfikowania paneli, w celu wyświetlenia ważnych treści. Są wysyłane do stacji zarządzającej podczas rejestrowania panelu. Domyślna nazwa i lokalizacja pliku to `./config.conf`.

```
server_ip=fe80::1234
panel_name=Lobby Front
tags=[oled][huge][orange apricot]
```

2.2 Harmonogram

`expire`, `weekday`, `hour` są opcjonalne. W przypadku braku `expire`, zawartość nigdy nie wygaśnie. W przypadku braku pozostałych dwóch pól, zawartość będzie wyświetlana niezależnie od dnia tygodnia i / lub godziny. Pierwszeństwo ma pierwsza zawartość, która spełnia wszystkie warunki. Dni tygodnia są indeksowane od 1.

```
file=./img.jpg
expire=2021.05.15 19:30
weekday=[1][2][3]
hour=[05:30-08:30][15:00-19:00]
---
file=./vids/video.mp4
expire=2021.06.30 00:00
```

3 Interfejs użytkownika

3.1 Stacja zarządzająca

Stacja zarządzająca posiada prosty CLI.
Dostępne polecenia:

1. `schedule ./my_sched.sched`
   Spowoduje rozesłanie nowego harmonogramu i wymaganych plików do paneli.
   `./my_sched.sched` to ścieżka do pliku zawierającego harmonogram.
2. `panic ./emergency.mkv --name "Lobby Front"`

3. `panic ./emergency.mkv --tags [oled][huge]`
   Spowoduje wysłanie ważnych treści do paneli, których nazwa to `Lobby Front` lub które posiadają co najmniej jeden z podanych tagów. Ważne treści będą wyświetlane tak długo, aż panel nie otrzyma nowego harmonogramu poleceniem `schedule`
4. `bye`
   Spowoduje wyłączenie stacji zarządzającej
5. `top 10`
   Wyświetlenie 10 ostatnich linii logów.

3.2 Panele

Panele nie posiadają interfejsu użytkownika. Wszystkie potrzebne informacje są wczytywane z prostego pliku konfiguracyjnego.

4 Pozostałe założenia funkcjonalne

Proces uruchomienia paneli powinien się ograniczyć tylko i wyłącznie do startu aplikacji klienta, która sama wykona wszystkie wymagane czynności, aby panel mógł zacząć wyświetlać zawartość. Po udanej rejestracji panelu w stacji zarządzającej zostanie do niego natychmiastowo wysłany adres multicast do nasłuchiwania nowych harmonogramów.

5 Założenia niefunkcjonalne

Stacje zarządzające i panele powinny sobie radzić ze wszystkimi przewidzianymi błędami, aby zapewnić nieprzerwaną pracę, stabilność i niezawodność, w szczególności błędy transmisji danych nie powinny uniemożliwić poprawnego działania systemu.

6 Przypadki użycia

6.1 Rejestracja panelu reklamowego

1. Użytkownik uruchamia panel reklamowy (opcjonalnie podając plik konfiguracyjny)
2. Panel rejestruje się, wysyłając komunikat stacji zarządzającej
3. Panel otrzymuje adres multicast do nasłuchiwania nowych harmonogramów

6.2 Zarządzanie harmonogramem oraz wyświetlanymi treściami

1. Użytkownik, za pośrednictwem stacji zarządzającej, modyfikuje informacje o harmonogramie lub treściach
2. Stacja zarządzająca wysyła aktualne dane do paneli
3. Panele wyświetlają treści zgodnie z otrzymanym harmonogramem

6.3 Zlecenie natychmiastowego wyświetlenia informacji

1. Użytkownik, za pośrednictwem stacji zarządzające, zleca natychmiastowe wyświetlenie informacji podanej podgrupie paneli
2. Stacja zarządzająca wysyła treść do natychmiastowego wyświetlenia
3. Panele wyświetlają żądaną treść, ignorując treści wynikające z harmonogramu

7 Obsługa błędów

7.1 Brak harmonogramu

Panel wyświetlał będzie informację o braku harmonogramu.

