

**AGH**

**Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie**

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,  
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

**KATEDRA METROLOGII I ELEKTRONIKI**

**Praca dyplomowa magisterska**

**Opracowanie oprogramowania do kompresji,  
przesyłu oraz składowania danych ze zdalnego  
systemu pomiaru parametrow ruchu drogowego**

**Development of software for compressing,  
transmitting and storing data from remote traffic  
measurement system**

Autor:

Tomasz Jonak

Kierunek studiów:

Automatyka i Robotyka

Opiekun pracy:

dr inż. Zbigniew Marszałek

Kraków, 2018

*Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykowania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.*

.....  
podpis

# **Spis treści**

<b>1. Wstęp teoretyczny .....</b>	5
1.1. Protokół TCP.....	5
1.2. Komunikacja za pomocą gniazda sieciowego z użyciem protokołu TCP .....	7
1.3. Kompresja danych .....	8
<b>2. Analiza problemu .....</b>	11
2.1. Struktura generowanych danych .....	12
2.2. Analiza ilościowa plików generowanych przez stanowisko pomiarowe.....	13
2.3. Obliczenie wartości średniego strumień danych .....	16
2.4. Oszacowanie przepustowość łącza sieciowego .....	16
2.5. Dobór metod kompresji .....	17
2.6. Wymagania projektowe .....	28
<b>3. Projekt systemu przesyłu .....</b>	31
3.1. Identyfikowanie nowych danych pomiarowych.....	32
3.2. Schemat działania systemu .....	33
3.3. Protokół przesyłu .....	37
3.4. Scenariusze awarii i sposoby ich obsługi .....	39
3.5. Wybór języka programowania.....	42
3.6. Obsługa danych strumieniowych .....	44
3.7. Implementacja komponentu Sender .....	45
3.8. Implementacja komponentu Receiver .....	50
<b>4. Testy działania systemu przesyłu .....</b>	53
4.1. Zachowanie systemu w warunkach symulacyjnych .....	53
4.2. Zachowanie systemu w warunkach roboczych .....	58
<b>5. Podsumowanie .....</b>	63
<b>Bibliografia.....</b>	64



# **Wstęp**

Systemy pomiarowe służą badaniu wielkości fizycznych za pomocą zestawu czujników. Wyniki pomiarów składowane lokalnie na stanowisku pomiarowym, zanim zostaną dostarczone do miejsca obróbki nie posiadają istotnej wartości. Dopiero dostarczenie ich do przestrzeni składowania posiadającej możliwość dystrybucji do kolejnych systemów, a następnie obróbka pozwalają na wyciągnięcie wniosków. Stanowiska pomiarowe w zależności od badanego zagadnienia mogą generować duże ilości danych, ich wymagana lokalizacja nie zawsze pozwala na doprowadzenie łącza sieciowego pozwalającego na bezproblemową transmisję do systemów przetwarzania.

Przykładem takiego systemu jest system pomiaru parametrów ruchu drogowego rejestrujący tzw. profile magnetyczne pojazdów zlokalizowany w miejscowości Gardawice. Dziennie generuje on duże ilości surowych, nieprzetworzonych danych. Dane te wykorzystywane są do celów badawczych na KMiE AGH. Stanowisko pomiarowe posiada połączenie sieciowe z uczelnią o niskiej przepustowości. Aby umożliwić przesył na bieżąco wymagane jest odpowiednie gospodarowanie łączem rozumiane jak wykorzystanie go w sposób oszczędny i dobrze przemyślany.

## **Cel pracy**

Niniejsza praca powstała w celu zaprojektowania oraz wdrożenia systemu służącego automatycznemu przesyłowi danych ze stanowiska pomiarowego do przestrzeni magazynowania w Katedrze Metrologii i Elektroniki wydziału EAIiIB AGH w warunkach ograniczonej przepustowości łącza.

## **Zakres pracy**

Praca obejmuje analizę danych generowanych przez stanowisko pomiarowe, oszacowanie strumienia generowanych danych oraz przepustowości łącza, dobór metod kompresji, opracowanie mechanizmu ochrony przed utratą danych oraz projekt i implementację oprogramowania realizującego pobór, kompresję, przesył oraz składowanie danych.

## Układ pracy

Rozdział 1 zawiera opisy technologii użytych w pracy takich jak kompresja, protokół TCP oraz gniazda sieciowe. Rozdział 2 opisuje funkcjonowanie stanowiska pomiarowego, analizę sprawności kompresji generowanych danych pomiarowych, oszacowanie przepustowości łącza oraz wymagania względem projektu oprogramowania. Rozdział 3 zawiera projekty komponentów składających się na system przesyłu, strategię działania w przypadku błędów środowiska oraz opis implementacji oprogramowania. Zachowanie implementacji w warunkach symulacyjnych oraz w środku docelowym opisuje rozdział 4. Na rozdział 5 składają się wnioski z poprzedzających rozdziałów, proponowane ścieżki rozwoju pracy oraz podsumowanie.

# **1. Wstęp teoretyczny**

Niniejszy rozdział zawiera opisy zagadnień oraz technologii użytych w pracy. Zagadnienia dotyczą warstwy sieciowej, nawiązywania połączeń sieciowych z użyciem protokołu TCP oraz kompresji.

## **1.1. Protokół TCP**

Protokół użyty w pracy jako kanał przesyłu danych . W modelach OSI oraz TCP/IP kategoryzowany jest jako protokół transportowy - pozwala na komunikację procesów na dwóch różnych maszynach, które komunikują się za pomocą sieci Internet. Protokół posiada następujące cechy:

- uporządkowany przesył danych (ang. ordered data transfer)
- retransmisja zgubionych pakietów (ang. packet retransmission)
- zgodność danych nadanych z odebranymi (ang. error-free transmission)
- kontrola przepływu do odbiorcy (ang. flow control)
- kontrola przepływu po sieci (ang. congestion control)

Uporządkowanie oraz brak błędów przesyłu są podstawowymi wymaganiem względem transportu plików, stanowią podstawę wykorzystania tego protokołu do realizacji projektu. Szeroki opis wszystkich cech dostępny jest w sekcji 1.5 dokumentu [1].

TCP jest protokołem stanowym, kanał komunikacyjny nazywany jest połączeniem (ang. connection). Jego nawiązanie oraz zamknięcie realizowane jest za pomocą odpowiednich sekwencji wiadomości: otwarcia oraz zamknięcia.

### **Sekwencja otwarcia połączenia**

Nawiązanie połączenia jest realizowane za pomocą przesłania serii pakietów posiadających odpowiednie flagi - SYN oraz ACK. W nawiązaniu połączenia wyróżniamy klienta - stronę nawiązującą połączenie oraz serwer - stronę nasłuchującą. Sekwencja ma następującą postać.

1. Klient wysyła pakiet z flagą SYN

2. Serwer odpowiada pakietem z flagami SYN oraz ACK
3. Klient wysyła pakiet z flagą ACK

Po odebraniu ostatniego pakietu sesję uważa się za poprawnie zainicjalizowaną i obie strony mogą przejść do przesyłu i odbioru danych. Opis szczegółowy w sekcji 3.4 dokumentu [1].

### **Retransmisje**

Nagłówek pakietu TCP posiada dwa 32 bitowe pola służące śledzeniu kolejności pakietów i potwierdzenia ich odbioru. Są to odpowiednio numer sekwencyjny (ang. sequence number) oraz numer potwierdzenia (ang. acknowledgement number). Na każdy wysłany pakiet o danym numerze sekwencyjnym N oczekiwany jest pakiet nadany przez serwer z flagą ACK oraz numerem sekwencyjnym o wartości N.

Potrzebę retransmisji można zidentyfikować na podstawie przekroczenia wartości oczekiwania (ang. timeout) na potwierdzenie danego pakietu oraz bazując na otrzymanej sekwencji potwierdzeń. Ilustruje to poniższy przykład.

W przypadku gdy pakiet zostanie zgubiony (o numerze sekwencyjnym N+1), zgodnie z protokołem odbiorca nie może wysłać pakietu ACK z wartością N+1 ani żadną następną aż do otrzymania pakietu, ponownie wysyła więc pakiet ACK dla ostatniego poprawnie odebranego pakietu (numer N). Otrzymanie trzykrotnie potwierdzenia dla pakietu N sygnalizuje nadawcy potrzebę retransmisji pakietu N+1. Precyzyjny opis działania numerów sekwencyjnych oraz ich relacji do zapewnienia uporządkowanego przesyłu opisuje sekcja 3.3 dokumentu [1].

### **Sekwencja zamknięcia**

Zakończenie połączenia może zostać zainicjowane przez obie strony komunikacji. Poprawne zamknięcie sesji wykorzystuje flagi FIN oraz ACK.

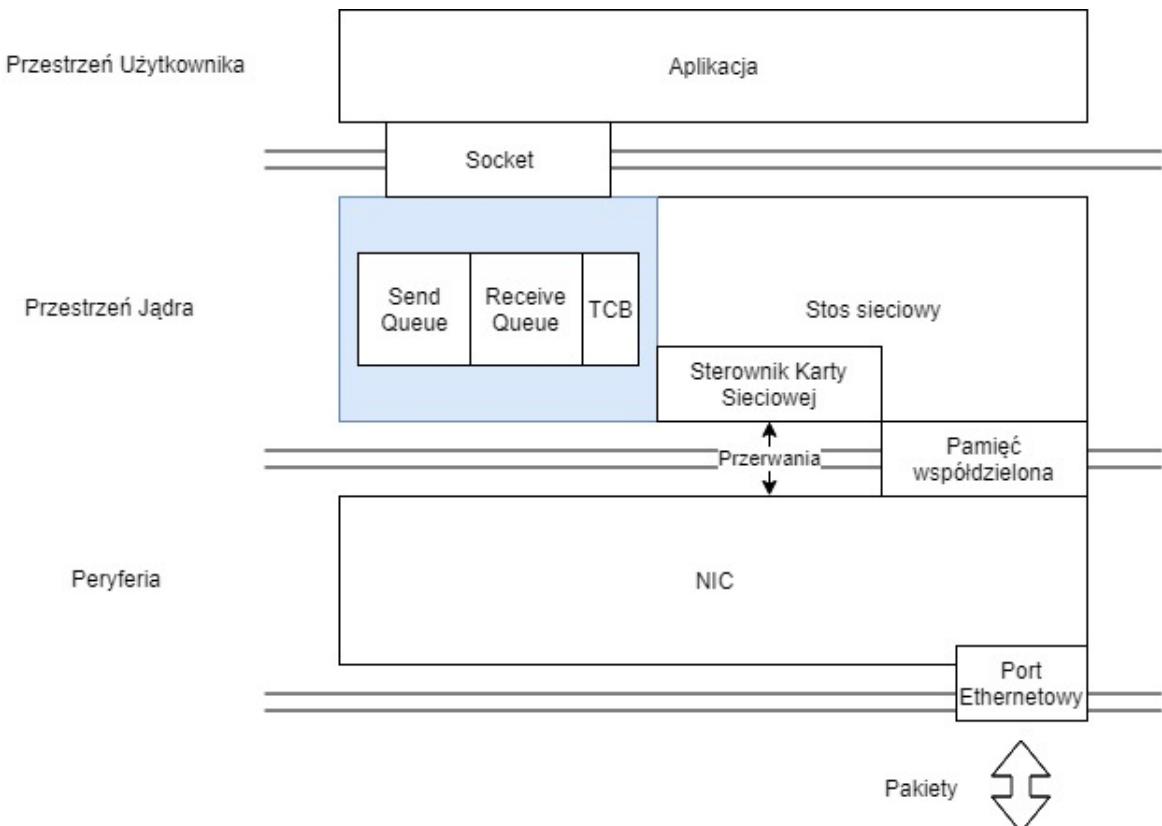
1. Inicjator wysyła pakiet z flagą FIN
2. Odbiorca wysyła pakiet z flagą ACK
3. Odbiorca wysyła pakiet z flagą FIN
4. Inicjator wysyła pakiet z flagą ACK

Kroki 2 oraz 3 mogą zostać zrealizowane za pomocą jednego pakietu wysłanego przez odbiorcę. Wysłanie pakietu FIN przez nadawcę sygnalizuje odbiorcy że nie otrzyma od niego więcej danych, sesję zamkniętą z jednej strony nazywamy half-open (pol. pół-otwartą) [1].

## 1.2. Komunikacja za pomocą gniazda sieciowego z użyciem protokołu TCP

W celu wyjaśnienia sposobu obsługi odczytu i zapisu danych z gniazd sieciowych, w niniejszym rozdziale przedstawiony zostanie opis konstrukcji gniazda. Wiedza ta stanowi powód zastosowania architektury opartej o wątki w rozdziale 3.

W komunikacji za pomocą gniazda wyróżnić można trzy elementy: aplikację użytkownika, jądro systemu operacyjnego oraz kartę sieciową (NIC, ang. Network Interface Card). Rysunek 1 przedstawia budowę gniazda sieciowego. Widoczne są na nim wymienione elementy oraz kanały komunikacyjne pomiędzy nimi. Aplikacja prowadzi interakcję z systemem operacyjnym za pomocą gniazda (ang. socket). Jądro systemu operacyjnego komunikuje się zaś z kartą sieciową za pomocą przerwań, oraz pamięci współdzielonej. Kolorem niebieskim oznaczono w przestrzeni jądra zestaw struktur danych, odpowiadających pojedynczemu połączeniu. Składa się on z kolejki nadawczej (ang. send queue), kolejki odbiorczej (ang. receive queue) oraz bloku kontrolnego połączenia (TCB, ang. TCP control block).



Rys. 1. Elementy odpowiedzialne za komunikację TCP z użyciem socketów

Użytkownik realizujący komunikację z pomocą gniazda zapisuje dane do kolejki nadawczej, bądź odczytuje z kolejki odbiorczej. W trakcie nadawania dane są pobierane z kolejki nadawczej, dzielone na pakiety i przekazywane do karty sieciowej, w celu nadania. W trakcie odbioru karta sieciowa przekazuje odebrane pakiety do jądra systemu, za pomocą pamięci

współdzielonej, gdzie są one szeregowane na podstawie numerów sekwencyjnych, a następnie dane zapisywane są w kolejce odbiorczej.

Wykonując odczyt użytkownik podaje ilość danych, którą oczekuje odebrać z kolejki. Jeśli kolejka zawiera mniej danych, niż żądana ilość, zwracana jest dostępna ilość. W przypadku, gdy wątek wykona odczyt, a kolejka odbioru będzie pusta, zostanie on uśpiony przez system operacyjny. Wątek zostanie wybudzony w momencie pojawięcia się danych w kolejce. Sytuację taką nazywamy wywołaniem blokującym. Analogicznie zachowuje się wywołanie send, gdy kolejka posiada mniej wolnego miejsca niż przekazana ilość, zapisane zostaje tyle ile pozostało miejsca. Podobnie w przypadku zapisu do pełnej kolejki, wątek zostaje uśpiony do czasu zwolnienia miejsca.

Wywołanie blokujące w aplikacji jednowątkowej prowadzi do jej zablokowania. Jest to efekt niepożądany jeśli ma ona reagować na wiele zdarzeń jednocześnie. W takim przypadku wymagane jest zastosowanie wielu wątków, oraz zapewnienie synchronizacji ich działania. Przykładem takiej sytuacji jest jednoczesne odbieranie informacji o zapisaniu na dysk nowych pomiarów i nadawanie danych.

## 1.3. Kompresja danych

W przypadku, gdy prędkość generowania danych przekracza przepustowość łącza sieciowego, może dojść do wyczerpania zasobów systemowych - pamięci ram oraz pojemności dysku twardego. Przypadek taki obserwowany jest w pracy. Aby zapobiec jego wystąpieniu wymagana jest obróbka danych, która zmniejszy ich wielkość. Czynność taka nazywana jest kompresją.

Kompresją nazywamy operację, która przekształca plik danych w nowy plik mniejszej wielkości, nazywany plikiem skompresowanym. Operacja odtworzenia pliku oryginalnego z pliku skompresowanego nazywana jest dekompresją.

Wyróżnia się dwie główne rodziny metod kompresji, stratne oraz bezstratne [2]. Rozróżnienie prowadzone jest w oparciu o wynik dekompresji. Jeśli reprezentacja skompresowana pozwala na odtworzenie oryginalnego pliku, metodę nazywamy bezstratną. Stosuje się je w przypadku danych, których integralność jest kluczowa, takich jak skompilowane programy komputerowe czy kod źródłowy. Przykładem takiej metody jest kodowanie Huffmana (ang. Huffman coding) [3].

Kompresja stratna odrzuca część informacji, w celu osiągnięcia większego stopnia kompresji. Stosuje się ją w przypadkach, w których niesiona informacja może zostać odtworzona pomimo zaburzeń, takich jak transport plików graficznych, wideo oraz ścieżek dźwiękowych. Przykładami tego typu kompresji są standardy JPEG [4] oraz H.264 [5].

Sprawność kompresji mierzona jest za pomocą stosunku wielkości reprezentacji wyjściowej do wejściowej. Wartość ta nazywana jest współczynnikiem kompresji [2]. Przedstawia ją wzór 1.

$$\eta = \frac{V_{out}}{V_{in}} * 100\% \quad (1)$$

Opisuje ona jakim procentem objętości oryginalnego pliku jest objętość pliku wynikowego.

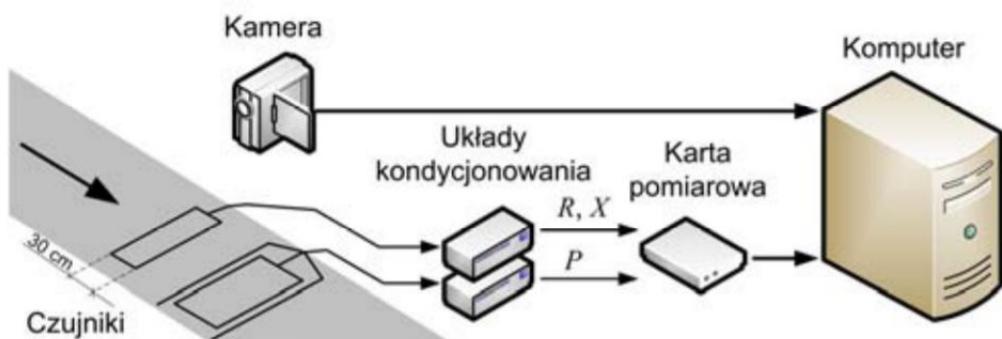


## 2. Analiza problemu

Celem rozdziału jest opis sposobu działania stanowiska pomiarowego, zbadanie struktury generowanych danych, oraz ich własności ilościowych, oszacowanie średniego strumienia danych generowanego przez pomiary oraz zmierzenie przepustowości sieci łączącej stanowisko pomiarowe z komputerem, do którego wysyłane będą dane pomiarowe.

