Warszawa, dnia 06.01.2016

System dystrybucji i uruchamiania spotów reklamowych - projekt końcowy

Zespół*: Damian Ostrowski, Marek Bednarski, Cezary Grunwald, Piotr Rżysko*
Lider projektu: *Piotr Rżysko*

Zaktualizowana dokumentacja końcowa

1. Treść zadania

W systemie pracuje stacja zarządzająca i zbiór sterowników paneli reklamowych. Stacja multicastowo dystrybuuje filmy i harmonogramy ich wyświetlania. 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 sieci IPv6 i IPv4 (bez NAT). Należy też zaprojektować moduł do Wireshark umożliwiający wyświetlanie i analizę zdefiniowanych komunikatów.

2. Nazwa projektowanego systemu

AdvertCast

3. Przyjęte założenia funkcjonalne i niefunkcjonalne

1. Założenia funkcjonalne

- Administrator serwera w pliku konfiguracyjnym może zdefiniować listę filmów wraz z harmonogramem ich wyświetlania. Plik musi być zapisany w formacie CSV. Reszta parametrów działania aplikacji podawana jest jako parametry uruchomienia.
- Opis pojedynczego filmu będzie składała się ze ścieżki na dysku lokalnym serwera do pliku z filmem oraz znacznikiem czasu
- Administrator będzie mógł zdefiniować adres grupy multicastowej do której wysyłane będą filmy.
- Administrator sterowników paneli reklamowych będzie mógł zdefiniować adres serwera oraz port służący do odbierania komunikatów.

2. Założenia niefunkcjonalne

- Maksymalny rozmiar pliku z filmem to 15MB.
- Minimalna długość filmu to 1min.
- Akceptowane formaty filmów to: .avi, .mp4, .mov.
- System powinien oferować niezawodność transmisji na poziomie 90%.
- Wszystkie pracujące w systemie stacje powinny mieć zainstalowany system Linux.
- Wszystkie pracujące w systemie stacje powinny mieć dostęp do internetu.
- Wszystkie pracujące w systemie stacje powinny mieć włączoną synchronizację czasu z internetem.
- Oddzielne kanały komunikacji dla danych oraz komunikatów.
- Komunikaty wysyłane przez TCP.
- Wykrywanie błędów przy wykorzystaniu numerowania datagramów.

4. Podstawowe przypadki użycia

1. Uruchomienie serwera

1. Administrator wprowadza do pliku konfiguracyjnego dane dotyczące plików filmów i harmonogramy wyświetlania ich - jest to plik w formacie CSV.
2. Administrator uruchamia program serwera podając jako argumenty:interfejs sieciowy dla multicastu, adres grupy multicastowej, port UDP. port TCP oraz ścieżka do pliku CSV z konfiguracją filmów. 224.0.0.1 -u 8888 -t 5555 -f.
3. Aplikacja stwierdza poprawność wprowadzonych danych
4. Uruchamiany jest proces dystrybucji filmów

Alternatywnie:

3a. Aplikacja stwierdza niepoprawność danych
    1. Aplikacja kończy działanie

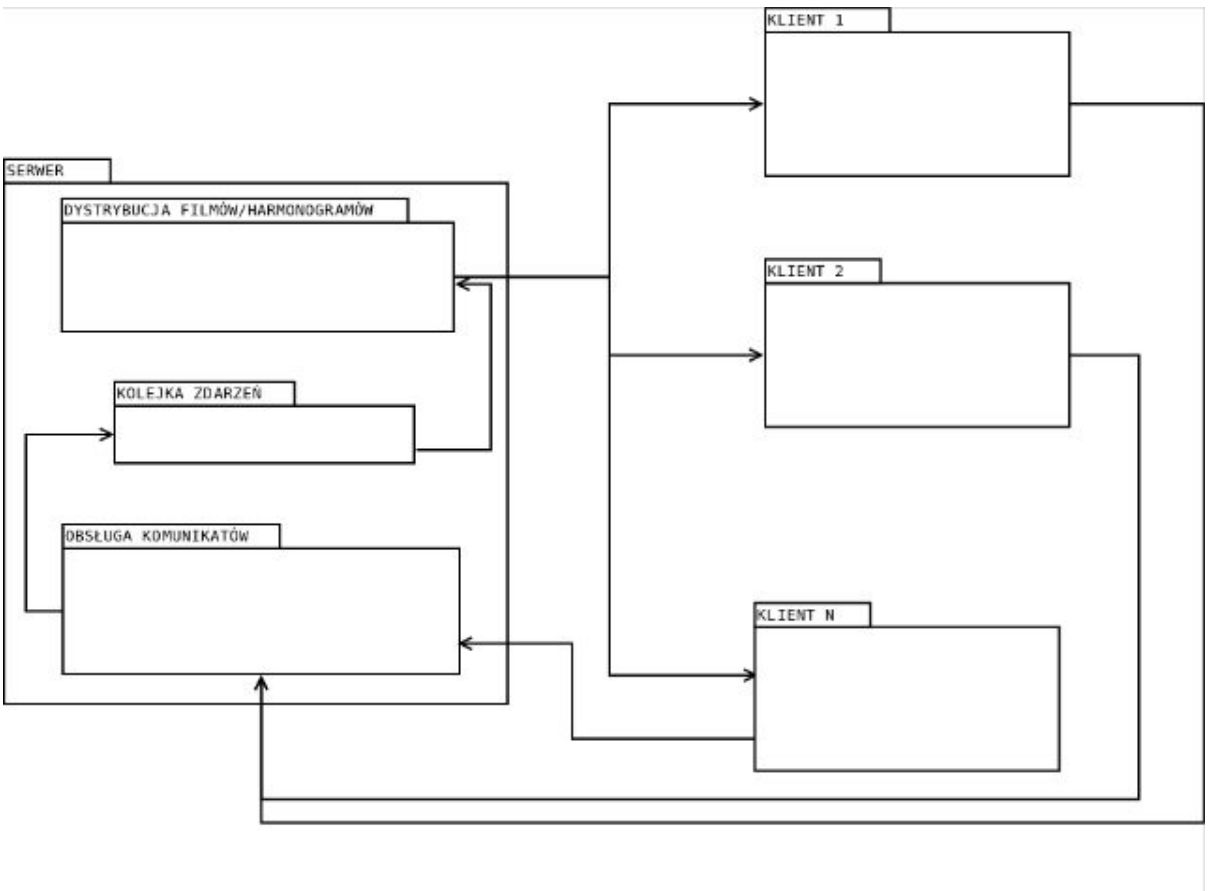
3a. Aplikacja stwierdza niepoprawność adresu
    1. Aplikacja kończy działanie

5. Wybrane środowisko sprzętowo-programowe i narzędziowe

1. Środowisko sprzętowo-programowe
    - system został zrealizowany w środowisku uniksowym (Linux)
    - aplikacja została napisana w języku C++ z wykorzystaniem API gniazd BSD
    - w projekcie wykorzystywano biblioteki STL oraz Google Test

1. Środowisko narzędziowe
    - kod systemu był tworzony z wykorzystaniem IDE CLion
    - debugowanie prowadzono z wykorzystaniem narzędzi dostarczonych przez środowisko CLion oraz programu Wireshark
    - do budowy aplikacji wykorzystano system CMake, który jest zintegrowany ze środowiskiem CLion (opcjonalnie może zostać wykorzystany uniksowy (g)make)
    - do budowy środowiska testowego wykorzystano kontenery linuksowe zarządzane przez program Docker

6. Architektura rozwiązania



- **Kolejka zdarzeń**

  Jest to kolejka priorytetowa, której elementami są kolejne filmy które powinny zostać wysłane do sterowników paneli reklamowych. Filmy w kolejce są uporządkowane według zaplanowanego czasu ich odtwarzania. Początkowo w kolejce znajdują się filmy które zostały wprowadzone jako konfiguracja serwera.