7.2 Błędny harmonogram

Jeśli harmonogram zawiera błędy składniowe lub opisane w nim pliki nie istnieją, błąd zostanie zgłoszony i harmonogram nie zostanie przesłany do panelu.

7.3 Brak pliku konfiguracyjnego

Wyświetlenie odpowiedniej informacji zwrotnej użytkownikowi i zamknięcie aplikacji.

7.4 Błąd transmisji danych

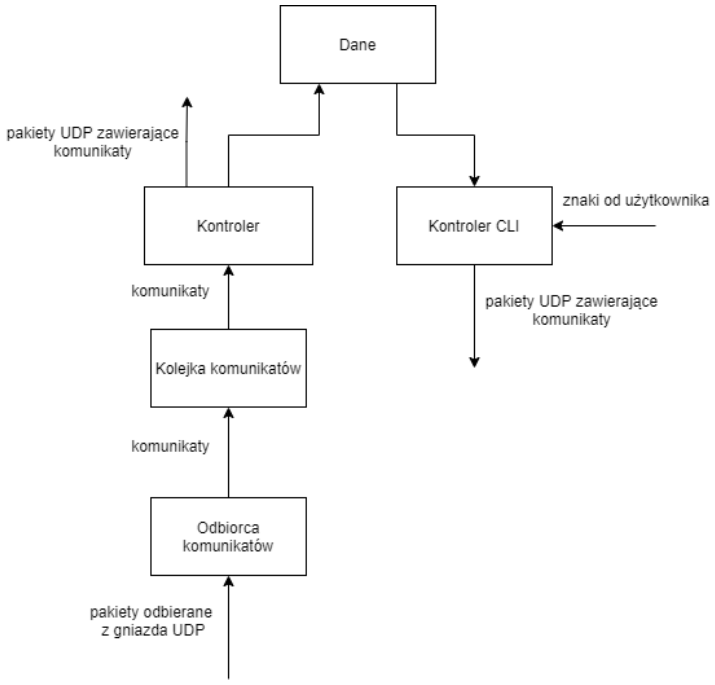
W przypadku, gdy w wyniku błędu sieciowego, jeden lub więcej pakietów zostanie zagubionych, odbiorca wyśle komunikat z prośbą o ponowne przesłanie brakujących segmentów.

8 Środowisko programistyczne

Projekt został zrealizowany na systemie operacyjnym Ubuntu 20.04 LTS, w języku C++17. Do testów została wykorzystana biblioteka GoogleTest, a do debugowania programy gdb, valgrind oraz Wireshark. Do budowania projektu użyto CMake oraz g++. Do logowania skorzystaliśmy z biblioteki Loguru (https://github.com/emilk/loguru).