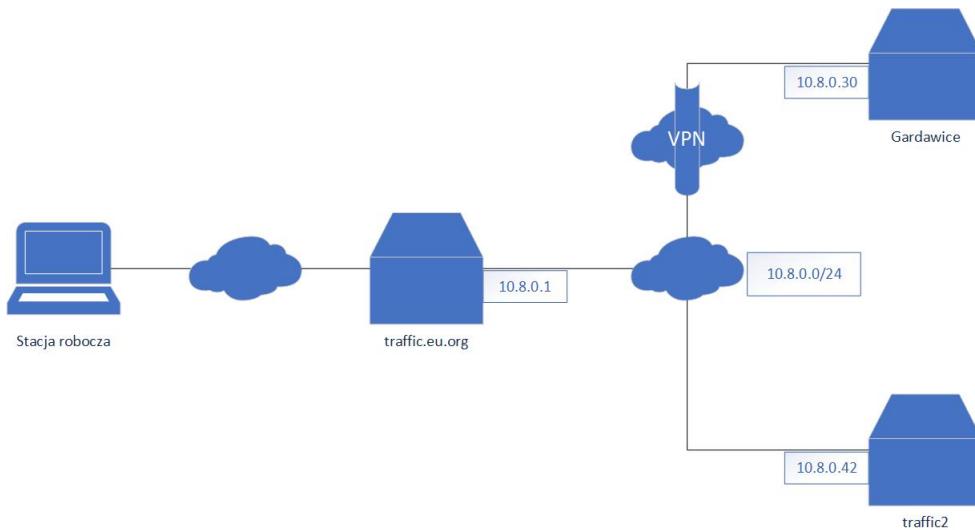
Katedra Metrologii i Elektroniki AGH prowadzi badania dotyczące odczytu informacji o pojazdach poruszających się po drodze za pomocą zestawu czujników pętlowych. Aparatura pomiarowa została zamontowana na odcinku drogi krajowej DK-81, w miejscowości Gardawice.

W skład stanowiska pomiarowego wchodzą czujniki, połączone z układem przetwarzania do postaci cyfrowej, kamera oraz komputer. Za obsługę stanowiska pomiarowego odpowiedzialne jest oprogramowanie napisane w środowisku LabVIEW. Odpowiada ono za zapis zmierzonych przebiegów sygnałów oraz sekwencji wideo, na dysku twardym. Rysunek 2 przedstawia schemat stanowiska.



Rys. 2. Schemat stanowiska pomiarowego, źródło: Z. Marszałek „[6]

Stanowisko pomiarowe posiada również połączenie sieciowe z Katedrą Merologii. Zestawiony został tunel VPN (ang. virtual private network) zapewniający bezpieczeństwo i szyfrowanie komunikacji. Schemat tego połączenia przedstawia rysunek 3. Miejscem docelowym składowania pomiarów jest maszyna traffic2. Na rysunku przedstawiono ponadto maszynę traffic.eu.org, do której łączy się użytkownik w celu uzyskania dostępu do pozostałych maszyn. Dostęp realizowany jest za pomocą protokołu SSH (ang. Secure SHell).



**Rys. 3.** Schemat połączeń sieciowych pomiędzy KMiE AGH a stanowiskiem pomiarowym

Interakcje ze środowiskiem w Gardawicach możliwe są za pomocą programu Cygwin, który pozwala używać maszyny z systemem operacyjnym Windows za pomocą konsoli w konwencji Unixowej.

## 2.1. Struktura generowanych danych

Rezultatem pomiaru jest para plików - przebiegi czasowe sygnałów w formacie lvm (ang. LabVIEW measurement) oraz sekwencję wideo w formacie AVI (ang. Audio Video Interleave). Pliki posiadają jednakową nazwę, generowaną na podstawie czasu wykonania pomiaru, oraz grupowane są w katalogach dobowych. Katalogi dobowe znajdują się w pojedynczym katalogu głównym. Tabela 1 przedstawia schemat stosowany do wygenerowania nazw plików.

**Tab. 1.** Schemat nazw plików

M	<b>Rok</b>	<b>Miesiąc</b>	<b>Dzień</b>	_	<b>Godzina</b>	<b>Minuta</b>	<b>Sekunda</b>
---	------------	----------------	--------------	---	----------------	---------------	----------------

Schemat nazwy folderu przedstawia tabela 2. Komórki pogrubione w obu tabelach stanowią zmienne zależne od momentu wygenerowania pliku.

**Tab. 2.** Schemat nazwy katalogu

R	<b>Rok</b>	_	<b>Miesiąc</b>	_	<b>Dzień</b>
---	------------	---	----------------	---	--------------

Zawartość katalogu głównego składowania danych pomiarowych zaprezentowana jest na rysunku 4. Oprócz katalogów z danymi, które generowane są automatycznie część plików spełniających schemat nazwy katalogu, posiada rozszerzenie .tar.bz2. Oznaczają one skompresowane katalogi z pomiarami dobowymi. Pliki te są śladem manualnej ingerencji, mającej na celu zaoszczędzenie miejsca na dysku twardym.

```
tomaszjonak@Dell-0755 /cygdrive/e/RejGARD/pomiary
$ ls
d_26_02_2018.sh          R2017_07_28.tar.bz2  R2017_08_21.tar.bz2  R2017_09_27.tar.bz2  R2017_11_04.tar.bz2  R2017_11_28.tar.bz2  R2018_03_10
dofile.sh                R2017_07_29.tar.bz2  R2017_08_22.tar.bz2  R2017_09_28.tar.bz2  R2017_11_05.tar.bz2  R2017_11_29.tar.bz2  R2018_03_11
dofile_1.11.log           R2017_07_30.tar.bz2  R2017_08_23.tar.bz2  R2017_09_29.tar.bz2  R2017_11_06.tar.bz2  R2017_11_30.tar.bz2  R2018_03_12
dofile_11.11.sh          R2017_07_31.tar.bz2  R2017_08_24.tar.bz2  R2017_09_30.tar.bz2  R2017_11_07.tar.bz2  R2017_12_01.tar.bz2  R2018_03_13
komprresa_tanjcf_rm.sh   R2017_08_01.tar.bz2  R2017_08_25.tar.bz2  R2017_10_01.tar.bz2  R2017_11_08.tar.bz2  R2017_12_02.tar.bz2  R2018_03_14
R2017_07_09.tar.bz2      R2017_08_02.tar.bz2  R2017_08_26.tar.bz2  R2017_10_02.tar.bz2  R2017_11_09.tar.bz2  R2017_12_03.tar.bz2  R2018_03_02
R2017_07_10.tar.bz2      R2017_08_03.tar.bz2  R2017_08_27.tar.bz2  R2017_10_05.tar.bz2  R2017_11_10.tar.bz2  R2017_12_04.tar.bz2  R2018_03_03
R2017_07_11.tar.bz2      R2017_08_04.tar.bz2  R2017_08_28.tar.bz2  R2017_10_06.tar.bz2  R2017_11_11.tar.bz2  R2017_12_05.tar.bz2  R2018_03_04
R2017_07_12.tar.bz2      R2017_08_05.tar.bz2  R2017_08_29.tar.bz2  R2017_10_07.tar.bz2  R2017_11_12.tar.bz2  R2017_12_06.tar.bz2  R2018_03_05
R2017_07_13.tar.bz2      R2017_08_06.tar.bz2  R2017_08_30.tar.bz2  R2017_10_08.tar.bz2  R2017_11_13.tar.bz2  R2017_12_07.tar.bz2  R2018_03_06
R2017_07_14.tar.bz2      R2017_08_07.tar.bz2  R2017_08_31.tar.bz2  R2017_10_21.tar.bz2  R2017_11_14.tar.bz2  R2017_12_08.tar.bz2  R2018_03_07
R2017_07_15.tar.bz2      R2017_08_08.tar.bz2  R2017_09_09.tar.bz2  R2017_10_22.tar.bz2  R2017_11_15.tar.bz2  R2017_12_09.tar.bz2  R2018_03_08
R2017_07_16.tar.bz2      R2017_08_09.tar.bz2  R2017_09_13.tar.bz2  R2017_10_23.tar.bz2  R2017_11_16.tar.bz2  R2017_12_10.tar.bz2  R2018_03_09
R2017_07_17.tar.bz2      R2017_08_10.tar.bz2  R2017_09_15.tar.bz2  R2017_10_24.tar.bz2  R2017_11_17.tar.bz2  R2018_02_15.tar.bz2  R2018_03_10
R2017_07_18.tar.bz2      R2017_08_11.tar.bz2  R2017_09_16.tar.bz2  R2017_10_25.tar.bz2  R2017_11_18.tar.bz2  R2018_02_28  R2018_03_11
R2017_07_19.tar.bz2      R2017_08_12.tar.bz2  R2017_09_18.tar.bz2  R2017_10_26.tar.bz2  R2017_11_19.tar.bz2  R2018_03_01  R2018_03_12
R2017_07_20.tar.bz2      R2017_08_13.tar.bz2  R2017_09_19.tar.bz2  R2017_10_27.tar.bz2  R2017_11_20.tar.bz2  R2018_03_02  R2018_03_13
R2017_07_21.tar.bz2      R2017_08_14.tar.bz2  R2017_09_20.tar.bz2  R2017_10_28.tar.bz2  R2017_11_21.tar.bz2  R2018_03_03
R2017_07_22.tar.bz2      R2017_08_15.tar.bz2  R2017_09_21.tar.bz2  R2017_10_29.tar.bz2  R2017_11_22.tar.bz2  R2018_03_04
R2017_07_23.tar.bz2      R2017_08_16.tar.bz2  R2017_09_22.tar.bz2  R2017_10_30.tar.bz2  R2017_11_23.tar.bz2  R2018_03_05
R2017_07_24.tar.bz2      R2017_08_17.tar.bz2  R2017_09_23.tar.bz2  R2017_10_31.tar.bz2  R2017_11_24.tar.bz2  R2018_03_06
R2017_07_25.tar.bz2      R2017_08_18.tar.bz2  R2017_09_24.tar.bz2  R2017_11_01.tar.bz2  R2017_11_25.tar.bz2  R2018_03_07
R2017_07_26.tar.bz2      R2017_08_19.tar.bz2  R2017_09_25.tar.bz2  R2017_11_02.tar.bz2  R2017_11_26.tar.bz2  R2018_03_08
R2017_07_27.tar.bz2      R2017_08_20.tar.bz2  R2017_09_26.tar.bz2  R2017_11_03.tar.bz2  R2017_11_27.tar.bz2  R2018_03_09
```

Rys. 4. Katalog główny danych pomiarowych

Strukturę katalogu dobowego z dnia 12.05.2018 przedstawia rysunek 5. Zawartość ograniczona została do pierwszych dziesięciu linii. Widoczne są pary plików będące rezultatami kolejnych pomiarów. W ciągu doby stanowisko pomiarowe wygenerowało 11376 plików co odpowiada 5688 pomiarom. Łączna wielkość danych z tego dnia wynosi 7.1 gigabajta.

```
tomaszjonak@Dell-0755 /cygdrive/e/RejGARD/pomiary/R2018_05_12
$ ls -lh | head -10
total 7.1G
-rwxrwx---+ 1 Administrators None 332K May 12 00:00 M180512_000010.avi
-rwxrwx---+ 1 Administrators None 818K May 12 00:00 M180512_000010.lvm
-rwxrwx---+ 1 Administrators None 325K May 12 00:00 M180512_000018.avi
-rwxrwx---+ 1 Administrators None 819K May 12 00:00 M180512_000018.lvm
-rwxrwx---+ 1 Administrators None 330K May 12 00:00 M180512_000036.avi
-rwxrwx---+ 1 Administrators None 818K May 12 00:00 M180512_000036.lvm
-rwxrwx---+ 1 Administrators None 324K May 12 00:01 M180512_000104.avi
-rwxrwx---+ 1 Administrators None 818K May 12 00:01 M180512_000104.lvm
-rwxrwx---+ 1 Administrators None 322K May 12 00:03 M180512_000328.avi

tomaszjonak@Dell-0755 /cygdrive/e/RejGARD/pomiary/R2018_05_12
$ ls -1 | wc -l
11376
```

Rys. 5. Katalog z pomiarami z jednego dnia

Rysunek 5 przedstawia zawartość katalogu z danymi pomiarowymi z dnia 12.05.2018 ograniczoną do pierwszych dziesięciu linii. Można zaobserwować pary plików odpowiadające kolejnym pomiarom. Ostatni widoczny pomiar został wykonany 3 minuty i 28 sekund po północy. W ciągu doby stanowisko pomiarowe wygenerowało 11376 plików o łącznej objętości 7.1 gigabajta.

## 2.2. Analiza ilościowa plików generowanych przez stanowisko pomiarowe

Celem analizy ilościowej jest zbadanie zmienności w danych generowanych przez stanowisko pomiarowe. Interesującymi cechami są średnie wielkości plików oraz ich zmienność

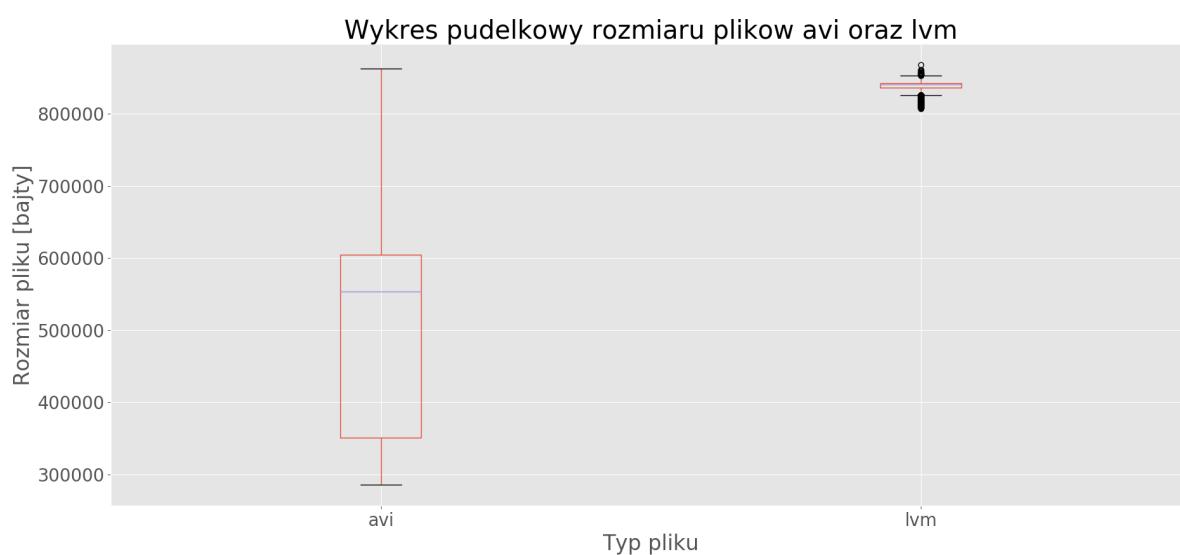
dobowa. Wnioski z tego rozdziału posłużą w dalszej części pracy do przygotowania testów działania aplikacji. Analizę prowadzono na podstawie zestawu danych wygenerowanych w dniach od 11.11.2017 do 10.12.2017.

Dziennie generowanych jest ok. 5.6 tysiąca par plików. Ilość pomiarów jest związana z natężeniem ruchu drogowego, stąd można zakładać okresowe wahania spowodowane m.in. cyklem dobowym oraz świętami. Właściwości statystyczne generowanych plików, z podziałem ze względu na typ, prezentuje tabela 3. Dane w tabeli, wyłączając liczbę plików, przedstawione są w bajtach. Wiersze Q1-Q3 odpowiadają kolejnym kwartylem.

**Tab. 3.** Właściwości statystyczne danych pomiarowych z badanego okresu

	avi	lvm
Liczba plików	167804	167804
Średnia	502728	838532
Odchylenie standardowe	133068	5378
Wartość minimalna	285696	806863
Q1 (25%)	351232	835712
Q2 (50%)	553984	840304
Q3 (75%)	604672	842676
Wartość maksymalna	862208	867866

Reprezentacją graficzną danych z tabeli 3 jest wykres pudełkowy zaprezentowany na rysunku 6. Pozioma niebieska linia oznacza medianę. Czerwony prostokąt ograniczony jest z dołu i góry przez odpowiednio pierwszy i trzeci kwartyl (Q1 oraz Q3). Dwie pozostałe poziome linie znajdują się w odległości danej wzorem  $1.5 \cdot IQR$ , gdzie  $IQR = Q3 - Q1$  od odpowiednich brzegów prostokąta. Czarne kropki oznaczają obserwacje odstające (ang. outliers) o wartościach nietypowych w kontekście badanego zbioru.