- **Obsługa komunikatów**

  Moduł ten odpowiedzialny jest za odbiór i przetwarzanie komunikatów pochodzących od sterowników paneli reklamowych.
  Moduł ma za zadanie oczekiwać przez określony czas (ustalony podczas testów) na komunikaty od wszystkich sterowników. Jeżeli po ww. czasie nie uzyska odpowiedzi od danego sterownika uznaje go jako nieaktywny i w kolejnych iteracjach ignoruje komunikaty od niego.
  Moduł na podstawie liczby otrzymanych komunikatów NAK decyduje w jaki sposób należy obsłużyć retransmisję. W przypadku wystąpienia błędów dodaje do kolejki film do wysłania z informacją o sposobie retransmisji.
  Po wykonaniu ww. działań moduł wznawia działanie dystrybucji filmów i harmonogramów, sam natomiast zostaje zawieszony.

- **Dystrybucja filmów/harmonogramów**

  Moduł ten odpowiedzialny jest za dostarczenie do zdefiniowanej w elemencie kolejki grupy odbiorników filmu który znajduje się na początku kolejki.
  Moduł powinien odrzucać filmy których czas wyświetlania już minął. W przypadku braku filmów w kolejce kończy działanie serwera.
  Po wysłaniu filmu wznawia działanie obsługi komunikatów, sam natomiast zostaje zawieszony.

- **Klient**

  Aplikacja klienta jest odpowiedzialna za odbieranie filmów oraz sprawdzanie błędów transmisji. W przypadku wystąpienia błędu będzie wysyłała komunikat NAK do serwera przez specjalnie przygotowany kanał. W przypadku braku błędu musi wysłać komunikat ACK.

7. Sposób testowania

Program testowano z pomocą serwisu TravisCI. Są to testy jednostkowe napisane dla każdego elementu składowego z pomocą biblioteki Google Test. Testy są uruchamiane dla każdego commita.

8. Sposób demonstracji rezultatów, tj. scenariusze testów akceptacyjnych do zaprezentowania przy odbiorze projektu.

Testy akceptacyjne polegają na realizacji prostych scenariuszy:

- Uruchomienie systemu za pomocą kontenerów (Docker) oraz przeprowadzenie transmisji do pojedynczego klienta
- Uruchomienie systemu za pomocą kontenerów (Docker) oraz przeprowadzenie transmisji do wiele klientów
- Uruchomienie systemu za pomocą kontenerów (Docker) oraz przeprowadzenie transmisji do wielu klientów z dołączaniem

W testach używamy kontenerów linuksowych dla serwera oraz każdego klienta, kontenery posiadają interfejsy sieciowe i umożliwiają łącznie się między nimi.