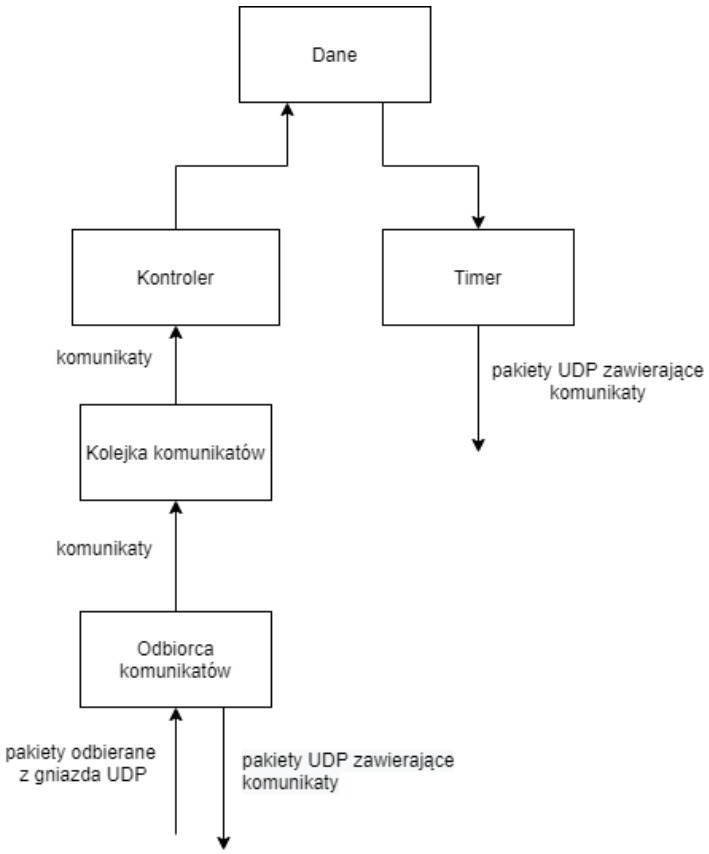
9 Architektura

9.1 Stacja zarządzająca



Odbiorca komunikatów
Odbiera pakiety z gniazda UDP, umieszcza je w kolejce i powraca do dalszego nasłuchiwania.
Kontroler
Przetwarza pakiety z kolejki: deserializuje je na komunikaty, sprawdza ich poprawność i na ich podstawie podejmuje odpowiednie działania.
Kontroler CLI
Odbiera polecenia podawane na standardowe wejście i na ich podstawie podejmuje odpowiednie działania.
Dane
Wszystkie struktury danych używane przez serwer (rejestr paneli, rejestr plików).

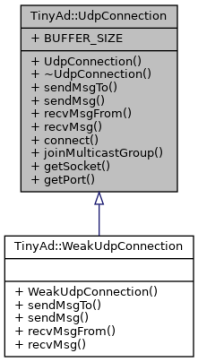
9.2 Panel



Odbiorca komunikatów
Odbiera pakiety z gniazda UDP i umieszcza w kolejce i powraca do dalszego nasłuchiwania. Wysyła wiadomość typu `CONTROL_SLOW_DOWN_TRANSMISSION`, jeśli nastąpi przepełnienie kolejki.
Kontroler
Przetwarza pakiety z kolejki: deserializuje je na komunikaty, sprawdza ich poprawność i na ich podstawie podejmuje odpowiednie działania.
Timer
Okresowo wysyła do stacji wiadomości typu NAK oraz `PanelHello`.
Dane
Wszystkie struktury danych używane przez klienta (harmonogram, konfiguracja, rejestr plików).

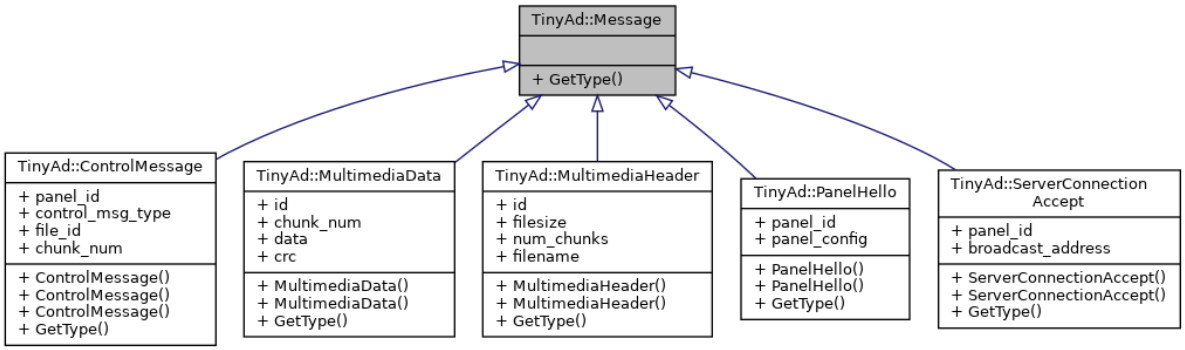
10 Implementacja systemu

W celu ułatwienia implementacji protokołu komunikacji, przygotowano klasę będącą abstrakcją nad UNIXowy interfejs do gniazd UDP (1). Powstała również klasa symulująca wadliwe połączenie, w którym część pakietów jest losowo gubiona. Zastosowany został wzorzec fasady.