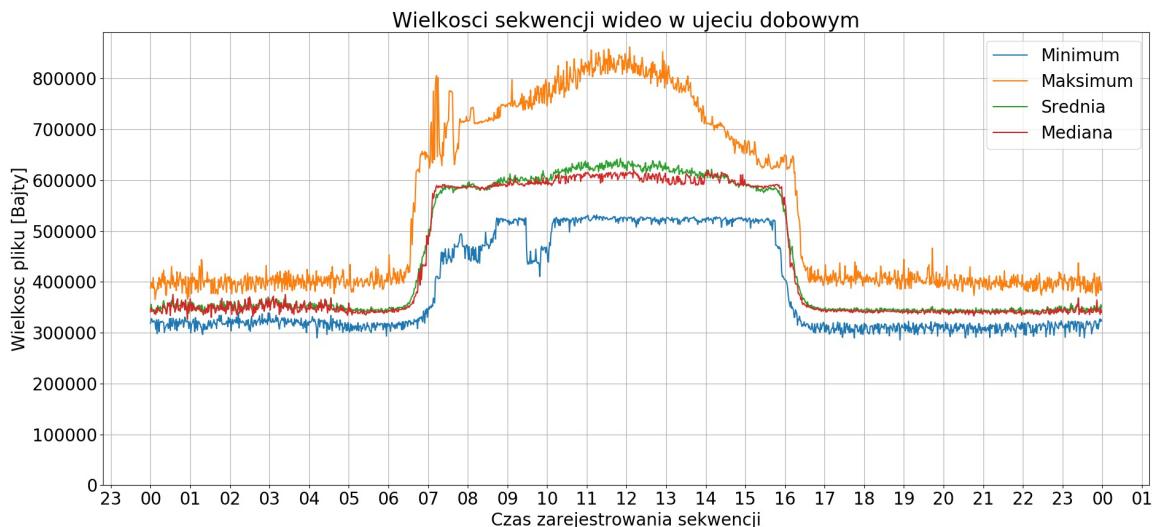


**Rys. 6.** Wykres pudełkowy rozmiarów plików z badanego okresu

Zarówno z rysunku 6, jak i tabeli 3 można odczytać wysoki rozrzut rozmiaru generowanych sekwencji wideo. Reprezentację graficzną danych w ujęciu dobowym przedstawiono

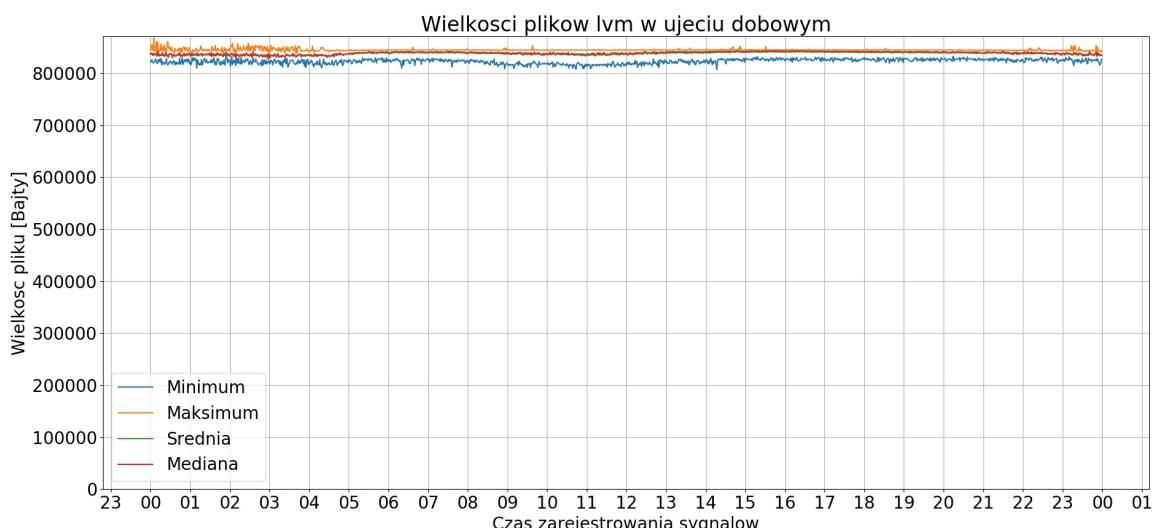
na rysunku 7. Wielkości plików zgrupowane zostały do minut a następnie dla każdej minuty przedstawiono minimum, maksimum, średnią oraz medianę. Na rysunku widoczne są dwie istotne zmiany rozmiaru, pierwsza między 6.30 a 7.30, druga między 15.30 a 16.30. Godziny te odpowiadają wschodowi i zachodowi słońca w listopadzie. Zmienność rozmiaru sekwencji wideo jest więc zależna od naświetlenia stanowiska pomiarowego.

Na wykresie można również zaobserwować pojawienie się pustego pliku po godzinie 17.



Rys. 7. Zależność rozmiaru sekwencji wideo od momentu zarejestrowania w ujęciu dobowym

Analogiczną metodę prezentacji zastosowano dla plików lvm. Przedstawia ją rysunek 8. Zmienność rozmiaru pliku nie wykazuje żadnej istotnej cykliczności w ujęciu dobowym.



Rys. 8. Zależność rozmiaru przebiegów czasowych sygnałów od czasu pomiaru

## 2.3. Obliczenie wartości średniego strumień danych

Średni strumień danych określa prędkość napływu danych pomiarowych do przestrzeni dyskowej. Jego wartość wyliczona została dzieląc sumę danych wygenerowanych w ciągu badanego okresu przez ilość sekund składających się na ten okres, formułę tą prezentuje wzór 2.

$$S = \frac{V}{T} \left[ \frac{B}{s} \right] \quad (2)$$

S - strumień, V - ilość danych w bajtach, T - czas

Obliczenia przeprowadzono dla 30 pełnych dni (11.11 - 10.12).

$$T = 30 * 24 * 60 * 60 = 2592000 s$$

$$V_{avi} = 84359710 kB$$

$$V_{lvm} = 140709087 kB$$

$$V = V_{avi} + V_{lvm} = 225068797 kB$$

$$S = \frac{225068797 kB}{2592000 s} \approx 86.8 \frac{kB}{s}$$

Wartość średnia w symulacji nie uwzględnia zmienności dobowej sekwencji wideo, zaobserwowanej w rozdziale 2.2.

## 2.4. Oszacowanie przepustowości łącza sieciowego

Oszacowanie przepustowości łącza sieciowego ma w pracy dwa cele. Obliczenie współczynnika kompresji oraz stworzenie środowiska symulacyjnego, o parametrach zbliżonych do rzeczywistych.

Przepustowość zbadano z użyciem narzędzia iPerf [7]. Pomiar został przeprowadzony pomiędzy maszynami traffic2 oraz gardawice, zgodnie z oznaczeniami na rysunku 3. Pomiar odbywa się w schemacie klient-serwer, w badanym przypadku rolę klienta przyjęła maszyna w miejscowości Gardawice.

Rezultat działania aplikacji iPerf widziany z komputera w miejscowości Gardawice przedstawia rysunek 9. Nawiązano połączenie z komputerem traffic2 nasłuchującym pod adresem IP 10.8.0.42 na porcie TCP 31337. Opcja -t definiuje czas pomiaru w sekundach, trwał on godzinę. Ostatnia linia na rysunku prezentuje podsumowanie pomiarów, zmierzona przepustowość wyniosła 58.1 kB/s.

```
tomaszjonak@Dell-0755 ~/iperf3/iperf-3.1.3-win64
$ iperf -f KBytes -c 10.8.0.42 -p 31337 -t 3600
-----
Client connecting to 10.8.0.42, TCP port 31337
TCP window size: 8.00 KByte (default)
-----
[  3] local 10.8.0.30 port 50365 connected with 10.8.0.42 port 31337
[ ID] Interval      Transfer     Bandwidth
[  3] 0.0-3600.8 sec  209152 KBytes  58.1 KBytes/sec
```

Rys. 9. Wyjście standardowe klienta programu iPerf w miejscowości Gardawice

Wynik pomiarów zaraportowany na maszynie traffic2 przedstawia rysunek 10. Argumenty rozruchowe aplikacji zawierają flagę -s odpowiadającą za uruchomienie aplikacji w trybie serwerowym. Flagi -B i -p sterują odpowiednio adresem IP i portem TCP, na których nasłuchuje aplikacja. Rysunek zawiera rezultaty trzech pomiarów, gdzie pomiar o identyfikatorze 4 odpowiada pomiarowi przedstawionemu na rysunku 9.

```
^Ctomaszjonak@traffic2 ~/iperf/usr/bin $ ./iperf -s -f KBits -B 10.8.0.42 -p 31337
-----
Server listening on TCP port 31337
Binding to local address 10.8.0.42
TCP window size: 85.3 KByte (default)
-----
[  4] local 10.8.0.42 port 31337 connected with 10.8.0.30 port 50363
[ ID] Interval      Transfer     Bandwidth
[  4] 0.0-30.7 sec   896 KBytes  29.2 KBytes/sec
[  5] local 10.8.0.42 port 31337 connected with 10.8.0.30 port 50364
[  5] 0.0-31.0 sec   384 KBytes  12.4 KBytes/sec
[  4] local 10.8.0.42 port 31337 connected with 10.8.0.30 port 50365
[  4] 0.0-3607.3 sec  209152 KBytes  58.0 KBytes/sec
```

Rys. 10. Wyjście standardowe serwera programu iPerf na maszynie traffic2

Wynikowa przepustowość zaraportowana przez serwer jest o 0.1 kB/s niższa niż w przypadku klienta. Może to być spowodowane buforowaniem danych przez stos sieciowy maszyny nadającej. W pracy przyjęto przepustowość łączka wynoszącą 58 kB/s.

## 2.5. Dobór metod kompresji

W sytuacji, gdy ilość danych generowanych przez stanowisko pomiarowe przekracza możliwości przesyłu łączka sieciowego może dojść do zapełnienia przestrzeni dyskowej. W badanym przypadku zgodnie z danymi z rozdziału 2.3 średni strumień generowanych danych wynosi 86.8 kilobajta na sekundę, a przepustowość łączka obliczona w rozdziale 2.4 wynosi 58 kilobajtów na sekundę. Wysyłanie danych w postaci nieprzetworzonej doprowadzi do zapełnienia przestrzeni dyskowej. Aby zapobiec temu scenariuszowi należy zmniejszyć wielkość strumienia wejściowego stosując kompresję.

W celu doboru metody kompresji należy wyznaczyć wartość graniczną współczynnika kompresji zdefiniowanego wzorem 1. Pozwoli ona odrzucić metody kompresji, które nie zapewniają wystarczającego zmniejszenia wielkości strumienia danych.

$$S_{in} = 86.8 \frac{kB}{s}$$
 - średni strumień generowany przez stanowisko pomiarowe

$$S_{out} = 58 \frac{kB}{s}$$
 - przepustowości sieci

$$\eta = \frac{V_{compressed}}{V_{original}} \cdot 100\% = \frac{58}{86.8} \cdot 100\% \approx 66.8\%$$

Analizę kompresji w toku niniejszego rozdziału rozbito ze względu na typ pliku na podrozdziały 2.5.1 oraz 2.5.2. Graniczna wartość współczynnika ma zastosowanie dla sumarycznego strumienia, stąd brana pod uwagę jest dopiero w rozdziale 2.5.3.

Do badań wybrano trzy popularne algorytmy kompresji, przedstawia je tabela 4. W badaniach użyto implementacji dostępnych jako biblioteki języka Python z ustawieniami domyślnymi, które przedstawia kolumna „Poziom Kompresji”.

**Tab. 4.** Badane algorytmy kompresji bezstratnej

Nazwa programu	Nazwa algorytmu	Poziom kompresji
bzip2	Transformata Burrowsa-Wheela [8]	9
lzma	Lempel-Ziv-Markov chain algorithm [9, 10]	6
zlib	deflate [11]	6

Zbiór danych testowych w niniejszym rozdziale stanowiły dane wygenerowane przez stanowisko pomiarowe dnia 06.10.2017.

### 2.5.1. Kompresja obrazu wideo

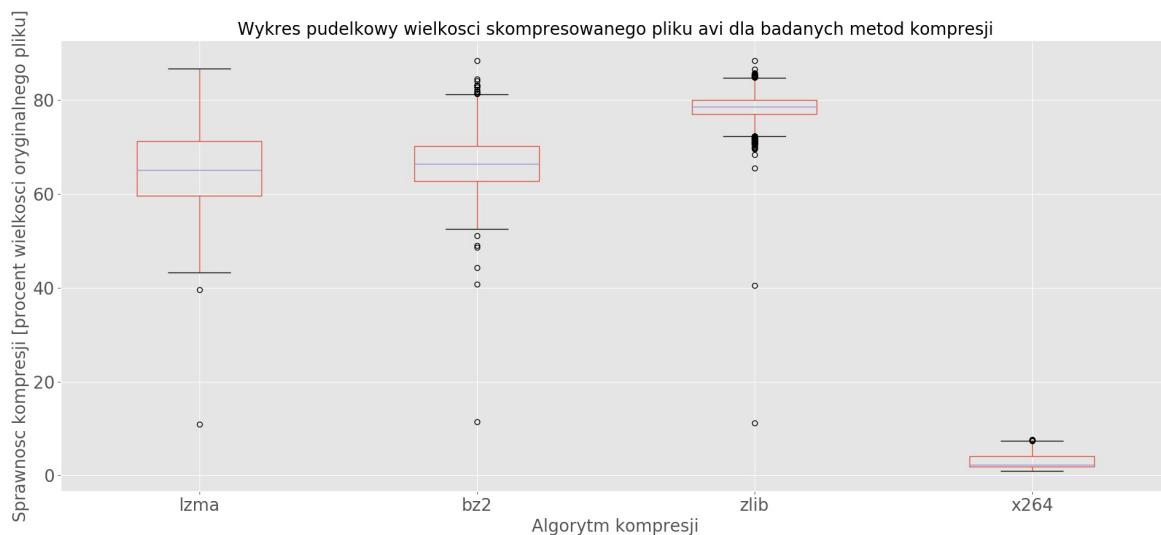
Obraz wideo uzyskany podczas przejazdu pojazdu służy do sprawdzenia typu pojazdu i ścieżki przejazdu przez stanowisko z czujnikami pętlowymi (istotny jest fakt zmiany pasa ruchu). Umożliwia to zastosowanie kompresji stratnej pod warunkiem zachowania możliwości wizualnej identyfikacji wymienionych cech pojazdu.

Badanie stopnia kompresji przeprowadzono na czterech algorytmach, trzy wymienionych w tabeli 4 oraz standard kompresji H.264.

Kompresję H.264 zrealizowaną za pomocą biblioteki x264 będącej częścią pakietu ffmpeg. Jest on zbiorem metod identyfikacji plików, algorytmów kompresji oraz transformacji pomiędzy formatami. Został wywołany przy użyciu następującej komendy.

```
ffmpeg -i <nazwa_pliku> -preset slow -crf 32 -c:v libx264 <plik_wyjsciowy>
```

Wyniki badania w formie wykresu pułapkowego przedstawia rysunek 11. Widać na nim wyraźną przewagę H.264 nad pozostałymi metodami.



Rys. 11. Wynik testów kompresji obrazu wideo w postaci wykresu pudełkowego

Tabela 5 stanowi uzupełnienie rysunku 11. Użycie biblioteki x264 oferuje średni współczynnik kompresji na poziomie 2.97% oraz najniższe odchylenie standardowe spośród badanych metod. Tym sposobem stanowi najlepszego kandydata do zastosowania w projekcie.

Tab. 5. Wynik badania kompresji obrazu wideo

	lzma	bzip2	zlib	x264
Ilość plików	6103	6103	6103	6103
Średnia	65.57	66.57	78.30	2.97
Odchylenie standardowe	6.83	5.28	2.93	1.39
Wartość minimalna	10.95	11.45	11.14	0.98
Q1 (25%)	59.61	62.81	76.98	1.88
Q2 (50%)	65.14	66.36	78.63	2.30
Q3 (75%)	71.28	70.18	80.09	4.10
Wartość maksymalna	86.72	88.37	88.40	7.64

Zgodnie z informacjami ze wstępu niniejszego rozdziału, wybór kompresji stratnej wymaga ewaluacji utraty informacji podczas kompresji. Pakiet ffmpeg oferuje dwa główne parametry sterujące działaniem metody.

- **crf** (ang. Constant Rate Factor) - określa stosunek jakości obrazu do stopnia kompresji. Przyjmuje wartości z przedziału 0-51, gdzie 0 oznacza kompresję bezstratną, a 51 najlepszą dostępną jakość. Wybór wartości tego parametru jest zależny od akceptowalnej jakości obrazu i wymaganego współczynnika kompresji. Wartość domyślna: 23.
- **preset** - opisuje stosunek prędkości kodowania do stopnia kompresji. Określany angielskimi słowami ('ultrafast', ..., 'medium', ..., 'veryslow'). Wartość domyślna 'medium'.

Parametr preset nie wpływa na jakość obrazu wynikowego, z tego powodu pod kątem utraty informacji badano parametr crf. Rysunek 12 przedstawia klatkę z oryginalnej sekwencji wideo. Rysunki 13 i 14 ilustrują wzrost zniekształceń w zależności od wartości parametru crf.

Przedstawione wartości parametru nie zaburzają możliwości identyfikacji rodzaju pojazdu, liczby osi czy pasa ruchu. Uznano za odpowiednie zastosowanie wartości 32 dla tego parametru.



Rys. 12. Oryginalna klatka z filmu



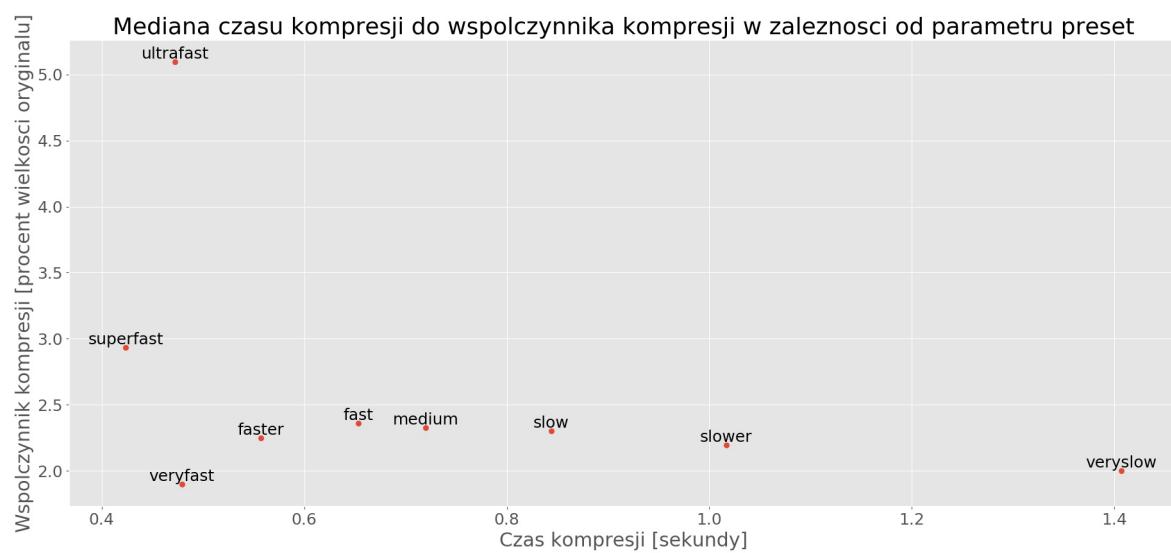
Rys. 13. Klatka skompresowana, crf 23



Rys. 14. Klatka skompresowana, crf 32

W celu doboru optymalnych z punktu widzenia pracy parametrów przeprowadzono analizę wpływu parametru preset na wielkość pliku wynikowego oraz czas kompresji. Analiza została przeprowadzona przy wartości parametru crf dobranej w poprzednim akapicie.