9. Podział prac w zespole

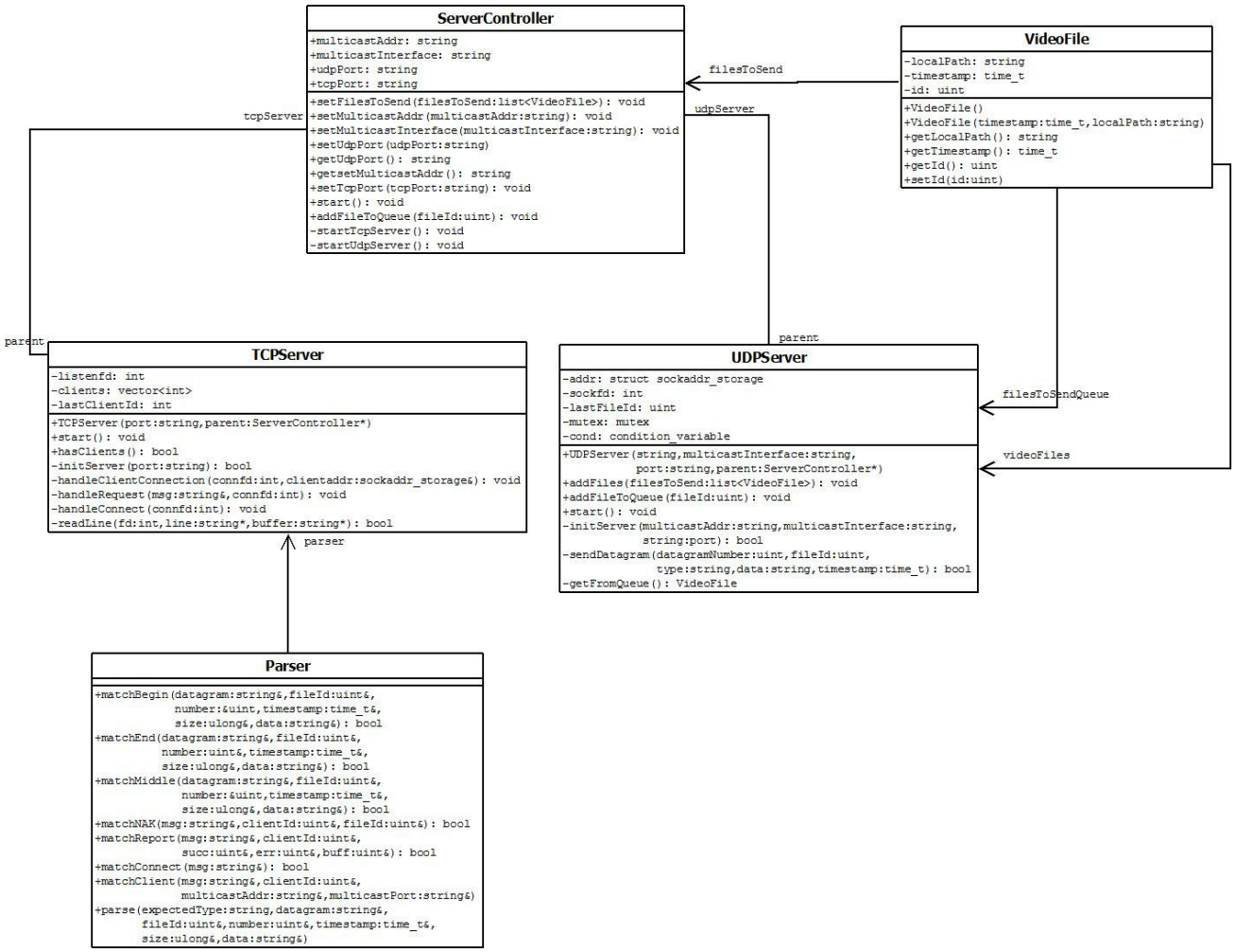
- ***Damian Ostrowski***
  - ***Organizacja dokumentacji kodu***
  - ***Programowanie modułów stacji zarządzającej oraz sterowników***
- ***Marek Bednarski***
  - ***Organizacja testów, obsługa CI***
  - ***Programowanie modułów stacji zarządzającej oraz sterowników***
- ***Cezary Grunwald***
  - ***Organizacja dokumentacji projektowej***
  - ***Programowanie modułów stacji zarządzającej oraz sterowników***
- ***Piotr Rżysko***
  - ***Organizacja pracy zespołu***
  - **Moduł do Wireshark umożliwiający wyświetlanie i analizę zdefiniowanych komunikatów.**
  - ***Programowanie modułów stacji zarządzającej oraz sterowników***