Rysunek 1: Diagram klas dla fasady odpowiedzialnej za komunikację UDP

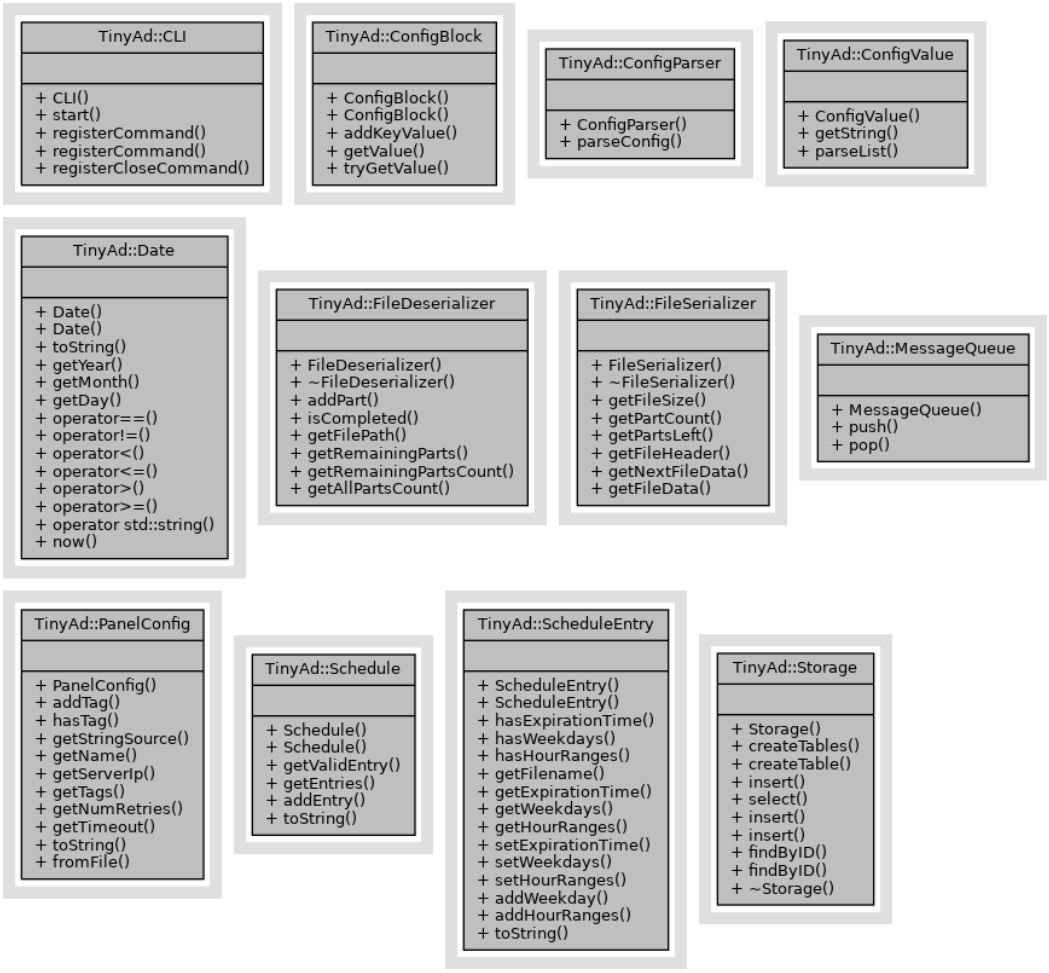
Wszystkie komunikaty są reprezentowane przez klasy potomne wspólnej klasy abstrakcyjnej wymagającej implementacji funkcji, która zwraca typ wiadomości (2). Informacja ta jest wykorzystywana przy deserializacji.