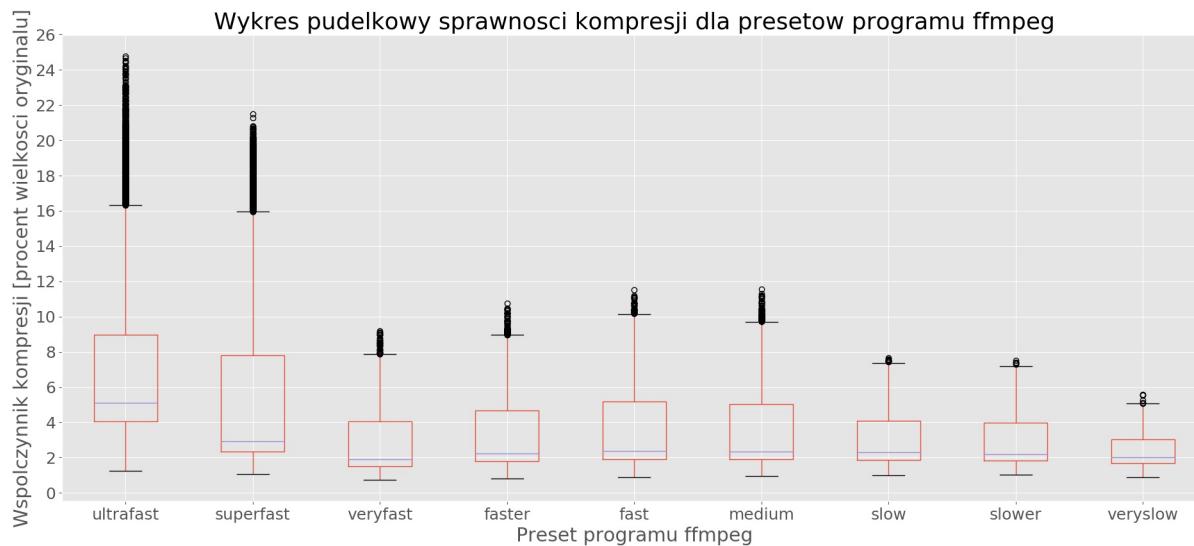
Rysunek 15 przedstawia medianę czasu kompresji oraz współczynnika kompresji dla każdego dostępnego ustawienia parametru preset. Najmniejszy czas kompresji obserwowany jest dla wartości 'superfast', najwyższy współczynnik kompresji - 'veryfast'.



Rys. 15. Wykres punktowy czasu do współczynnika kompresji w zależności od wartości parametru preset

W badanym przypadku zastosowano również metodę wizualizacji rozkładu danych za pomocą wykresów pudełkowych. Wykresy dla sprawności i czasu kompresji przedstawiają rysunki 16 i 17. Dane liczbowe zawierają odpowiednio tabele 6 i 7.

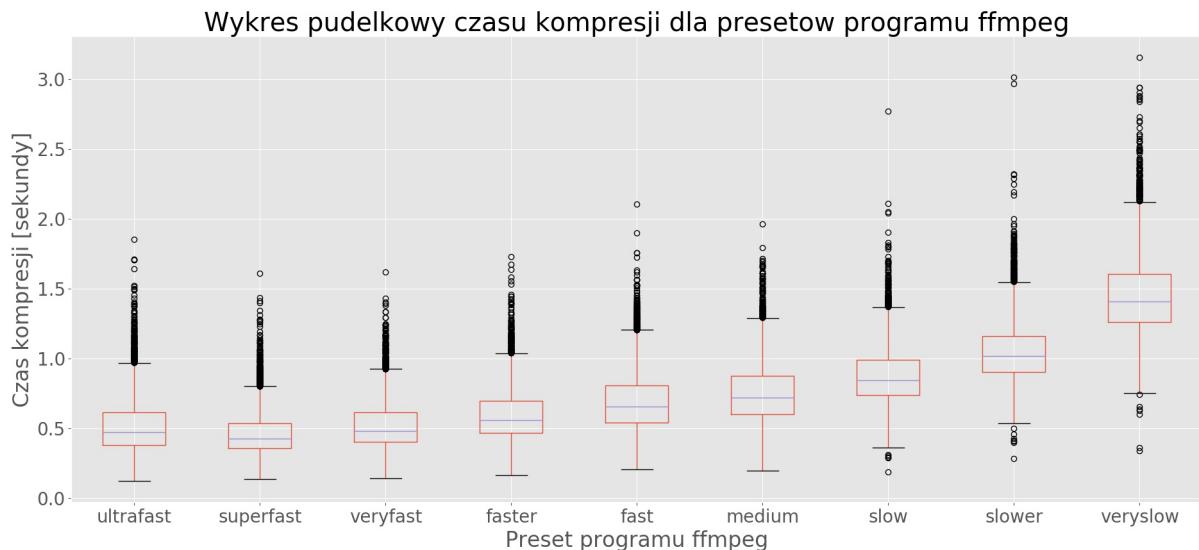
Odrzucono wartości 'ultrafast' oraz 'superfast', pomimo najniższego czasu kompresji ich sprawność zdecydowanie odbiega od pozostałych oraz cechuje się niską stabilnością. Ustawienia 'faster', 'fast' oraz 'medium' prezentują mniejszą sprawność niż 'veryfast' przy wyższym oczekiwany czasie kompresji.



Rys. 16. Wykres pudełkowy sprawności kompresji w zależności od wartości parametru preset

**Tab. 6.** Wartości liczbowe pomiarów skuteczności kompresji w zależności od wartości parametru preset

	ultrafast	superfast	veryfast	faster	fast	medium	slow	slower	veryslow
Ilość plików	6103	6103	6103	6103	6103	6103	6103	6103	6103
Średnia	7.89	6.45	2.88	3.38	3.81	3.79	2.97	2.89	2.33
Odchylenie standardowe	5.66	6.19	1.85	2.17	2.64	2.66	1.39	1.40	0.82
Wartość minimalna	1.24	1.06	0.73	0.81	0.89	0.96	0.98	1.02	0.88
Q1 (25%)	4.05	2.35	1.52	1.79	1.89	1.90	1.88	1.81	1.67
Q2 (50%)	5.09	2.94	1.90	2.25	2.36	2.33	2.30	2.20	2.00
Q3 (75%)	8.96	7.79	4.06	4.66	5.20	5.03	4.10	3.99	3.03
Wartość maksymalna	24.79	21.49	9.19	10.74	11.54	11.55	7.64	7.52	5.59



Rys. 17. Wykres pudełkowy czasu kompresji w zależności od wartości parametru preset

Tab. 7. Wartości liczbowe pomiarów czasu kompresji w zależności od wartości parametru preset

	ultrafast	superfast	veryfast	faster	fast	medium	slow	slower	veryslow
Ilość plików	6103	6103	6103	6103	6103	6103	6103	6103	6103
Średnia	0.52	0.46	0.52	0.60	0.69	0.76	0.88	1.05	1.45
Odchylenie standardowe	0.20	0.16	0.17	0.18	0.21	0.21	0.20	0.21	0.28
Wartość minimalna	0.12	0.14	0.14	0.16	0.21	0.20	0.19	0.29	0.34
Q1 (25%)	0.38	0.36	0.40	0.47	0.54	0.60	0.74	0.90	1.26
Q2 (50%)	0.47	0.42	0.48	0.56	0.65	0.72	0.84	1.02	1.41
Q3 (75%)	0.61	0.54	0.61	0.70	0.81	0.88	0.99	1.16	1.60
Wartość maksymalna	1.85	1.61	1.62	1.73	2.10	1.96	2.77	3.01	3.15

Na podstawie analizy danych przedstawionych na rysunkach 16 i 17 wybrano wartość 'veryslow'. Zysk z krótszego czasu kompresji jest tracony w momencie gdy dane skompresowane oczekują na wysyłkę w buforze nadawczym połączenia. Zmniejszenie wagi kosztem czasu kompresji odciąża łącze sieciowe.

## 2.5.2. Kompresja plików lvm

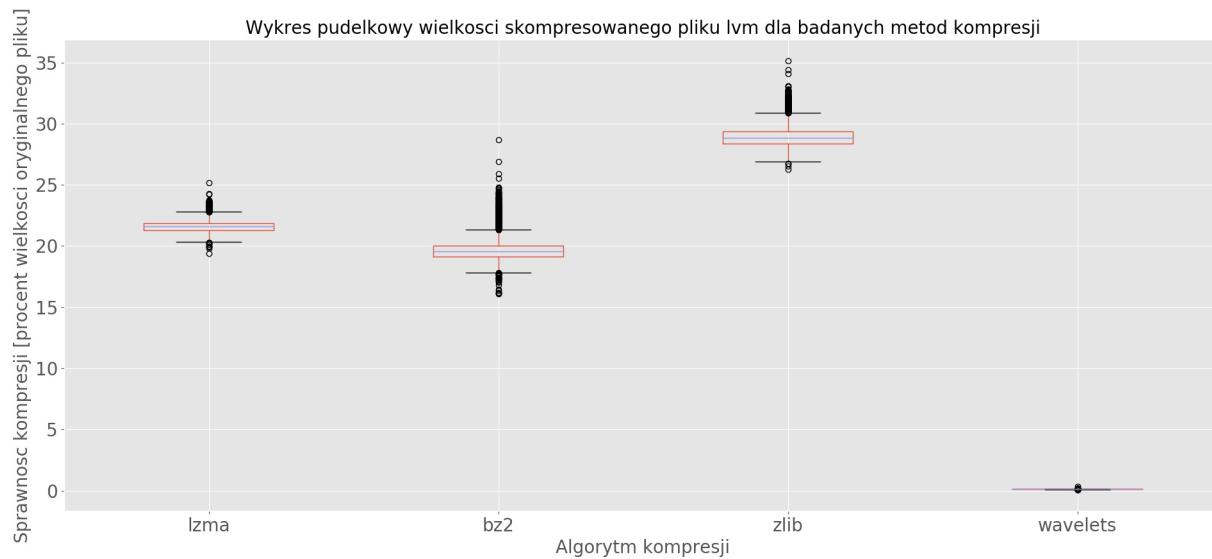
Profile magnetyczne składowane są w postaci plików lvm (ang. LabVIEW Measurement). Zawartość przykładowego pliku przedstawia rysunek 18. Dane podzielone są na siedem kolumn. Pierwsza z nich zawiera czas wykonania próbki z rozdzielcością 0.0001 sekundy. Poostałe zawierają przebiegi czasowe sygnałów, tzw. profile magnetyczne.

```
woyski@shipwreck-wind:/mnt/d/linux_workspace/mgr/implementacja/client_storage/R2017_10_06$ head M171005_235959.lvm
0.000000 -0.080204 0.200847 -0.070002 0.228491 -0.076913 0.092902
0.001000 -0.070660 0.207100 -0.062433 0.228491 -0.102583 0.083358
0.002000 -0.069673 0.214011 -0.051902 0.240668 -0.100279 0.082700
0.003000 -0.082179 0.214998 -0.041370 0.256136 -0.103241 0.071511
0.004000 -0.083166 0.225529 -0.031827 0.265350 -0.100279 0.060980
0.005000 -0.078559 0.234744 -0.036434 0.260414 -0.098634 0.049132
0.006000 -0.074939 0.232111 -0.038738 0.259427 -0.103570 0.047816
0.007000 -0.075926 0.222567 -0.039725 0.244617 -0.126278 0.048575
0.008000 -0.067040 0.226188 -0.025574 0.241326 -0.126667 0.046499
0.009000 -0.065066 0.225200 -0.033472 0.221580 -0.116405 0.035639
```

**Rys. 18.** Przykładowa zawartość pliku lvm

Warto zauważyć, że czas wykonania próbki jest równy indeksowi wiersza pomnożonemu przez 0.0001, stąd istnieje możliwość usunięcia pierwszej kolumny przed wysłaniem i odzworzenia po odebraniu pliku w pracy nie zastosowano jednak tej metody.

Zbadano ponownie zestaw algorytmów wymieniony w tabeli 4. Do testów dołączono algorytm kompresji sygnałowej oparty o zastosowanie falek, którego prototyp powstał w Katedrze Metrologii wydziału EAIiIB AGH. Ze względu na brak oficjalnej nazwy algorytm w toku pracy nazywany jest algorytmem falkowym, na rysunkach użyte zostało angielskie słowo wavelets (pol. falki).

**Rys. 19.** Wyniki testów kompresji plików lvm w formie wykresu pudełkowego**Tab. 8.** Wynik testów kompresji plików lvm w ujęciu statystycznym

	lzma	bzip2	zlib	wavelets
Ilość plików	6103	6103	6103	6103
Średnia	65.57	66.57	78.30	2.97
Odcchylenie standardowe	6.83	5.28	2.93	1.39
Wartość minimalna	10.95	11.45	11.14	0.98
Q1 (25%)	59.61	62.81	76.98	1.88
Q2 (50%)	65.14	66.36	78.63	2.30
Q3 (75%)	71.28	70.18	80.09	4.10
Wartość maksymalna	86.72	88.37	88.40	7.64

Rysunek 19 oraz tabela 8 przedstawiają wyniki badań kompresji plików lvm. Kompresja falkowa wybija się na tle pozostałych metod. Średni współczynnik kompresji wyniósł 2.97%, co jest wartością rzadkiem wielkości niższą niż w przypadku pozostałych metod.

Kompresja stratna wymaga analizy utraty informacji. Za miarę utraty informacji przyjęto błąd średniokwadratowy dany wzorem 3.

$$mse = \frac{1}{n} \sum_{i=0}^{n-1} (x_i - \dot{x}_i)^2 \quad (3)$$

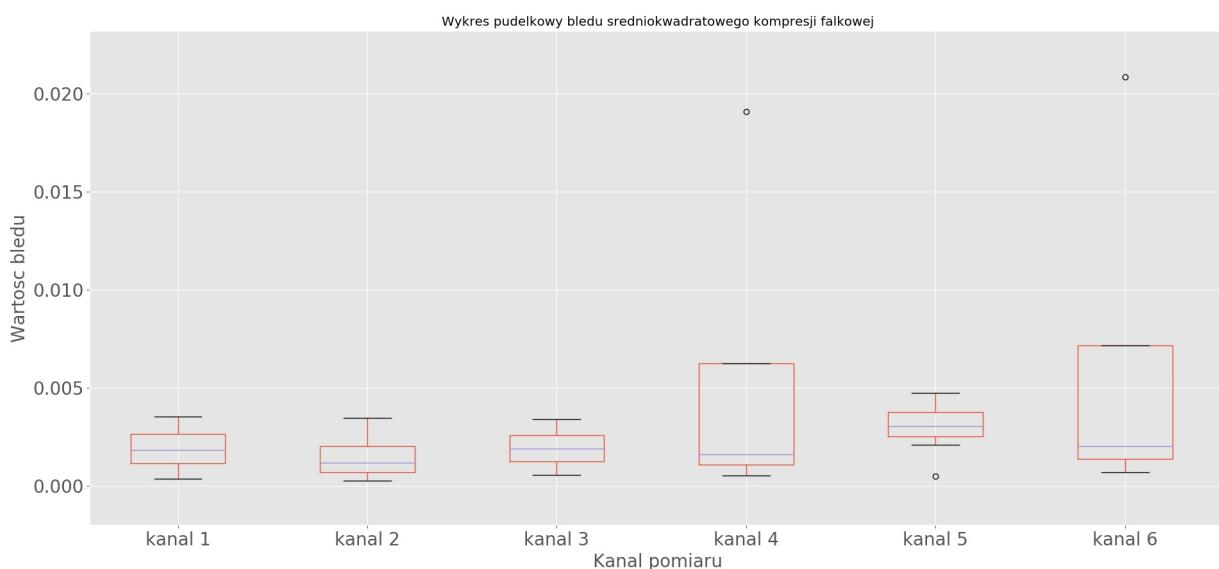
$n$  - liczba próbek

$x_i$  - i-ta próbka oryginalnego sygnału

$\dot{x}_i$  - i-ta próbka zdekompresowanego sygnału

Sygnal zdekompresowany jest wynikiem zastosowania sekwencji kompresja-dekompresja na sygnale oryginalnym. Wyniki eksperymentu przedstawiają rysunek 20 oraz tabela 9. W analizie wyróżniono kanały pomiarowe ze względu na inny oczekiwany przebieg sygnału.

Dane pokazują wartości w zakresie od jednej tysięcznej do jednej setnej. Na rysunku widać, że są również pomiary odstające, których wartość błędu nie przekracza jednej trzydziestej.

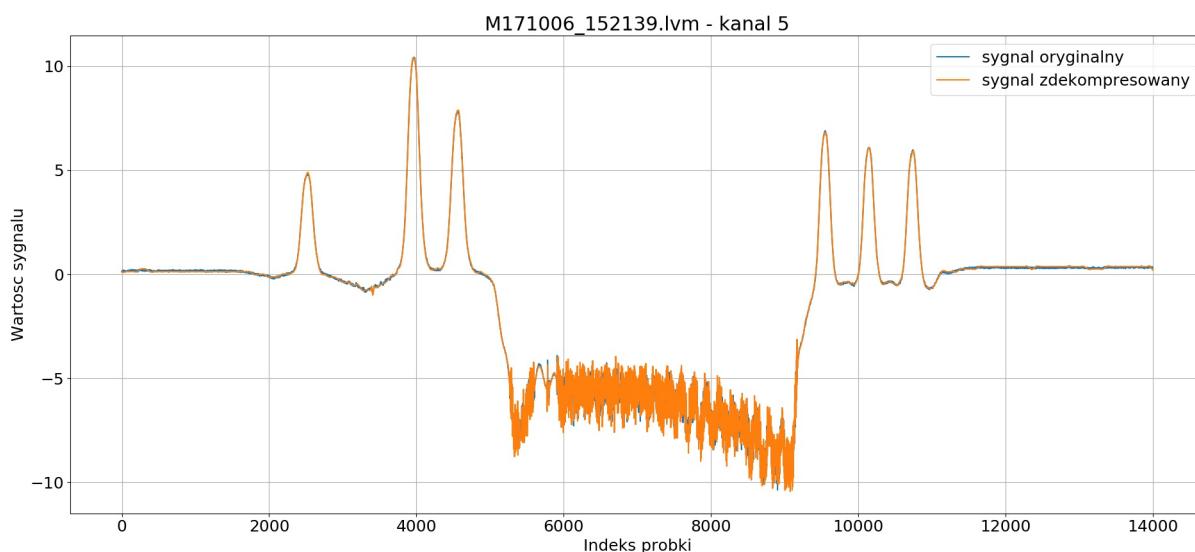


Rys. 20. Wykres pudełkowy błędu średnio-kwadratowego kompresji algorytmem falkowym

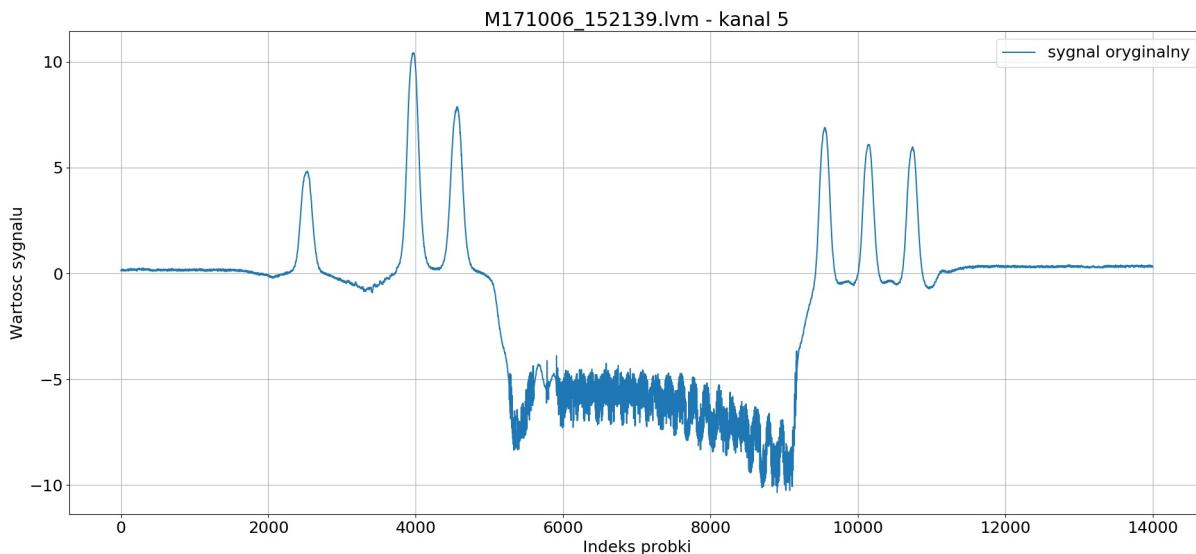
	Kanał 1	Kanał 2	Kanał 3	Kanał 4	Kanał 5	Kanał 6
Ilość plików	6104	6104	6104	6104	6104	6104
Średnia	0.001829	0.001238	0.001856	0.001611	0.003053	0.002091
Odcchylenie standardowe	0.000687	0.000658	0.000535	0.000539	0.000500	0.000685
Wartość minimalna	0.000341	0.000247	0.000536	0.000503	0.002096	0.000757
Q1 (25%)	0.001288	0.000699	0.001472	0.001242	0.002637	0.001560
Q2 (50%)	0.001840	0.001115	0.001944	0.001556	0.002987	0.001966
Q3 (75%)	0.002374	0.001549	0.002287	0.001957	0.003436	0.002591
Wartość maksymalna	0.003515	0.003460	0.003403	0.019099	0.004734	0.020863

**Tab. 9.** Dane statystyczne błędu średnio-kwadratowego kompresji algorytmem falkowym

Przebieg o najwyższym błędzie zaobserwowanym w zbiorze testowym przedstawia rysunek 21. Sygnał oryginalny przedstawia rysunek 22. Metoda falkowa nawet w przypadku pesymistycznym zachowuje kluczowe cechy sygnału.