10. Adres projektu na serwerze kontroli wersji

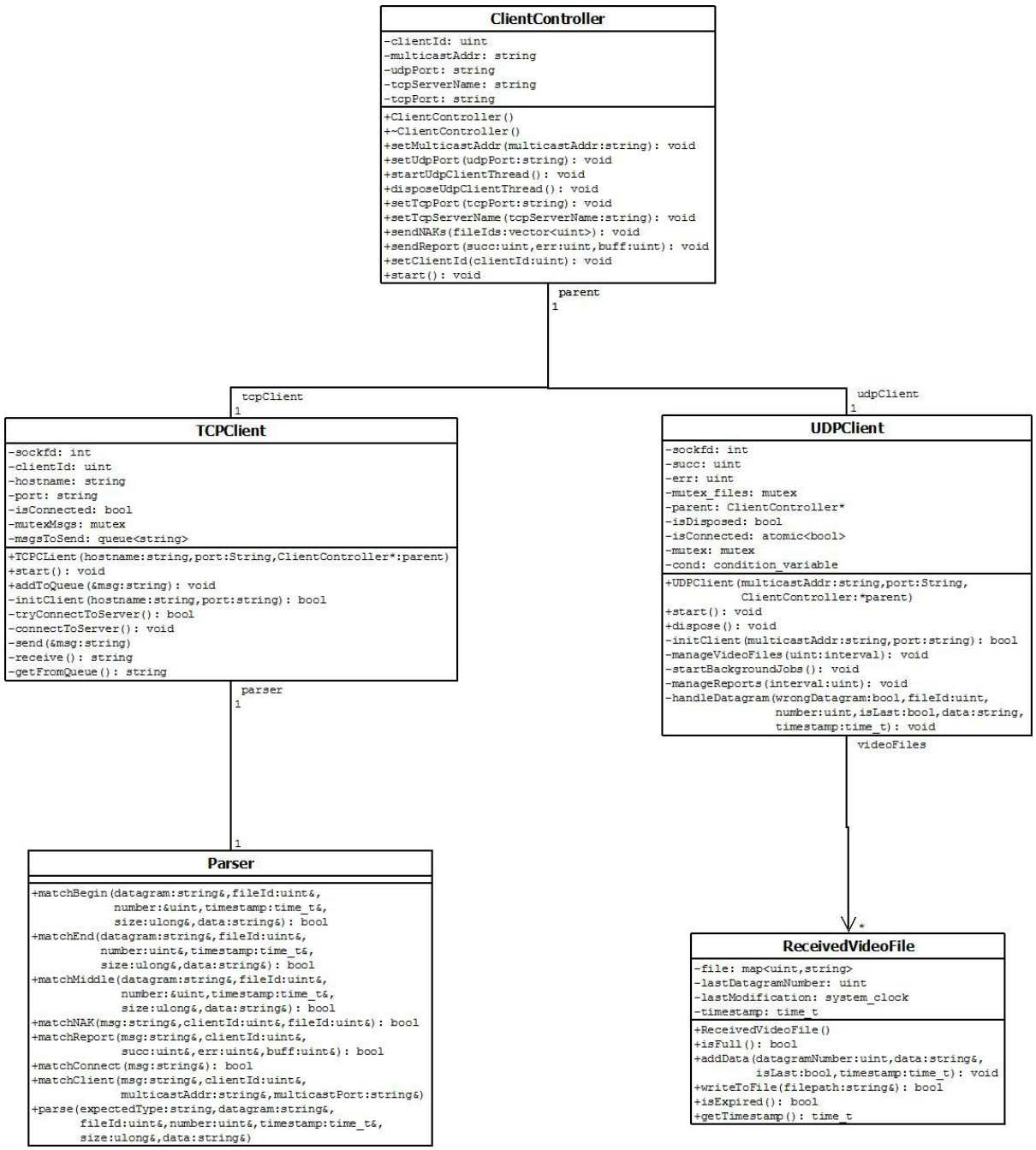
https://github.com/piotreq53/TIN

11. Diagramy klas:

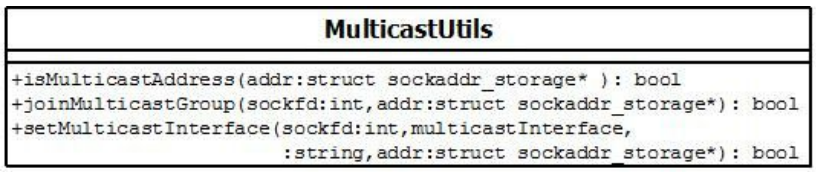
1. Serwer

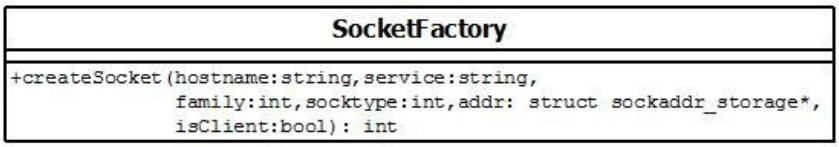


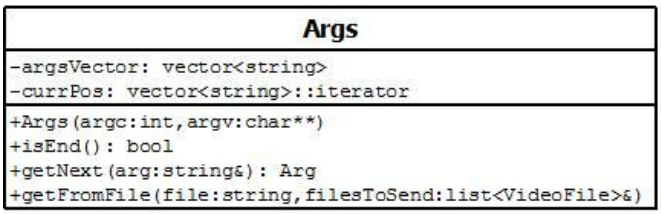
2. Klient


ClientController
-clientId: uint
-multicastAddr: string
-udpPort: string
-tcpServerName: string
-tcpPort: string
+ClientController()
+~ClientController()
+setMulticastAddr(multicastAddr:string): void
+setUdpPort(udpPort:string): void
+startUdpClientThread(): void
+disposeUdpClientThread(): void
+setTcpPort(tcpPort:string): void
+setTcpServerName(tcpServerName:string): void
+sendNAKs(fileIds:vector<uint>): void
+sendReport(succ:uint,err:uint,buff:uint): void
+setClientId(clientId:uint): void
+start(): void
parent
1
tcpClient
1
udpClient
1
TCPClient
-sockfd: int
-clientId: uint
-hostname: string
-port: string
-isConnected: bool
-mutexMsgs: mutex
-msgsToSend: queue<string>
+TCPClient(hostname:string,port:String,ClientController*:parent)
+start(): void
+addToQueue(&msg:string): void
-initClient(hostname:string,port:string): bool
-tryConnectToServer(): bool
-connectToServer(): void
-send(&msg:string)
-receive(): string
-getFromQueue(): string
UDPClient
-sockfd: int
-succ: uint
-err: uint
-mutex_files: mutex
-parent: ClientController*
-isDisposed: bool
-isConnected: atomic<bool>
-mutex: mutex
-cond: condition_variable
+UDPClient(multicastAddr:string,port:String,
ClientController:*parent)
+start(): void
+dispose(): void
-initClient(multicastAddr:string,port:string): bool
-manageVideoFiles(uint:interval): void