Rysunek 2: Diagram klas przestawiający typy komunikatów

Do pozostałych klas, wykorzystanych w projekcie należą:

- `CLI` – ułatwia tworzenie tekstowego interfejsu użytkownika
- `ConfigBlock` – reprezentuje blok par klucz-wartość w pliku konfiguracyjnym (bloki oddzielone są separatorem)
- `ConfigParser` – parser plików konfiguracyjnych
- `ConfigValue` – reprezentuje wartość odpowiadającą kluczowi w pliku konfiguracyjnym
- `Date` (oraz inne klasy przechowujące czas)
- `FileDeserializer` – tworzy plik na podstawie odebranych pakietów
- `FileSerializer` – generuje komunikaty wykorzystywane do przesyłania danych
- `MessageQueue` – kolejka przechowująca odbierane pakiety z wbudowaną synchronizacją
- `PanelConfig` – reprezentuje plik konfiguracyjny panelu
- `Schedule` – reprezentuje harmonogram
- `ScheduleEntry` – reprezentuje pojedynczy wpis w harmonogramie
- `Storage` – interfejs do bazy danych (nieużywany)



Rysunek 3: Pozostałe klasy wykorzystywane w projekcie

11 Komunikaty

11.1 Struktura komunikatów

11.1.1 MultimediaHeader

Komunikat wysyłany przez stację zarządzającą. Zawiera metadane pliku, który ma zostać przesłany.

```
struct MultimediaHeader {
    std::string id;
    uint32_t filesize;
    uint32_t num_chunks;
    std::string filename;
};
```

- `id` – unikalny identyfikator pliku
- `filesize` – rozmiar pliku
- `num_chunks` – liczba fragmentów, na które podzielony zostanie plik
- `filename` – nazwa pliku

11.1.2 MultimediaData

Zawiera dane stanowiące fragment pliku. Wysyłane przez stację zarządzającą i odbierane przez panele.

```
struct MultimediaData {
    std::string id;
    uint32_t chunk_num;
    std::string data;
    uint32_t crc;
};
```

- `id` – unikalny identyfikator pliku
- `chunk_num` – numer fragmentu pliku
- `data` – dane pliku
- `crc` – suma kontrolna

11.1.3 ControlMessage

Komunikat służący do transmisji wiadomości kontrolnych. Są wysyłane zarówno przez stację jak i przez panele.

```
struct ControlMessage {
    std::string panel_id;
    ControlMsgType control_msg_type;
    std::string file_id;
    int chunk_num;
};
```

- `panel_id` – unikalny identyfikator panelu
- `control_msg_type` – typ wiadomości kontrolnej
- `file_id` – unikalny identyfikator pliku
- `chunk_num` – numer fragmentu pliku

Do typów wiadomości kontrolnych należą:

- `CONTROL_NAK` – komunikat informaujący o brakującym fragmencie danych pliku; wysyłany przez panel
- `CONTROL_FILE_HEADER_REQUESTED` – informuje o brakującym nagłówku pliku; wysyłany przez panel
- `CONTROL_SLOW_DOWN_TRANSMISSION` – wysyłany przez panel, gdy kolejka nieobsłużonych komunikatów się zapełni
- `CONTROL_FILE_NOT_FOUND` – wysyłany przez serwer, gdy pożądany plik nie istnieje

11.1.4 PanelHello

Wysyłany przez panel w celu rejestracji w stacji zarządzającej. Służy również za keep-alive.

```
struct PanelHello {
    std::string panel_id;
    std::string panel_config;
};
```

- `panel_id` – unikalny identyfikator panelu
- `panel_config` – konfiguracja panelu

11.1.5 ServerConnectionAccept

Komunikat wysyłany przez serwer, jako odpowiedź na `PanelHello`.

```
struct ServerConnectionAccept {
    std::string panel_id;
    std::string broadcast_address;
};
```

- `panel_id` – unikalny identyfikator panelu
- `broadcast_address` – adres multicast do nasłuchiwania przez panel