**Rys. 21.** Sygnał oryginalny i po procesie kompresji i dekompresji o najwyższym błędzie średnio-kwadratowym



Rys. 22. Sygnał oryginalny, którego sekwencja kompresja-dekompresja ma najwyższy błąd średniankwiadratowy

Algorytm falkowy został zbadany z ustawieniami domyślnymi, zaczerpniętymi z implementacji w języku Matlab dostarczonej jako referencyjna. Modyfikacje parametrów algorytmu znaczco wpływają zarówno na wartość błędu jak i stopień kompresji sygnału. Badania dotyczące wpływu wartości parametrów algorytmu na stopień kompresji i wartość błędu wykraczają jednak poza zakres niniejszej pracy.

Implementacja tego algorytmu w postaci biblioteki języka Python została dołączona do źródeł pracy. Może ona posłużyć jako podstawa do dalszych badań nad algorytmem.

### 2.5.3. Wynikowy schemat kompresji

Analiza kompresji poszczególnych typów plików, przeprowadzona w rozdziałach 2.5.1 i 2.5.2, nie daje pełnej informacji o kompresji. Stanowisko pomiarowe generuje parę plików, stąd aby wyliczyć współczynnik kompresji należy skompresować osobno każdy z typów plików a następnie porównać sumę wielkości plików wejściowych z sumaryczną wielkością plików skompresowanych. Zależność tą opisuje wzór 4.

$$\eta_{[metoda1, metoda2]} = \frac{\dot{V}_{avi}^{metoda1} + \dot{V}_{lvm}^{metoda2}}{V_{original}} \cdot 100\% \quad (4)$$

Wielkość oryginalna oznacza sumę wielkości wszystkich plików ze zbioru pomiarowego.

$$V_{original} = V_{avi} + V_{lvm} = 10055751.505 \text{ Kb}$$

Wielkość skompresowana oznacza sumę wielkości wszystkich plików danego typu skompresowanych daną metodą

$$\dot{V}_{typ}^{metoda} = \sum_{plik \in typ} \text{wielkość}(metoda(plik))$$

Tabela 10 przedstawia współczynniki kompresji obliczone dla poszczególnych par metod kompresji za pomocą wzoru 4. Dane przedstawiono w formie macierzowej, gdzie indeks wierszowy oznacza metodę kompresji pliku lvm, a indeks kolumnowy pliku avi (sekwencji

wideo). Najlepszą sprawność oferuje para metod (wavelet, x264), ich zastosowanie redukuje rozmiar danych do 1.74% wielkości wejściowej.

**Tab. 10.** Macierz sumarycznego współczynnika kompresji

		avi			
		lzma	bz2	zlib	x264
lvm	lzma	40.32	41.08	46.56	13.86
	bzip2	39.26	40.02	45.49	12.80
	zlib	44.47	45.22	50.70	18.00
	wavelet	28.21	28.97	34.44	1.74

Warto nadmienić, że wszystkie proponowane pary kompresorów oferują współczynnik kompresji niższy, niż wielkość graniczna. Wybranie schematu o najkorzystniejszej wartości, ubezpiecza aplikację na wypadek nagłego wzrostu prędkości generowania pomiarów, bądź spadku przepustowości łącza.

## 2.6. Wymagania projektowe

Analiza danych rejestrowanych przez stanowisko pomiarowe prowadzi do następujących wniosków.

- Ilość danych, generowanych przez stanowisko pomiarowe, nie jest stała w czasie
- Częstotliwość pojawiania się nowych plików jest zmienna
- Dane składowane są w podkatalogach konkretnego katalogu na dysku
- Przepustowość łącza nie pozwala na bezpośredni przesył danych, wymagana jest kompresja
- Wielkość sekwencji wideo jest zmienna w ciągu doby
- Nazwy plików posiadają regularną strukturę, pozwala to na ich porządkowanie w czasie bez użycia metadanych pliku

Wymagania projektowe:

- System ma automatycznie nawiązywać ponowne połączenie w przypadku problemów z siecią. Przewidywana jest niska stabilność łącza sieciowego.
- W przypadku utraty połączenia sieciowego aplikacja ma być w stanie przechować informacje o generowanych plikach i ponowić wysyłanie po odzyskaniu połączenia
- Parametry aplikacji muszą mieć formę przełączników rozruchowych bądź plików konfiguracyjnych.

Wymagane parametry konfigurowalne:

- Adres klienta
- Adres serwera
- Ścieżka do folderu, w którym składowane są wygenerowane dane pomiarowe
- Ścieżka do folderu, w którym składowane będą przesłane dane pomiarowe
- Schematy kompresji wybrane dla danych typów plików
- Lista typów plików do przesyłania



### **3. Projekt systemu przesyłu**

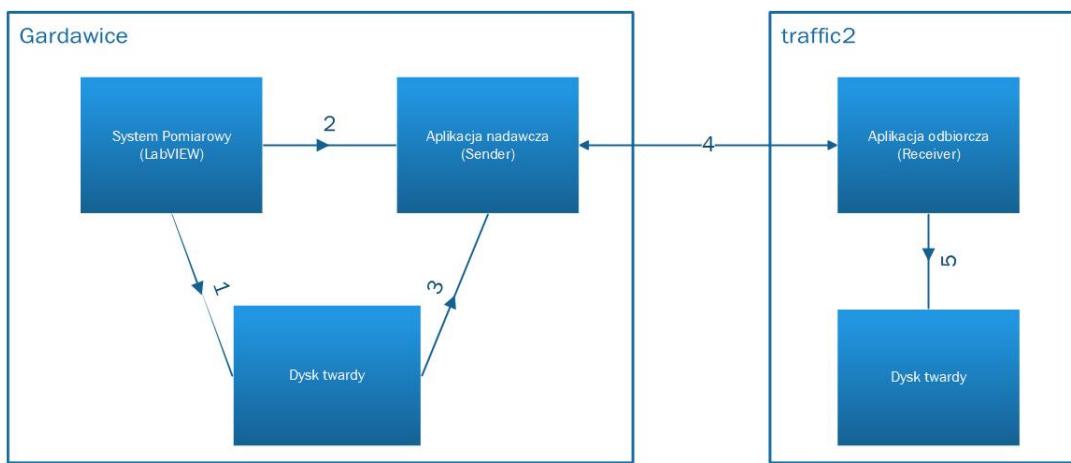
Niniejszy rozdział przedstawia projekt systemu służącemu przesyłowi danych, strategie ochrony przed ich utratą oraz informacje o sposobie konfiguracji i uruchamianiu gotowego projektu. Podstawowymi zadaniami systemu są:

1. Pobranie danych ze stanowiska pomiarowego
2. Bezstratny przesył sieciowy
3. Zapisanie danych na serwerze

Projekt musi również przestrzegać zestawu ograniczeń:

- Aplikacja zapisująca dane z kart pomiarowych do dysku nie może być modyfikowana
- Przepustowość łącza jest mniejsza niż strumień danych generowanych przez stanowisko pomiarowe
- Połączenie pomiędzy stanowiskiem pomiarowym a uczelnią może zostać zerwane
- Należy unikać instalacji dodatkowych programów/bibliotek na maszynie stanowiska pomiarowego. Motywowane jest to problemami z połączeniem i przepustowością

Najważniejsze elementy projektu zostały przedstawione na rysunku 23. Pogrupowano je ze względu na maszyny, na których się znajdują. Liniami oznaczone zostały kanały służące komunikacji, bądź przesyłowi danych. W pracy zaprojektowano komponenty Sender i Receiver, oraz kanały komunikacyjne 2 i 4. Linie 3 i 5 oznaczają interakcje z dyskiem i stanowią część komunikacji kanałów odpowiednio 2 i 4.



Rys. 23. Schemat rozwiązania

Komponent LabVIEW oznacza system generowania danych pomiarowych i jest zewnętrznym aktorem z perspektywy projektu.

### 3.1. Identyfikowanie nowych danych pomiarowych

Komponent Sender, w celu rozpoczęcia procedury wysyłania, potrzebuje informacji o zapisaniu na dysku twardym nowego pakietu danych pomiarowych. Uzyskanie tej informacji można uzyskać na szereg sposobów:

1. Komponent Sender monitoruje stan dysku samodzielnie identyfikując nowe pliki
2. Komponent Sender subskrybuje zdarzenia systemowe mówiące o pojawienniu się nowych plików
3. Komponent LabVIEW za pomocą wybranego kanału udostępnia informację o pojawienniu się nowego pliku

Propozycja pierwsza zakłada cykliczne sprawdzanie zawartości folderu bazowego danych pomiarowych. Porównywanie z poprzednim odczytem, a następnie identyfikowanie nowych plików na podstawie różnicy tych stanów. Model taki może wprowadzić opóźnienie, nowe pliki mogą pojawić się tuż po sprawdzeniu zawartości folderu z pomiarami. W pesymistycznym przypadku jest ono równe interwałowi pomiędzy kolejnymi sprawdzieniami. Ponadto czynność ta wymaga dużego obciążenia procesora.

Systemy operacyjne dostarczają mechanizmów automatycznego raportowania tego typu zdarzeń. Rodzina systemów Linux oferuje mechanizm inotify [12], Windows zaś wywołanie ReadDirectoryChangesW [13]. Wadą tego podejścia jest brak przenośności. Wymusza ono również na użytkowniku zastosowanie konkretnych języków programowania, w których udostępnione zostały biblioteki wymienionych mechanizmów, bądź bibliotek typu FFI (ang. foreign function interface). Biblioteki takie pozwalają na łączenie kodu napisanego w różnych językach. Wadą ich zastosowania jest zwiększenie stopnia skomplikowania kodu aplikacji.

Podejście piąte wymaga utworzenia dodatkowego serwisu raportującego w komponencie LabVIEW. Zmiana taka w założeniu nie wpływa na cykl działania aplikacji, wymaga jednak modyfikacji istniejącego rozwiązania. Została ona zaakceptowana i wprowadzona.

System pomiarowy nasłuchuje na porcie 6341 protokołu TCP, w przypadku nawiązania połączenia wysyła on za jego pomocą ścieżki do plików rozdzielone znakiem nowej linii po ich zapisaniu do dysku. Sekwencje raportów zapisania danych pomiarowych przedstawia rysunek 24. Połączenie zainicjowano za pomocą programu netcat.

```
tomaszjonak@Dell-0755 /cygdrive/e/RejGARD
$ nc 127.0.0.1 6341
R2018_05_15/M180515_215901
R2018_05_15/M180515_215930
R2018_05_15/M180515_215937
R2018_05_15/M180515_220003
```

Rys. 24. Ścieżki nadawane przez serwis LabVIEW

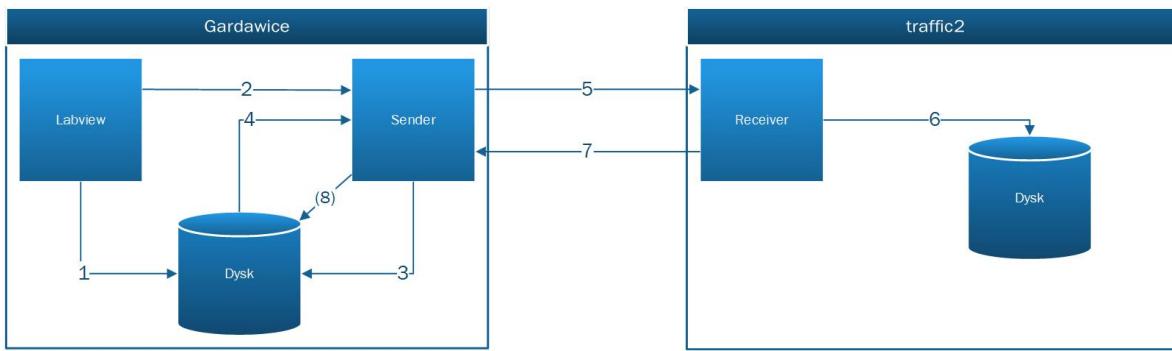
Ścieżki nadawane są w formie relatywnej, należy je interpretować względem katalogu bazowego danych pomiarowych. Ponadto nie posiadają one rozszerzenia, każdej ścieżce odpowiada para plików (avi, lvm). Zadaniem odbiorcy jest uzupełnienie ścieżki o powyższe informacje.

## 3.2. Schemat działania systemu

Schemat działania systemu służy opisowi interakcji pomiędzy komponentami będącymi jego częścią jak również dyskiem twardym. Ma on następującą postać:

1. LabVIEW zapisuje wynik pomiaru na dysku
2. LabVIEW wysyła informację o nowych danych do Sendera
3. Sender sprawdza obecność plików na dysku
4. Sender pobiera dane z dysku i przeprowadza kompresję zgodnie ze zdefiniowanym schematem
5. Dane zostają wysłane do Receivera
6. Receiver dekompresuje dane zgodnie ze zdefiniowanym schematem i zapisuje dane na dysku
7. Receiver wysyła informację o poprawnym zapisaniu danych do Sendera
8. (opcjonalnie) Sender usuwa przesłany plik z przestrzeni dyskowej

Powыższe informacje w formie graficznej prezentuje rysunek 3.2.



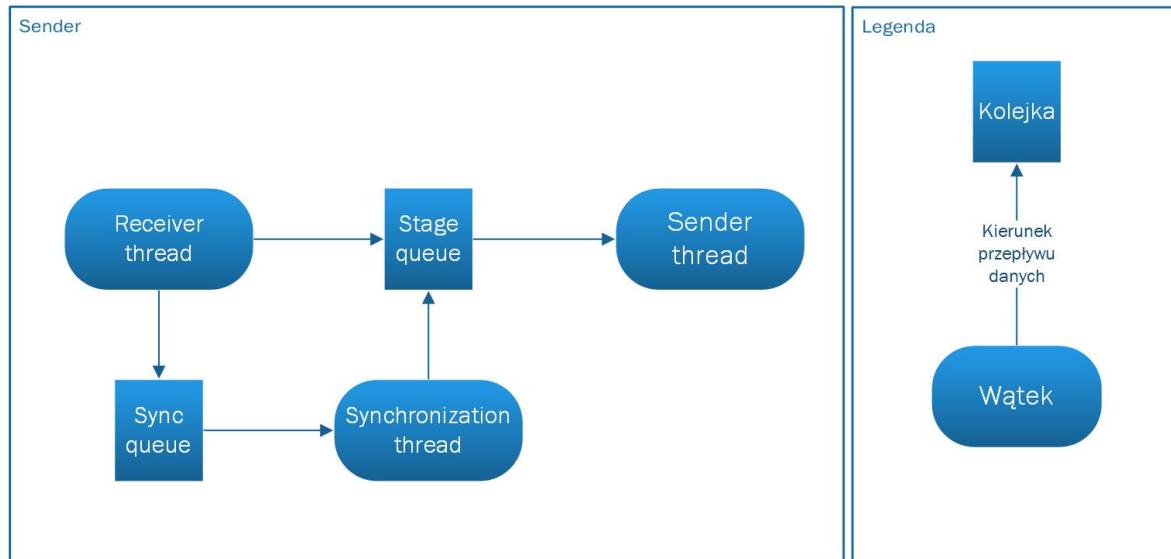
Rys. 25. Schemat przesyłu danych pomiarowych

Czynności pierwsza oraz druga dotyczą par plików będących wynikiem pomiaru. Pozostałe operują na pojedynczych plikach (ścieżkach).

### 3.2.1. Komponent Sender

Komponent Sender odpowiedzialny jest za równoległą obsługę dwóch czynności: odbierania informacji o nowych danych pomiarowych oraz przesyłu. W celu uniknięcia blokowania aplikacji na odczycie i zapisie, opisaną w rozdziale 1.2, zastosowano architekturę wielowątkową. Pliki do wysłania przekazywane są do wątku nadającego za pomocą kolejki dyskowej.