-startBackgroundJobs(): void
-manageReports(interval:uint): void
-handleDatagram(wrongDatagram:bool,fileId:uint,
number:uint,isLast:bool,data:string,
timestamp:time_t): void
parser
1
1
videoFiles
*
Parser
+matchBegin(datagram:string&,fileId:uint&,
number:&uint,timestamp:time_t&,
size:ulong&,data:string&): bool
+matchEnd(datagram:string&,fileId:uint&,
number:uint&,timestamp:time_t&,
size:ulong&,data:string&): bool
+matchMiddle(datagram:string&,fileId:uint&,
number:&uint,timestamp:time_t&,
size:ulong&,data:string&): bool
+matchNAK(msg:string&,clientId:uint&,fileId:uint&): bool
+matchReport(msg:string&,clientId:uint&,
succ:uint&,err:uint&,buff:uint&): bool
+matchConnect(msg:string&): bool
+matchClient(msg:string&,clientId:uint&,
multicastAddr:string&,multicastPort:string&)
+parse(expectedType:string,datagram:string&,
fileId:uint&,number:uint&,timestamp:time_t&,
size:ulong&,data:string&)
ReceivedVideoFile
-file: map<uint,string>
-lastDatagramNumber: uint
-lastModification: system_clock
-timestamp: time_t
+ReceivedVideoFile()
+isFull(): bool
+addData(datagramNumber:uint,data:string&,
isLast:bool,timestamp:time_t): void
+writeToFile(filepath:string&): bool
+isExpired(): bool
+getTimestamp(): time_t

3. Pozostałe


MulticastUtils
+isMulticastAddress(addr:struct sockaddr_storage*): bool
+joinMulticastGroup(sockfd:int,addr:struct sockaddr_storage*): bool
+setMulticastInterface(sockfd:int,multicastInterface,
:string,addr:struct sockaddr_storage*): bool


SocketFactory
+createSocket(hostname:string,service:string,
family:int,socktype:int,addr: struct sockaddr_storage*,
isClient:bool): int