11.2 Kodowanie komunikatów

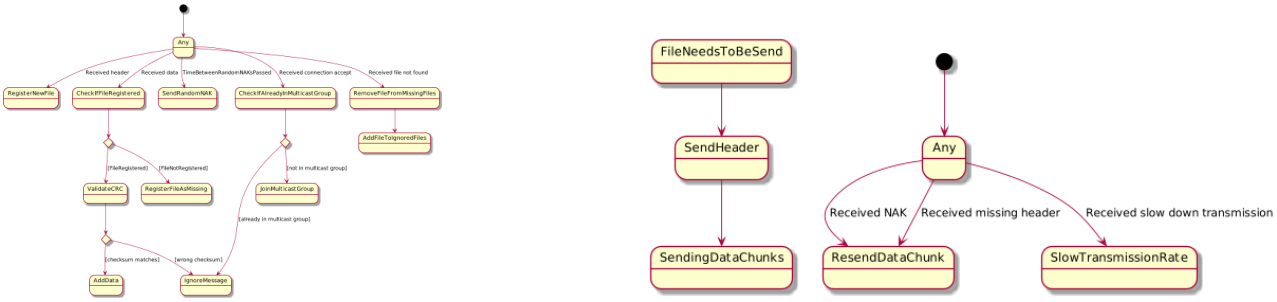
Przed wysyłaniem komunikatu dodawane jest na początku pole będące liczbą 8-bitową bez znaku informujące o typie wiadomości. Informacja ta jest konieczna, aby rozróżnić typ odebranego komunikatu. Ciągi znaków poprzedzane są ich długością.
Rozmiar danych przesyłanych w komunikacie `MultimediaData` jest ograniczony do 512 bajtów.

12 Przesyłanie danych

Serwer po uruchomieniu oczekuje na komunikaty `PanelHello` od paneli. Po otrzymaniu takiego komunikatu rejestruje panel. Przy zmianie harmonogramu serwer ładuje wszystkie pliki zawarte w harmonogramie i wysyła je wraz z harmonogramem do wszystkich zarejestrowanych paneli.

Panel po otrzymaniu nagłówka pliku rejestruje go jako aktywny. Panele posiadają oddzielny wątek sprawdzający, czy istnieją aktywne pliki, które nie zostały w całości pobrane, i wysyłający komunikaty NAK dla losowych fragmentów takich plików.

12.1 Pliki multimedialne



(a) Klient (b) Serwer

Rysunek 4: Automaty stanów dla niezawodnej transmisji plików

Pliki multimedialne są zbyt duże, żeby przesyłać je w jednym datagramie. Należy je zatem podzielić. Przesyłając plik w kawałkach, może się okazać, że część pakietów nie dotrze albo pakiety dotrą w innej kolejności. Aby rozwiązać ten problem numerujemy pakiety.

Transmisja pliku rozpoczyna się od wysłania przez serwer wiadomości z nagłówkiem (`MultimediaHeader`). Nagłówek zawiera informacje takie jak: nazwa pliku, jego rozmiar, liczba części, na które zostanie

podzielony, oraz identyfikator (UUID), po którym rozpoznajemy, czy pakiety dotyczą danego pliku. Po wysłaniu nagłówka rozpoczyna się transmisja pliku. W ramach transmisji wysyłane są wiadomości zawierające identyfikator pliku, numer części, dane i sumę kontrolną (`MultimediaData`). Klient odbierając pakiet danych sprawdza, czy dotyczy on obecnie przesyłanego pliku (porównując identyfikator). Następnie weryfikuje sumę kontrolną. Jeśli suma się nie zgadza, ignoruje pakiet (komunikat NAK zostanie wysłany później). Serwer sporadycznie (w losowych odstępach czasu) wysyła komunikaty NAK, dotyczące losowych, brakujących fragmentów plików.

13 Sposób testowania

W celu ułatwienia testowania, przygotowany został moduł do Wiresharka, umożliwiający badanie, czy stacja zarządzająca i panele wysyłają odpowiednie komunikaty oraz czy ich zawartość jest zgodna z oczekiwaniami.

Przydatne również są logi stacji zarządzającej, z których można odczytać, jakie akcje zostały wykonane przez stację lub panel.

Sprawdzając, czy aplikacja nie jest podatna na ataki, celowo preparowaliśmy komunikaty, analizując jak zachowa się system.