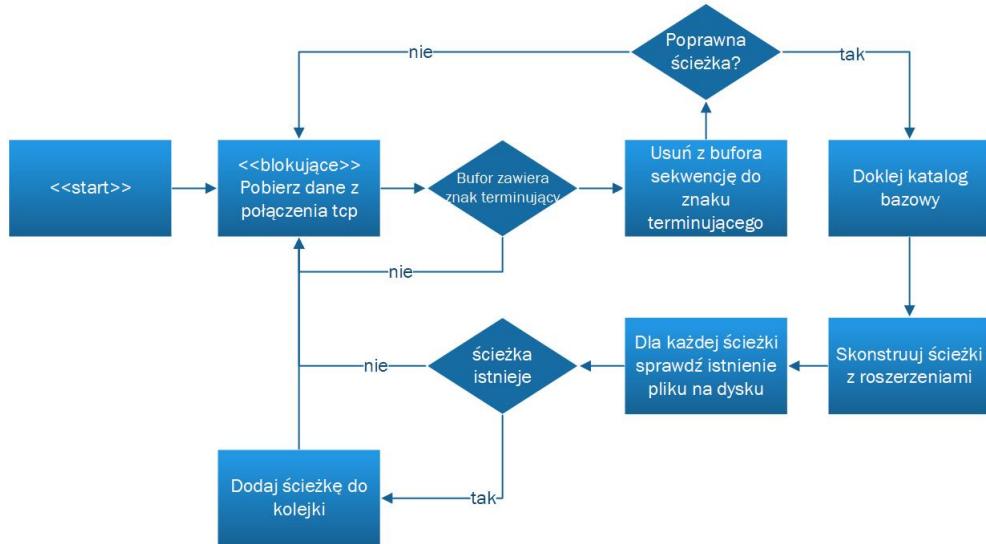
Wątki składające się na komponent Sender zaprezentowane zostały na rysunku 26. Przedstawia on również kolejki, które służą przechowywaniu informacji o plikach pomiędzy ich odebraniem a przesłaniem. Wątek SynchronizationThread odpowiedzialny jest za uzupełnienie stanu kolejki w przypadku awarii, jego działanie zostało opisane w podrozdziale 3.4.



Rys. 26. Schemat serwisu Sender

Schemat pracy wątku ReceiverThread przedstawia rys. 27. Odczytuje on sekwencje bajtów z połączenia TCP do napotkania znaku nowej linii. Sekwencja taka jest następnie spraw-

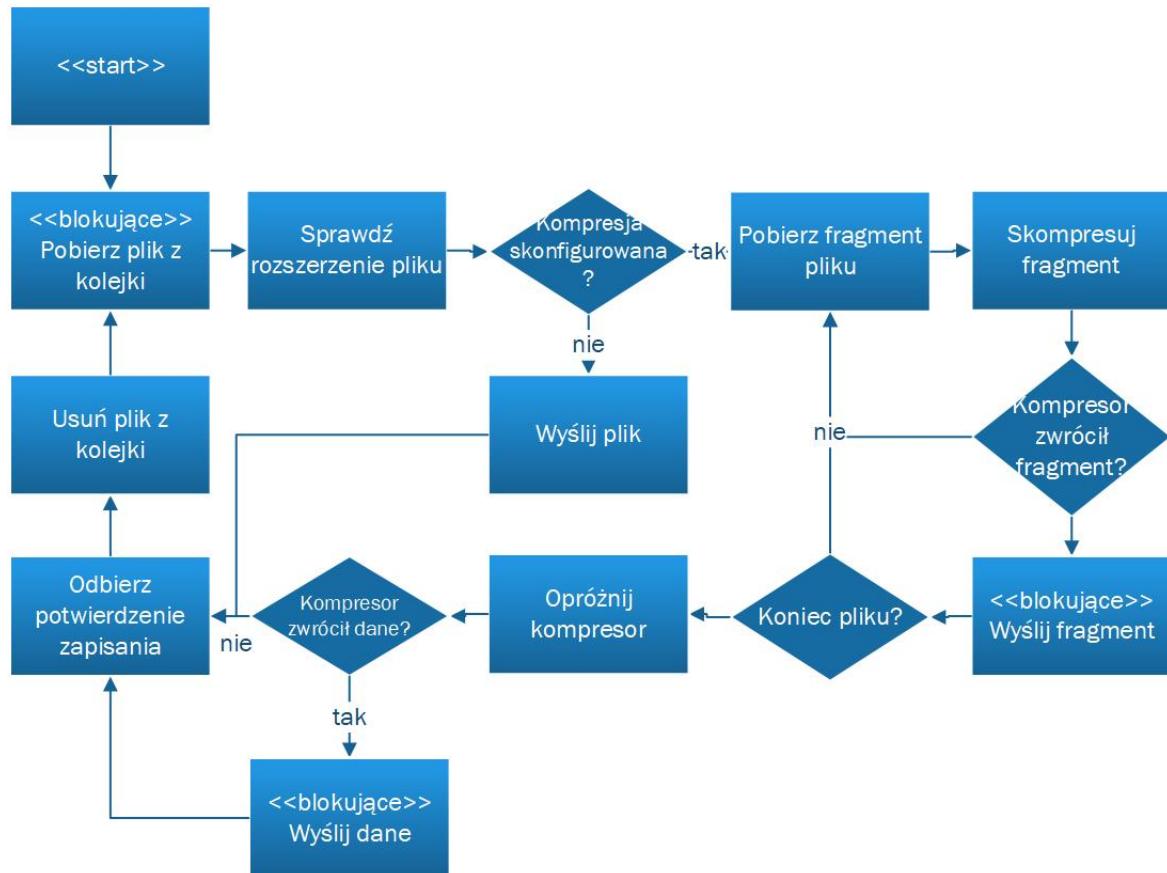
dzana pod kątem poprawności. Jeśli stanowi ona poprawną ścieżkę, uzupełniana jest o katalog bazowy oraz rozszerzenia plików. Pełne ścieżki weryfikowane są ponownie w celu sprawdzenia obecności pliku na dysku. Poprawna weryfikacja kończy się dodaniem pliku do kolejki wysyłania.



Rys. 27. Schemat pracy wątku ReceiverThread

Schemat pracy wątku SenderThread przedstawia rys. 28. Wątek oczekuje na pojawienie się danych w kolejce, analogicznie do odczytu z gniazda sieciowego. Gdy ścieżka pojawi się w kolejce jest ona pobierana, sprawdzane jest rozszerzenie pliku. Na tej podstawie dobierana jest metoda kompresji.

Rysunek zawiera uproszczone przedstawienie sekwencja wysyłania pliku nie wymagającego kompresji. W rzeczywistości, podobnie jak w przypadku kompresowanym, odczytuje on z dysku i nadaje plik fragmentami o ustalonej wielkości. Zabezpiecza to aplikację na wypadek przesyłania dużych plików. Przechowanie dużego pliku w pamięci ram może doprowadzić do jej przepełnienia.



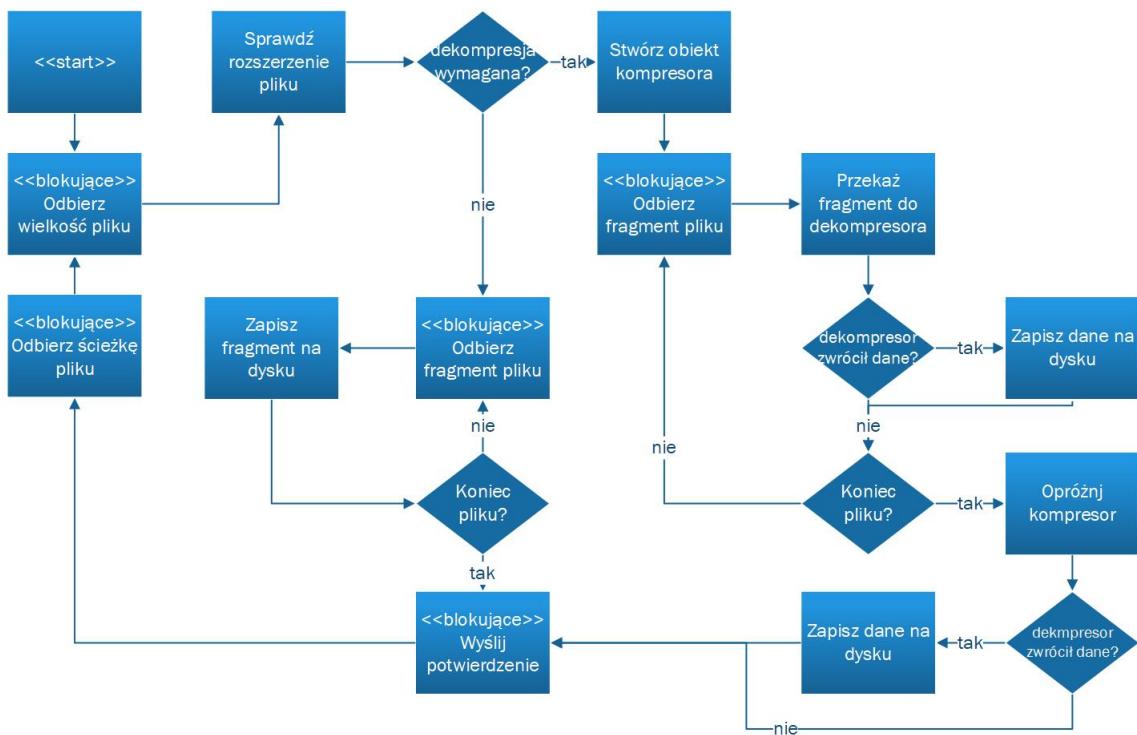
Rys. 28. Schemat pracy wątku SenderThread

Plik usuwany jest z kolejki dopiero po uzyskaniu potwierdzenia zapisania przez komponent Sender. Jeśli działanie aplikacji zostanie przerwane w trakcie wysyłania, informacja o potrzebie nadania pliku nie zostanie utracona.

### 3.2.2. Komponent Receiver

Serwis Receiver odpowiedzialny jest za odebranie połączenia od komponentu Sender, zdekodowanie ścieżki oraz wielkości pliku a następnie jego zapis na dysk. W zależności od typu pliku i konfiguracji aplikacji, może on również zdekompresować dane.

Schemat pracy komponentu Receiver przedstawia rys. 29. Nasłuchuje on na dane przychodzące z gniazda sieciowego i interpretuje je zgodnie ze schematem opisany w rozdziale 3.3. Plik zapisywany jest do dysku w miarę pojawiania się danych w kolejce odbiorczej gniazda sieciowego. W przypadku konfiguracji dekompresji dla odbieranego aktualnie typu pliku, dane odebrane z połączenia przetwarzane są przez dekompresor przed zapisaniem.



Rys. 29. Schemat pracy komponentu Receiver

Ważna z perspektywy pracy jest strategia zapisywania danych na komputerze docelowym. W projekcie zdecydowano się odtworzyć strukturę katalogów ze stanowiska pomiarowego. Katalog bazowy pomiarów jest wartością konfigurowalną. W przypadku ustawienia tego parametru na

```
/var/server_storage/
```

1

przesłany plik o ścieżce

```
R2018_03_11/M180311_182125.lvm
```

1

zostanie zapisany w ścieżce

```
/var/server_storage/R2018_03_11/M180311_182125.lvm
```

1

Odstępem od tej reguły jest przypadek gdy dekomprezja się nie powiedzie, w takim przypadku plik zostanie zapisany w ścieżce

```
/var/server_storage/failed_decompression/R2018_03_11/M180311_182125.lvm
```

1

### 3.3. Protokół przesyłu

Protokół TCP pozwala na transport ciągu bajtów. Zadaniem użytkownika jest dobranie, bądź skonstruowanie protokołu warstwy aplikacji, który zrealizuje przesył plików.

W badanym przypadku, aby poprawnie przesłać plik wymagane jest nadanie jego ścieżki w celu umiejscowienia go na dysku maszyny docelowej oraz wielkości, która pozwoli odbiorcy stwierdzić jaka część strumienia bajtów jest „ciąłem” pliku. Zarówno ścieżka jak i

wielkość pliku mogą mieć arbitralną wielkość liczoną w bajtach. Poinformowanie odbiorcy, gdzie kończy się dana informacja wymaga zastosowania sekwencji terminującej, nazywanej separatorem.

Konstrukcja separatora jest kluczowa dla poprawnej komunikacji, nie może on należeć do zbioru symboli, które mogą składać się na niesioną informację. Zarówno w przypadku ścieżek jak i wartości liczbowych można zastosować sekwencję znaków powrotu karetki i nowej linii.

Separatorów nie można stosować w przypadku danych zawierających losowe sekwencje bitów, takich jak pliki binarne, na przykład sekwencje wideo. Istnieje prawdopodobieństwo, że sekwencja stanowiąca separator wystąpi w środku danych i zaburzy transmisję. W takim przypadku wymagane jest przesłanie długości strumienia danych (pliku) przed przesaniem jego zawartości.

## Propozycja protokołu przesyłu

Opisem pliku w proponowanym protokole jest trójką uporządkowana postaci

<ścieżka do pliku>\r\n<długość pliku>\r\n<zawartość pliku>

1

która zostaje zapisana do strumienia. Komponent Receiver na podstawie ścieżki i wielkości pliku odbiera odpowiednią ilość bajtów z połączenia i zapisuje na dysku twardym. Sekwencję zastosowaną w pracy przedstawia algorytm1.

---

### Algorytm 1. Protokół przesyłu danych pomiarowych

#### Sender

1. wysyła ścieżkę do pliku względem katalogu bazowego pomiarów uzupełnioną o sekwencję „\r\n”
2. wysyła wielkość pliku zakodowaną w formie znaków ASCII uzupełnioną o sekwencję „\r\n”
3. wysyła zawartość pliku

#### Receiver

1. odbiera dane do sekwencji „\r\n”, sprawdza poprawność ścieżki
  2. odbiera dane do sekwencji „\r\n”, dekoduje do wartości liczbowej
  3. odbiera dokładnie tyle bajtów, ile zdekodowano w punkcie 2, zapisuje do dysku w ścieżce odebranej w punkcie 1 uzupełnionej o skonfigurowany katalog bazowy
- 

Schemat ten jest zredukowaną wersją sekwencji zapytanie-odpowiedź protokołu HTTP [14]. Linii zapytania (ang. request-line) odpowiada ścieżka pliku, zamiast nagłówków nadawana jest wielkość pliku po czym nadawane jest „ciało” zapytania, czyli zawartość pliku. W odpowiedzi wysyłana jest ścieżka z zapytania.

## 3.4. Scenariusze awarii i sposoby ich obsługi

Zagwarantowanie przesłania całości wygenerowanych pomiarów wymaga zabezpieczenia na przypadek awarii. Celem niniejszego rozdziału jest zidentyfikowanie możliwych scenariuszy błędów oraz zdefiniowanie zachowania systemu w momencie ich wystąpienia.

Zidentyfikowano następujące scenariusze błędów:

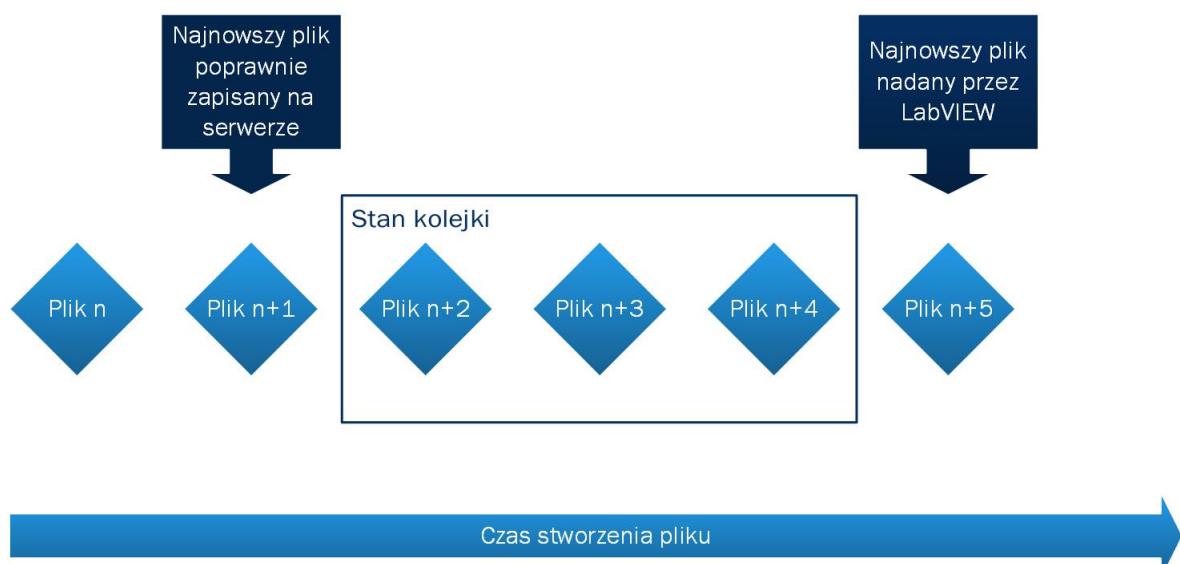
1. Odcięcie zasilania na stanowisku pomiarowym
2. Przerwanie łączności
3. Awaria komponentu LabVIEW

Dodatkowo należy liczyć się z awarią dysków twardych, jednak problemy tego typu obsługiwane są poprzez posiadanie zapasowych dysków twardych o identycznej zawartości z dyskiem głównym. Nie ma możliwości obsłużenia takiego scenariusza z poziomu aplikacji.

### 3.4.1. Odcięcie zasilania na stanowisku pomiarowym

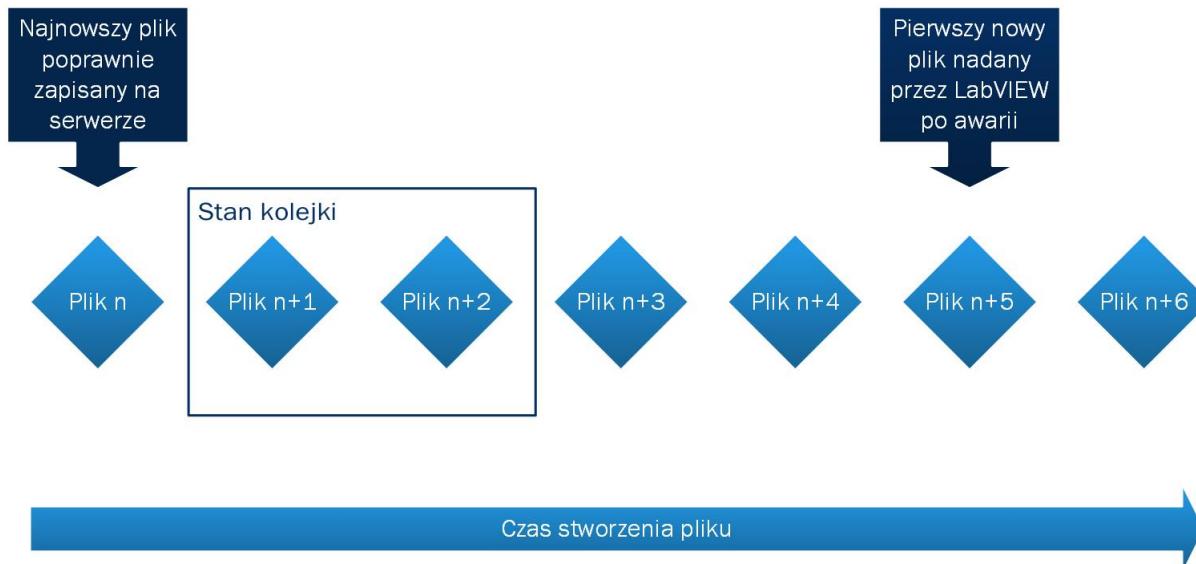
Głównym zagrożeniem z jakim ma do czynienia komponent Sender jest utrata informacji o danych pomiarowych czekających na wysłanie. Utrata zasilania powoduje wyczyszczenie stanu pamięci ram, co zaś powoduje utratę zawartości kolejki plików do wysłania. W celu przeciwdziałania takiemu scenariuszowi zastosowano kolejkę dyskową, jej stan nie jest tracony w momencie odcięcia zasilania.