Args
-argsVector: vector<string>
-currPos: vector<string>::iterator
+Args(argc:int,argv:char**)
+isEnd(): bool
+getNext(arg:string&): Arg
+getFromFile(file:string,filesToSend:list<VideoFile>&)

12. Definicje komunikatów:

Komunikat	Struktura	Opis
Komunikacja TCP		
NAK	NAK clientId fileId\n	Komunikat NAK w komunikacji klient -> serwer
REPORT	REPORT clientId succ err buff\n	Komunikat wysyłany od klienta do serwera. Zawiera informacje o stanie klienta: jego id, liczbę poprawnie odebranych filmów, liczbę niepoprawnie odebranych filmów, liczbę filmów oczekujących.
CONNECT	CONNECT\n	Wysyłany od klienta do serwera w celu nawiązania połączenia.
CLIENT	CLIENT multicastAddress multicastPort\n	
Komunikacja UDP		
BEGIN	BEGIN fileId datagramNumber timestamp data_size\n data	Komunikat wysyłany od serwera do klienta, oznaczający pierwszy datagram pliku. Zawiera numer filmu, numer datagramu, timestamp, rozmiar danych oraz dane
MIDDLE	MIDDLE fileId datagramNumber timestamp data_size\n data	Komunikat wysyłany od serwera do klienta, oznaczający wewnętrzny datagram pliku. Wysyłane pola są analogiczne do komunikatu BEGIN
END	END\n	Komunikat wysyłany od serwera do klienta,

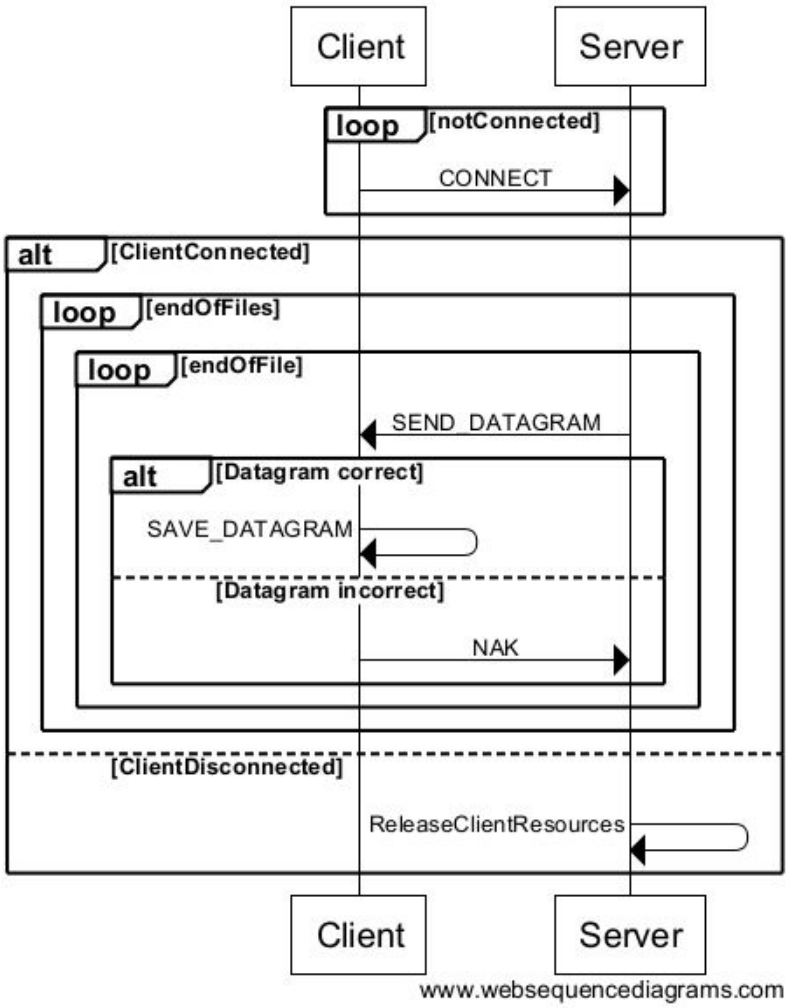
		oznaczający ostatni datagram pliku.