14 Wnioski z wykonanego testowania

1. W czasie działania systemu możemy zaobserwować w programie Wireshark, że odpowiednie komunikaty są wysyłane. Wyjątkiem są komunikaty typu ServerConnectionAccept, których – z jakiegoś nieznanego powodu – nie możemy zaobserwować w programie, mimo że są one wysyłane, gdyż system działa i loguje ich otrzymanie.
2. Szczegółowe logowanie wydarzeń okazało się przydatne przy sprawdzaniu działania systemu i do szybkiego lokalizowania błędów.
3. System jest częściowo odporny na preparowanie komunikatów (patrz sekcja 15).
4. Nawet przy wadliwym połączeniu, w którym część pakietów jest gubiona, system działa prawidłowo.

15 Analiza podatności bezpieczeństwa

Ze względu na brak autoryzacji stacji zarządzającej oraz paneli, każda osoba, mająca dostęp do sieci, wewnątrz której działa system, może przesłać dowolny harmonogram panelowi i zostanie on wyświetlony. Możliwe jest również otrzymywanie treści reklamowych mimo tego, że klientem nie jest panel.

System został zabezpieczony przed sytuacjami, w których złośliwy użytkownik podaje błędne dane w polach komunikatu. Istnieje na przykład limit rozmiaru pliku (4GB), który może zostać przesłany do panelu oraz usuwane są ścieżki relatywne z parametrów zawierających nazwy plików, aby uniknąć nadpisywania ważnych plików znajdujących się na systemie panelu. Weryfikowane są również pakiety zawierające dane, aby niemożliwym było wpisywanie danych w niepoprawnym miejscu pliku (umożliwiłoby to tworzenie bardzo dużych, "dziurawych" plików na systemie paneli).

W przypadku, gdy panel nigdy nie otrzyma brakującego fragmentu pliku albo informacji, że plik

nie istnieje, ciągle będzie wysyłał komunikaty NAK, co potencjalnie mogłoby zostać wykorzystane do przeprowadzenia ataku DDoS.

16 Sposób demonstracji rezultatów

16.1 Demonstracja działania dystrybucji harmonogramów oraz treści

1. Uruchomienie stacji zarządzającej
2. Podłączenie panelu do działającej stacji zarządzającej
3. Zlecenie przesłania harmonogramu oraz treści za pośrednictwem interfejsu tekstowego stacji zarządzającej
4. Zaprezentowanie panelu wyświetlającego wskazane treści zgodnie z harmonogramem

16.2 Demonstracja działania dystrybucji ważnych treści

1. Uruchomienie stacji zarządzającej
2. Podłączenie kilku paneli o różnych nazwach i tagach do działającej stacji zarządzającej
3. Zlecenie wyświetlenia pilnej treści panelowi o wskazanej nazwie
4. Zaprezentowanie panelu wyświetlającego żądaną treść
5. Zlecenie wyświetlenia pilnej treści podzbiorowi paneli o wskazanym tagu
6. Zaprezentowanie paneli wyświetlających żądaną treść
7. Wysłanie żądania wyświetlania treści zgodnie z harmonogramem
8. Pokazanie paneli wyświetlających treści według harmonogramu
9. Próba zlecenia wyświetlenia ważnej treści nieistniejącemu panelowi

17 Podział prac w zespole

- **Rafał Kulus** – warstwa abstrakcji na systemowe gniazda UDP (multicast, unicast), wielowątkowa obsługa żądań, wysyłanie i odbieranie harmonogramów (wraz z treściami), rejestracja panelu w stacji zarządzającej
- **Damian Kolaska** – serializacja i deserializacja pakietów wraz z testami, odbieranie i nadawanie komunikatów, warstwa abstrakcji na bazę danych, implementacja kontroli szybkości transmisji
- **Kamil Przybyła** – moduł do Wiresharka, parsowanie plików konfiguracyjnych, obsługa harmonogramów, interfejs tekstowy, wysyłanie pilnych treści