Awaria może mieć również miejsce w momencie gdy komponent LabVIEW zapisał już dany plik na dysk twardy, jednak komponent Sender nie umieścił go jeszcze w kolejce plików do wysłania. Sytuację przedstawia rysunek 30. Po przywróceniu zasilania komponent LabVIEW wznowi pomiary i generowanie plików, nie nastąpi jednak ponowne rozgłoszenie brakującego pliku, przedstawionego na rysunku jako n+5.



Rys. 30. Graficzna reprezentacja informacji przechowywanych przez aplikacje

Sekwencja uruchamiania komponentów prowadzi do drugiego scenariusza utraty danych. Występuje on w przypadku gdy komponent LabVIEW zdąży wykonać pomiar zanim Sender nawiążę ponowne połączenie na port 6341. Ponownie traciona jest informacja o pojawienniu się nowych plików. Sytuację ilustruje rysunek 31.



Rys. 31. Graficzna reprezentacja zgubienia plików w stanie 'nowy'

Przypadki te wymagają skanowania katalogu pomiarów w poszukiwaniu brakujących plików. Przeszukiwanie dysku jest czynnością kosztowną obliczeniowo. Wyznaczono zestaw informacji, na podstawie którego można ograniczyć przestrzeń przeszukiwania:

1. Ścieżka do ostatniego pliku, który zmienił stan na monitorowany
2. Ścieżka pierwszego plik stworzony przez LabVIEW po ponownym nawiązaniu połączenia

Przeszukiwanie bazuje na założeniu, że jeśli na dysku istnieje niezsynchronizowany plik, to został on stworzony później niż ostatni plik odebrany od komponentu LabVIEW przed wystąpieniem awarii. Z drugiej strony plik ten nie został stworzony wcześniej niż pierwszy zapierowany po ponownym uzyskaniu połączenia. Sekwencję przeszukiwania przedstawia algorytm 2.

---

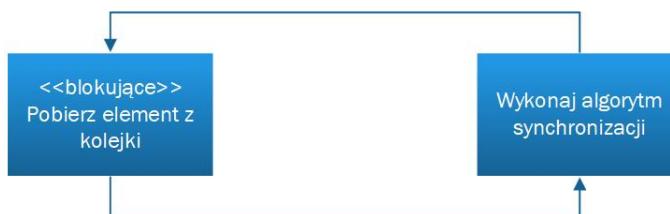
**Algorytm 2.** Algorytm synchronizacji plików po awarii

---

1. Znajdź wszystkie pliki w folderze źródłowym, które zostały stworzone w przedziale (ostatni zapisany, pierwszy nowy), posiadają monitorowane rozszerzenia i nie są folderami
  2. Dodaj te pliki do kolejki monitorowanych plików
  3. Znajdź wszystkie foldery, które zostały stworzone w badanym przedziale
  4. Dla każdego folderu, znajdź wszystkie pliki w folderze, które zostały stworzone w badanym przedziale i posiadają skonfigurowane rozszerzenia
  5. Dodaj te pliki do kolejki plików do wysłania
- 

W celu przechowywania informacji o ostatnim poprawnie zapisanym do kolejki pliku, zastosowano dodatkową wartość synchronizacyjną. Jest ona przechowywana na dysku twardym i modyfikowana po każdorazowym dodaniu pliku do kolejki. Synchronizacja odbywa się współbieżnie z podstawowym działaniem programu w wątku SynchronizationThread.

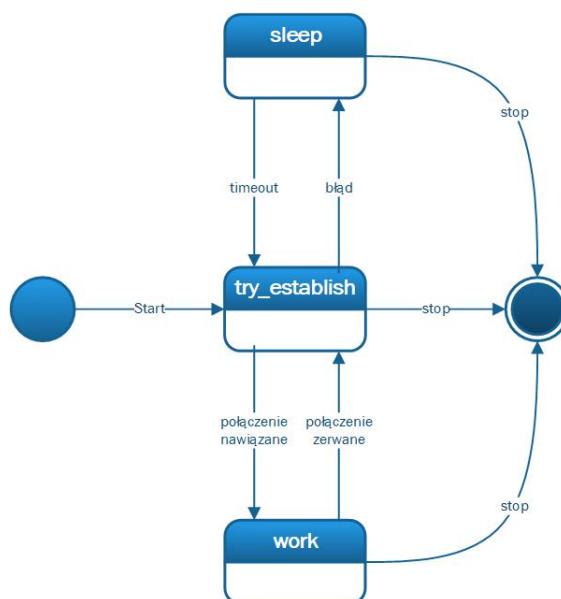
Rys. 32 przedstawia schemat działania wątku synchronizacyjnego. Informacje dodawane są do kolejki synchronizacyjnej tylko w momencie nawiązaniu połączenia z komponentem LabVIEW przez wątek SenderThread.



Rys. 32. Schemat działania wątku SynchronizationThread

### 3.4.2. Przerwanie łączności ze stanowiskiem pomiarowym

Przerwa w łączności może wystąpić w dowolnym momencie działania aplikacji i trwać arbitralnie długo. Należy zadbać, aby wątek realizujący komunikację sieciową zareagował na takie zdarzenie w kontrolowany sposób. Strategia obsługi zakłada cykliczne wykonywanie prób nawiązania połączenia. Poszczególne próby rozdzielone są czasem oczekiwania, którego wartość jest konfigurowana w trakcie uruchomienia aplikacji. Maszynę stanów tego mechanizmu przedstawia rysunek 32.



Rys. 33. Maszyna stanów nawiązywania połączenia

Wątek obsługujący połączenie sieciowe w momencie uruchomienia posiada stan try\_establish, odpowiedzialny za próbę wykonania połączenia. W przypadku błędu zostaje on uśpiony po czym próba jest ponawiana. Poprawne nawiązanie połączenia prowadzi do przejścia w stan work. W jego trakcie wykonywana jest obsługa połączenia. Zerwanie łączności ponownie wprowadza aplikację w stan try\_establish.

Od komponentu Receiver wymagane jest przejście w stan nasłuchu i oczekiwanie na połączenie przychodzące.

### 3.4.3. Awaria komponentu LabVIEW

Awaria ta wymaga obsługi przez komponent Sender. Wątek ReceiverThread przechodzi w sekwencję nawiązywania połączenia zgodną z maszyną stanów przedstawioną na rysunku 33. Wątek SenderThread kontynuuje nadawanie plików, w momencie opróżnienia kolejki przechodzi w stan bezczynności.

## 3.5. Wybór języka programowania

Dobór języka programowania do problemu jest kluczowy ze względu na ilość czasu wymaganego na implementację, możliwości rozbudowy rozwiązania oraz przenośność. Wyszczególniono następujące kryteria:

1. Wieloplatformowość - kod powinien nie wymagać żadnych zmian przed uruchomieniem na Windowsie i Linuxie.
2. Szybkie prototypowanie - najwolniejszym elementem systemu jest połączenie sieciowe, z tego powodu zastosowanie języków wykorzystujących procesor w sposób bardziej efektywny nie zmienia czasu reakcji systemu. Język powinien m.in. automatycznie zarządzać pamięcią.

3. Bogata biblioteka standardowa - ze względu na trudności w dostępie do stanowiska pomiarowego instalowanie zewnętrznych komponentów należy ograniczyć do minimum. Absolutnym minimum jest biblioteka obsługująca protokół TCP.

Języki C i C++ nie spełniają żadnego z założeń. Wymagają komplikacji na docelową platformę, nie posiadają w standardzie bibliotek pozwalających na operowanie stosem sieciowym oraz wymagają ręcznego zarządzania pamięcią. Cechą pozytywną ich zastosowania jest bezpośredni dostęp do wywołań systemowych takich jak inotify na systemach Linux. Wykorzystywanie połączeń TCP w języku C wymaga pracy bezpośrednio z użyciem gniazd, w języku C++ można zastosować to samo podejście, bądź bibliotekę boost::asio. W przypadku programów pisanych zarówno na systemu Linux i Windows, wymagane jest zastosowanie różnych kompilatorów oraz samodzielne wypracowanie warstw abstrakcji celem osiągnięcia przenośności kodu.

Z powodu powyższych problemów zastosowano język Python, spełnia on założenia wymienione na początku podrozdziału. Prócz powyższych cech kod w tym języku jest interpretowany, a nie kompilowany, co zdejmuję z użytkownika wymogu stworzenia systemu budowania dostosowanego do obu docelowych platform. Uruchomienie kodu wymaga skopiowania źródeł na platformę docelową i uruchomienia interpretera z odpowiednią ścieżką.

Tabela 11 przedstawia zestawienie funkcjonalności wymaganych przez projekt oraz ich dostępność w języku Python. Biblioteką standardową określany jest zestaw bibliotek, które są instalowane razem z interpreterem języka, programem Python, i nie wymagają samodzielnej instalacji.

**Tab. 11.** Zestawienie zagadnień wymaganych w projekcie i dostępności bibliotek

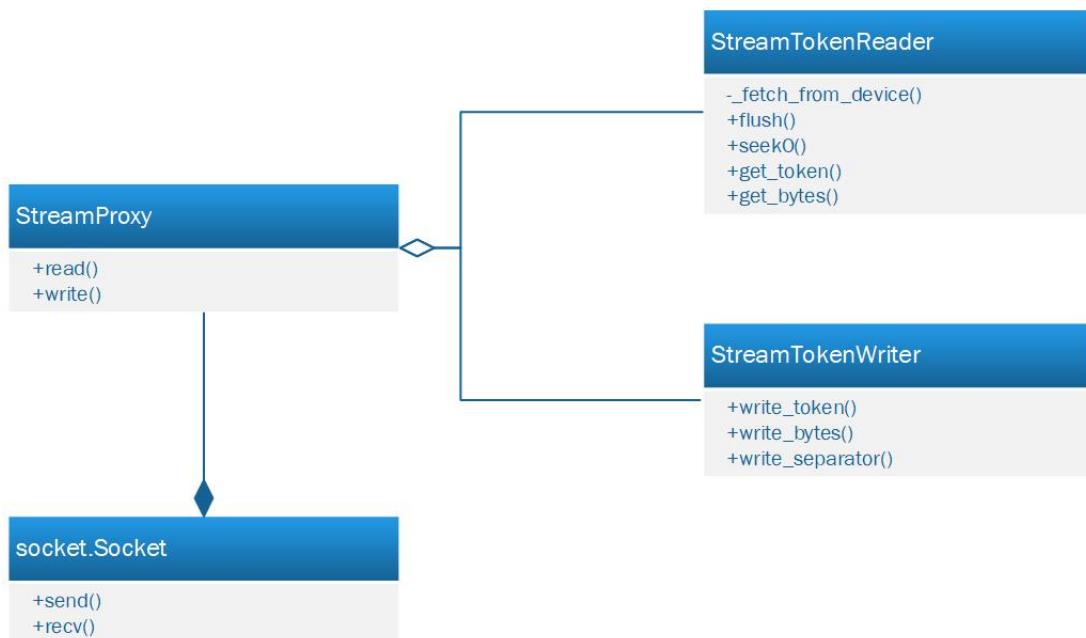
Zagadnienie	Nazwa biblioteki	Uwagi
Monitorowanie i manipulacja stanem dysku	pathlib	Dostępna w bibliotece standardowej
Obsługa połączeń sieciowych	socket, socketserver	Dostępne w bibliotece standardowej
Przetwarzanie parametrów podawanych z konsoli	argparse	Dostępna w bibliotece standardowej
Rejestrowanie i przetwarzanie czasu	datetime	Dostępna w bibliotece standardowej
Kompresja i dekompresja	lzma, bz2, zlib	Dostępne w bibliotece standardowej
Kolejki dyskowe	brak	Dostępne w postaci zewnętrznych bibliotek, nie koniecznie napisanych w pythonie z możliwością wywołania z tego języka
Kompresja x264	brak stabilnych	Pakiet ffmpeg wymaga uruchomienia jako oddzielny proces i przekierowania danych
Uruchamianie i manipulacja procesami systemowymi	subprocess	Dostępna w bibliotece standardowej

Analogiczny poziom abstrakcji oferuje język Ruby a równie bogaty zestaw bibliotek język Go. Nie zostały one jednak zastosowane w pracy.

## 3.6. Obsługa danych strumieniowych

W celu ułatwienia odbioru i nadawanie ustrukturyzowanych danych w implementacji wprowadzono zestaw klas pomocniczych. Kod aplikacji nie prowadzi zapisów bezpośrednio do strumienia. Do przeprowadzenia nadawania stosowana jest klasa StreamTokenWriter, odbiorowi zaś służy klasa StreamTokenReader. Diagram klas przedstawia rysunek 34.

Zapis danych o arbitralnej długości wymaga zakończenia ich sekwencją terminującą, czynność tą realizuje metoda write\_token z klasy StreamTokenWriter, za jej pomocą nadawane są ścieżka do pliku oraz długość w bajtach. Zawartość pliku nadawana jest za pomocą metody write\_bytes.



Rys. 34. Klasy służące do manipulacji strumieniem

Odebranie sekwencji zakończonych separatorem może wymagać więcej niż jednego odczytu z kolejki odbiorczej stosu sieciowego. Przechowanie danych do momentu napotkania separatora wymaga buforowania. Realizowane jest ono przez klasę StreamTokenReader. Sekwencję odczytu przedstawia algorytm 3.

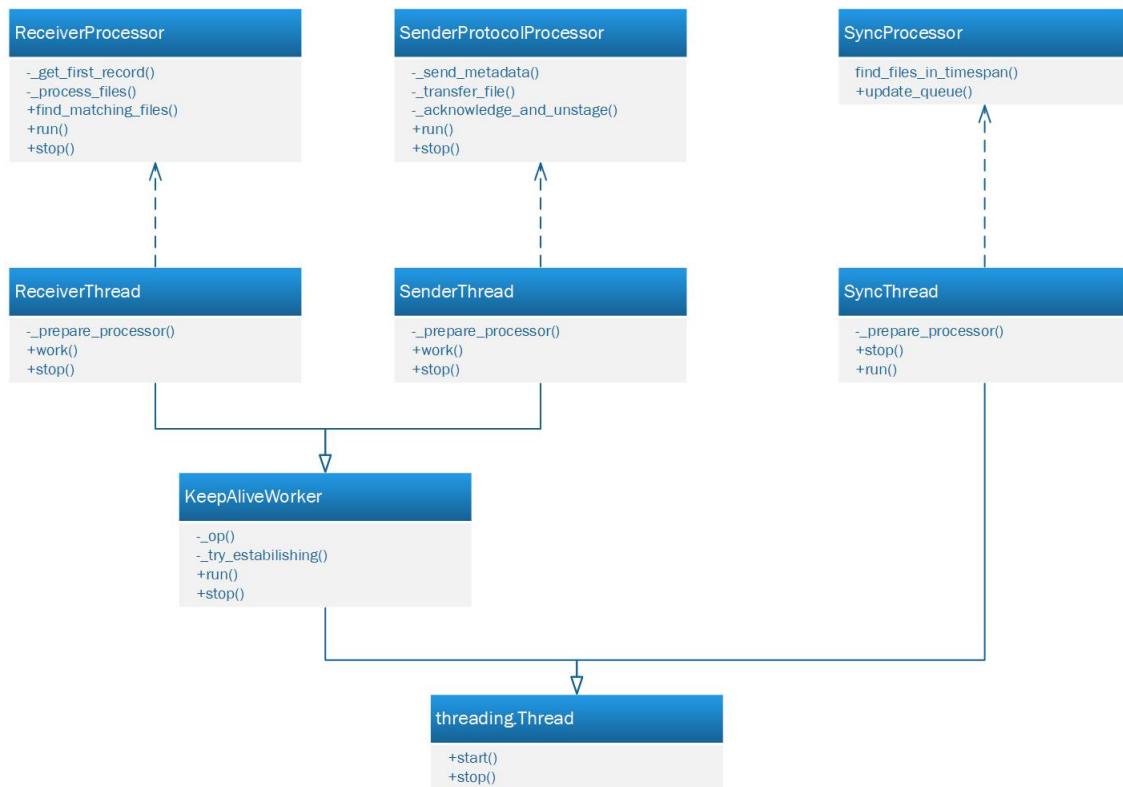
#### **Algorytm 3.** Pobierania danych o arbitralnej długości ze strumienia

1. Sprawdź czy w buforze znajduje się separator
  - (a) Tak: zwróć dane do jego pozycji i usuń je z bufora razem z separatorem, zakończ
  - (b) Nie: pobierz fragment danych, przejdź do punktu 1

## 3.7. Implementacja komponentu Sender

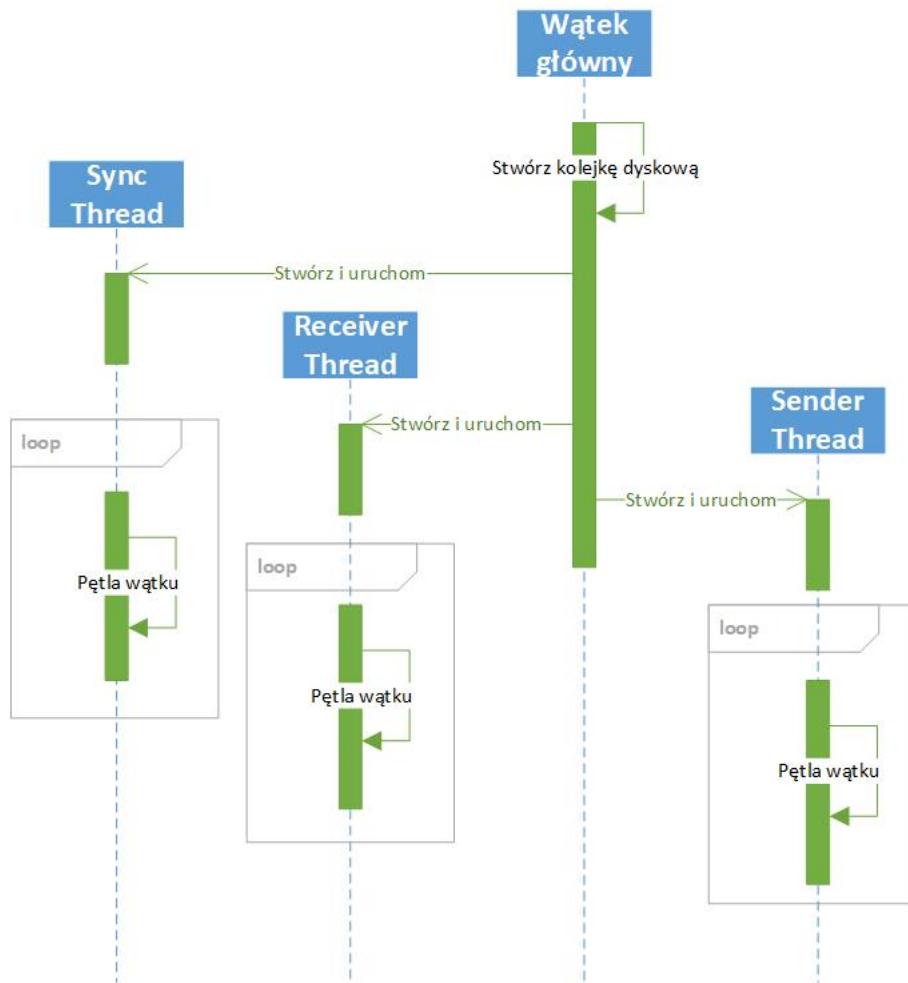
Kluczowymi elementami w implementacji Komponentu sender jest podział obsługi komunikacji na etap negocjacji połączenia i faktycznego przesyłu bądź odbioru danych.