13. Opis zachowania poszczególnych podmiotów komunikacji

Zarówno po stronie klienta jak i po stronie serwera działają różne zegary uruchamiające pewne funkcje:

1. manageVideoFiles() - funkcja po stronie klienta, zarządzająca plikami video (m. in. usuwanie przedawnionych filmów). Uruchamiana jest w osobnym wątku w ustalonym interwałem czasowym - określa go stała MANAGE_VIDEO_FILES_INTERVAL - domyślnie jest to sekunda.
2. manageReports() - funkcja po stronie klienta, zarządzająca wysyłaniem raportów o odbieraniu filmów. Uruchamiana jest w osobnym wątku z ustalonym interwałem czasowym - określa go stała MANAGE_REPORTS_INTERVAL i domyślnie wynosi 10 sekund.

Uproszczony diagram komunikacji: Klient - Serwer



14. Obsługa sytuacji wyjątkowych:

- jeśli klient wykryje że utracił połączenie z serwerem - próbuje je wznowić aż do skutku. Kolejne próby następują z ustalonym interwałem czasowym określanym przez stałą, RECONNECT_INTERVAL - domyślnie jest to 10 sekund.
- jeśli serwer wykryje, że klient nie odpowiada - zwalnia jego zasoby,

15. Wnioski z testowania

1. Podczas testowania nie wykryliśmy krytycznych błędów w aplikacji. Można się więc spodziewać, że aplikacja jest w stanie prawidłowo realizować strumieniowanie filmów.
2. Zauważyliśmy spory problem z przesyłaniem filmów przez protokół UDP. Znacznie większa niż się spodziewaliśmy część pakietów

jest przesyłana nieprawidłowo co skutkuje dużą ilością ponowień wysłania tego samego filmu.

16. Instrukcja instalacji oraz proponowane środowisko produkcyjne

Podczas budowy oraz pisania aplikacji korzystaliśmy z Kontenerów linuksowych - konkretnie narzędzia Docker. Kontenery są sposobem wirtualizacji systemu operacyjnego - znacznie “lżejszym” niż pełna wirtualizacja. Stwierdziliśmy również, że system oparty na kontenerach może stanowić dobre środowisko produkcyjne.
Aby zoptymalizować szybkość pracy z kontenerami, przygotowaliśmy skrypty odpowiedzialne za ich uruchamianie:

- run-containers.sh - skrypt uruchamiający zbiór kontenerów o adresach IP podanych jako parametry uruchomienia skryptu,
- deploy.sh buduje aktualną wersję na podstawie kodu źródłowego i wysyła na wskazane w argumentach maszyny wirtualne

Po wykonaniu tych skryptów posiadamy już kontenery gotowe do uruchomienia na nich aplikacji klienckich.
Następnie wystarczy uruchomić skrypt run-server.sh oraz po zalogowaniu się na odpowiednie kontenery uruchomić na nich aplikację kliencką podając w argumentach nazwę hosta oraz port.

17. Podsumowanie

Z realizacji zadania wynieśliśmy sporo doświadczeń. Część z nas po raz pierwszy miała do czynienia z użytymi w aplikacji technologiami dzięki czemu mogliśmy się wiele nauczyć. Dodatkowo bezsprzecznym jest fakt, iż projekt jest trudny - mimo stosunkowo niedużej objętości kodu szacujemy, że zajął nam wspólnie około 200 godzin. Najcenniejsze wydaje się być zdobycie doświadczenia w programowaniu aplikacji sieciowych - nie każdy z nas miał z tym styczność do tej pory, a po ukończeniu projektu możemy śmiało stwierdzić, że posiadamy przynajmniej podstawową wiedzę o komunikacji sieciowej i sposobach jej realizacji w języku C++. Dodatkowo użycie kontenerów zaproponowanych przez jednego z uczestników projektu pokazało reszcie z nas jak sprawnie można testować i uruchamiać aplikacje wymagające działania kilku urządzeń jednocześnie.