18 Harmonogram prac

- Do 27 kwietnia – dokumentacja wstępna
- 19-26 maja – działająca dystrybucja harmonogramów oraz wyświetlanie treści przez panele
- Do 1 czerwca – wysyłanie ważnych treści oraz komunikacja z warstwą składowania danych
- Do 7 czerwca – dokumentacja końcowa

19 Podsumowanie

19.1 Wnioski i doświadczenia

(*Damian Kolaska*) W przypadku kiedy dwa moduły powstają równolegle warto się upewnić, że kiedy zostaną ukończone da się je zintegrować oraz że każdy z nich jest w ogóle potrzebny. Mam

tutaj na myśli bazę danych, do której tworzyłem warstwę abstrakcji. Po ukończeniu modułu okazało się, że jego integracja z istniejącym już i zintegrowanym z aplikacją FileDeserializerem byłaby trudna, a zysk potencjalnie bardzo mały. Zakładałem, że baza mogłaby się przydać z kilku powodów:

- nie musielibyśmy dzielić plików za każdym startem aplikacji
- ładowaniem danych do pamięci mogłaby zarządzać sama baza danych
- moglibyśmy przechowywać dodatkowe metadane

Ostatecznie okazało się, że żadna z tych potrzeb się nie zmaterializowała, a moduł stał się niepotrzebny.

(*Kamil Przybyła*) Prawdopodobnie zbyt późno rozpoczęliśmy pracę nad implementacją protokołu komunikacyjnego, przez co pod koniec projektu niezbędne były szybkie poprawki. Na etapie pisania dokumentacji wstępnej trudno jest przewidzieć, jakie problemy ma zaprojektowany protokół, i od razu wpaść na rozwiązanie idealne. Zbyt późno zaczęliśmy też myśleć o potencjalnych podatnościach systemu. Być może pamiętając o tym wcześniej protokół byłby lepiej zaprojektowany i udałoby nam się zaimplementować autoryzację serwera i paneli.

(*Rafał Kulus*) Napisanie fasady na gniazda było super zabawą. Początkowo było ciężko i nic nie działało, ale jak już zaczęło, to wszystko stało się relatywnie proste. Projekt rzeczywiście był bardzo czasochłonny, jednak niemniej ciekawy. Szkoda, że zaczął się tak późno. Gdyby rozpoczął się wcześniej, bylibyśmy w stanie go o wiele bardziej rozbudować i lepiej dopracować. Zdecydowanie był to jeden z większych projektów na studiach, ale dał równie dużo satysfakcji z pisania.

19.2 Rozmiary stworzonych plików

```
136 src/cli.cpp
 85 src/config_parser.cpp
 63 src/file_deserializer.cpp
 58 src/file_serializer.cpp
145 src/message_deserializer.cpp
 32 src/message_queue.cpp
 73 src/panel_config.cpp
171 src/schedule.cpp
293 src/time.cpp
139 src/udp_connection.cpp
166 src/client/main.cpp
235 src/server/main.cpp
 45 include/cli.hpp
114 include/common.hpp
 13 include/config.hpp
 71 include/config_parser.hpp
 45 include/exceptions.hpp
 40 include/file_deserializer.hpp
 37 include/file_serializer.hpp
 29 include/message_deserializer.hpp
119 include/message.hpp
 35 include/message_queue.hpp
 76 include/message_serializer.hpp
 44 include/network_utils.hpp
 57 include/panel_config.hpp
 80 include/schedule.hpp
171 include/storage.hpp
183 include/time.hpp
 56 include/udp_connection.hpp
165 include/client/utils.hpp
193 include/server/utils.hpp

 31 test/apps/database_client.cpp
 41 test/apps/schedule_parsing.cpp
168 test/unit/config_parser.cpp
  9 test/unit/main.cpp
 32 test/unit/network_utils.cpp
 38 test/unit/panel_config.cpp
 83 test/unit/schedule.cpp
```

```
 103 test/unit/serial_deserial.cpp
 262 test/unit/time.cpp
3936 razem
```

19.3 Szacowany czas pracy

- Rafał Kulus – 60h
- Damian Kolaska – 40h
- Kamil Przybyła – 40h