Diagram klas przedstawia rysunek 35. Do tworzenia i obsługi wątków wykorzystana jest klasa Thread z biblioteki threading dostępnej w języku Python. Wątki SenderThread i ReceiverThread dziedziczą po klasie KeepAliveWorker realizującej maszynę stanów przedstawioną na rys. 33. Kod odpowiedzialny za realizację protokołów komunikacyjnych uruchamiany jest w metodzie work klas dziedziczących po KeepAliveWorker. Zostanie ona wywołana po poprawnym nawiązaniu połączenia na zadany adres. Komunikacja pomiędzy wątkami realizowana jest za pomocą kolejek zgodnie ze schematem opisany w sekcji 3.2.



Rys. 35. Diagram klas komponentu Sender

Schemat rozruchu aplikacji przedstawia rys. 36. Wątek główny aplikacji odpowiedzialny jest za uruchomienie wątków realizujących poszczególne funkcjonalności oraz poprawne ich zwolnienie w przypadku błędów. Regularny tryb działania nie przewiduje zatrzymywania działania aplikacji, nadawanie odbywa się ciągle, w miarę napływu danych pomiarowych. Każdy z wątków posiada mechanizm odpowiedzialny za przechwytywanie błędów oraz przywrócenia trybu operacyjnego. Kolejność rozruchu podyktywana jest priorytetem operacji. Synchronizacja i odbieranie pomiarów z LabVIEW dostarczają danych, na których może pracować wątek wysyłający. Pierwszy uruchamiany jest wątek synchronizacyjny, następnie wątek odbiorczy a na końcu nadawczy.



Rys. 36. Schemat rozruchu komponentu Sender

### 3.7.1. Konfiguracja i uruchamianie

Aplikacja przyjmuje serię parametrów konfiguracyjnych takich jak adresy komponentów Labview i Receiver, interwał pomiędzy próbami nawiązania połączenia czy schematy kompresji.

Tabela 12 przedstawia opcje konfiguracyjne komponentu Sender. Wyróżnione są dwie główne sekcje

- server - odpowiedzialna za interakcję z komponentem Receiver
- provider - odpowiedzialna za interakcję z komponentem Labview

Obie sekcje posiadają ustawienia dotyczące komunikacji sieciowej za pomocą protokołu TCP oraz interwał sterujący częstotliwością prób nawiązywania połączenia po błędzie. Aplikacja posiada konfigurowalny zestaw rozszerzeń oraz schematów kompresji. Konfiguracja zawiera również zestaw parametrów dotyczących lokalizacji przechowywania stanu aplikacji na dysku twardym.

**Tab. 12.** Opcje konfiguracyjne komponentu Sender

Sekcja	Opcja	Typ	Opis
server	host	Ciąg znaków	Adres IP komponentu Receiver
server	port	16 bit unsigned integer	Port TCP komponentu Receiver
server	retry_time	unsigned integer	Czas pomiędzy próbami nawiązania połączenia z komponentem Receiver
server	processor	Ciąg znaków	Włącza/wyłącza możliwość kompresji
provider	host	Ciąg znaków	Adres IP komponentu Labview
provider	port	Ciąg znaków	Port TCP komponentu Labview
provider	extensions	Lista ciągów znaków	Lista rozszerzeń plików akceptowanych przez aplikację
provider	retry_time	unsigned integer	Czas pomiędzy próbami nawiązania połączenia z komponentem Labview
provider	separator	Ciąg znaków	Separator pomiędzy kolejnymi ścieżkami nadawanymi przez komponent Labview
	storage_root	Ciąg znaków	Katalog bazowy pomiarów
	stage_queue_path	Ciąg znaków	Ścieżka do pliku przechowującego stan kolejki
	cache_path	Ciąg znaków	Ścieżka do pliku przechowującego ostatni nadany plik
	Compression	Mapa ciąg znaków na ciąg znaków	Mapowanie pomiędzy rozszerzeniami plików do wysłania a używanym kompresorem

Komponent umożliwia podanie konfiguracji za pomocą flag rozruchowych oraz w formie pliku w formacie JSON (ang. JavaScript Object Notation). Rysunek 37 przedstawia pełną konfigurację aplikacji w formie pliku. Konfiguracja zakłada odbieranie plików lvm oraz avi i składowanie ich w katalogu /tmp/client\_storage. Ponadto dla powyższych typów plików skonfigurowana została kompresja zgodna z wnioskami z sekcji 2.5.

```

1  {
2    "server": {
3      "host": "127.0.0.1",
4      "port": 50123,
5      "retry_time": 30,
6      "processor": "compression"
7    },
8    "provider": {
9      "host": "127.0.0.1",
10     "port": 50321,
11     "extensions": ["lvm", "avi"],
12     "retry_time": 30,
13     "separator": "\n"
14   },
15   "storage_root": "/tmp/client_storage",
16   "stage_queue_path": "/tmp/state_storage/stage.queue",
17   "cache_path": "/tmp/state_storage/client.cache",
18
19   "compression": {
20     "avi": "x264",
21     "lvm": "bzip2",
22     "mp4": null
23   }
24 }
```

Rys. 37. Przykładowa konfiguracja komponentu Sender w formacie JSON

Przełączniki podawane przy rozruchu aplikacji przedstawia tabela 13. W przypadku braku podania ścieżki do pliku konfiguracyjnego aplikacja uruchamiana jest z domyślnymi wartościami parametrów.

Flaga	Opis
-h, --help	Wypisuje dostępne opcje konfiguracyjne
-c, --config	Ścieżka do pliku konfiguracyjnego w formacie JSON
--options	Pozwala na nadpisanie konfiguracji z konsoli za pomocą wpisu w postaci „<sekcja>.<pole> <wartosc>”. Nie wspiera wartości będących słownikami bądź listami
--delete_acknowledged	Włącza tryb usuwania plików po poprawnym nadaniu

Tab. 13. Flagi rozruchowe aplikacji Sender

Przełącznik –options pozwala na zmianę pojedynczych opcji w trakcie rozruchu. Sposób jego stosowania przedstawia poniższy fragment kodu

```
python3 client.py -c config.json --options server.port 31337
```

1

przypadek ten nadpisuje parametr port z sekcji serwer wartością 31337.

Rozruch komponentu Sender z użyciem pliku konfiguracyjnego client.json przedstawia 38. Aplikację uruchomiono lokalnie bez dostępu do zewnętrznych komponentów, można zaobserwować komunikaty z maszyny stanów nawiązywania połączenia wątków Receiver-Thread i SenderThread.

```

→ implementacja git:(master) ✘ python3 client.py -c client.json
2018-06-24 14:05:53,903 [__main__|MainThread] INFO: Setting up client
2018-06-24 14:05:53,905 [__main__|MainThread] INFO: Pulling configuration from file (./client.json)
2018-06-24 14:05:53,909 [__main__|MainThread] INFO: Starting synchronization service
2018-06-24 14:05:53,909 [sync.client.ReceiverService|MainThread] INFO: Starting receiver service
2018-06-24 14:05:53,910 [__main__|MainThread] INFO: Starting sender service
2018-06-24 14:05:54,917 [sync.utils.Workers|ReceiverThread] WARNING: Connection refused, sleeping...
2018-06-24 14:05:54,917 [sync.utils.Workers|SenderThread] WARNING: Connection refused, sleeping...

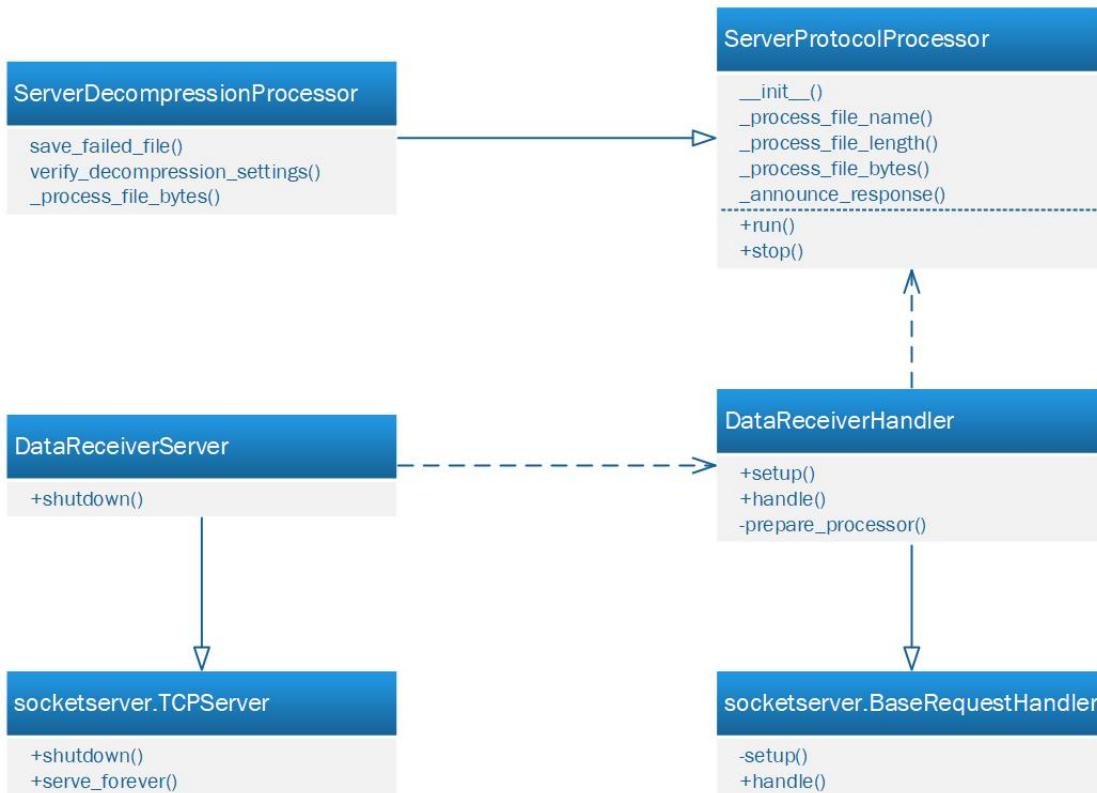
```

Rys. 38. Logi z rozruchu komponentu Sender

## 3.8. Implementacja komponentu Receiver

Komponent Receiver odpowiedzialny jest za odbiór i opcjonalną dekompresję. W architekturze klient-serwer jest on serwerem czyli oczekuje na przychodzące połączenia. Funkcjonalność ta została zrealizowana za pomocą biblioteki socketserver dostarczanej w standardzie języka Python.

Diagram klas komponentu Receiver przedstawia rysunek. W przypadku zerwania połączenia przejście w stan nasłuchu obsługuje klasa TCPServer z biblioteki standardowej. W momencie nawiązania połączenia klasa DataReceiverServer tworzy obiekt klasy DataReceiverHandler posiadający metodę handle, która uruchamia wątek główny obsługi połączenia. Maszyna stanów połączenia realizowana jest w klasie ServerProtocolProcessor, zgodnie ze schematem przedstawionym w sekcji 3.2.



Rys. 39. Schemat klas komponentu Receiver

### 3.8.1. Konfiguracja i uruchamianie

Konfiguracja komponentu Receiver ponownie zawiera katalog bazowy danych pomiarowych oraz adres nasłuchu. Zestaw dostępnych opcji konfiguracyjnych przedstawia tabela 14. Ilość parametrów jest zdecydowanie mniejsza niż w przypadku komponentu Sender.

**Tab. 14.** Opcje konfiguracyjne komponentu Receiver

Opcja	Typ	Opis
host	Ciąg znaków	Adres IP nasłuchu Receivera
port	16 bit unsigned integer	Port TCP nasłuchu Receivera
storage_root	Ciąg znaków	Ścieżka bazowa, w której zapisywane będą pomiary
decompression	Mapa ciąg znaków na ciąg znaków	Konfiguruje schemat dekompresji dla typu pliku

Konfigurację komponentu Receiver w formacie JSON przedstawia rysunek 40. Ponownie zaprezentowano wartości używane w testach lokalnych.

```

1 {
2   "host": "127.0.0.1",
3   "port": 50123,
4   "storage_root": "/tmp/server_storage",
5   "decompression": {
6     "lvm": "bzip2"
7   }
8 }
```

**Rys. 40.** Przykładowa konfiguracja komponentu Receiver w formacie JSON

Flagi, które można przekazać w trakcie uruchamiania aplikacji, przedstawia tabela 41. Podobnie jak w komponencie Sender, dostępna jest możliwość wczytania konfiguracji z pliku oraz nadpisania części konfiguracji argumentami podanymi z wiersza poleceń.

**Rys. 41.** Flagi rozruchowe komponentu Receiver

Flaga	Opis
-h, -help	Wypisuje dostępne opcje konfiguracyjne
-c, -config	Opcjonalna ścieżka do konfiguracji w pliku w formacie JSON
-options	Analogicznie jak w komponencie Sender

Rozruch komponentu Receiver przedstawia rysunek 42, Na wyjście standardowe wypisany został tryb podania konfiguracji oraz adres, na którym nasłuchuje aplikacja.

```

→ implementacja git:(master) ✘ python3 server.py -c server.json
2018-06-24 15:00:42,671 [__main__|MainThread] INFO: Setting up server
2018-06-24 15:00:42,672 [__main__|MainThread] INFO: Pulling configuration from file (./server.json)
2018-06-24 15:00:42,676 [__main__|MainThread] INFO: Listening ('127.0.0.1', 50123)
```

**Rys. 42.** Logi z rozruchu komponentu Receiver



## **4. Testy działania systemu przesyłu**

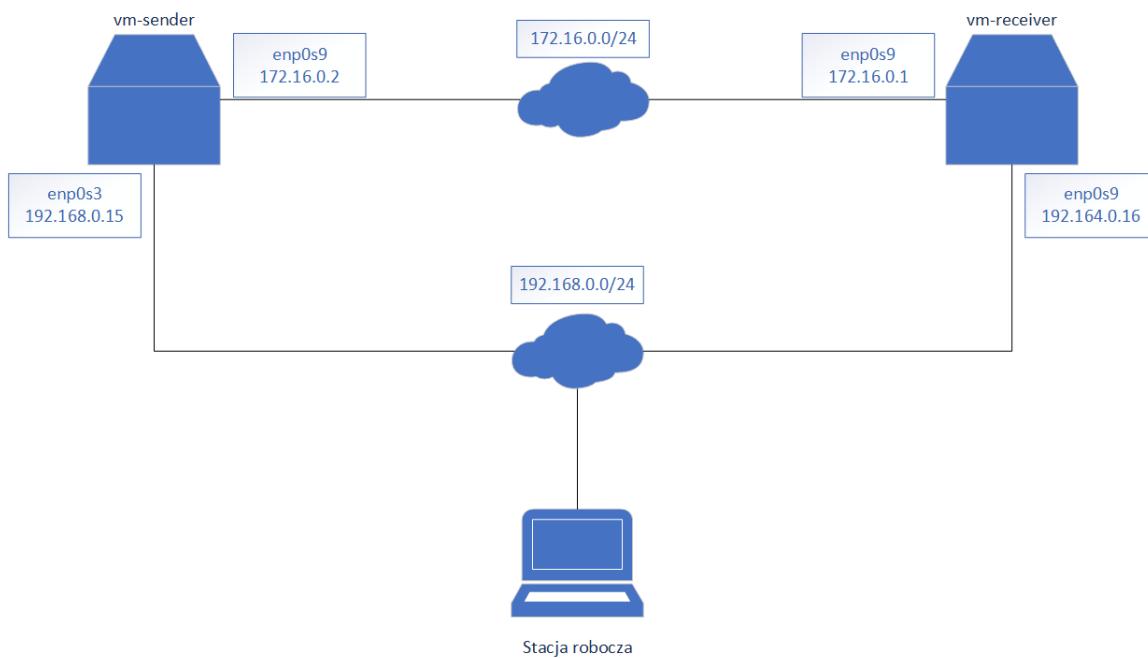
Celem niniejszego rozdziału jest weryfikacja działania systemu przesyłu. Wielkością opisującą działanie systemu jest długość kolejki plików do wysłania, jej zmienność w czasie opisuje czy system w danej konfiguracji zdolny jest nadawać dane bez opóźnień.

Testy symulacyjne przeprowadzono dla par kompresorów (bzip2, x264) oraz (wavelet, x264). Na stanowisku pomiarowym przeprowadzono test pary (bzip2, x264) ze względu na problemy z instalacją biblioteki PyWavelets, realizującej transformację falkową.

### **4.1. Zachowanie systemu w warunkach symulacyjnych**

Celem zbadania przepustowości aplikacji w konfiguracji realizującej kompresje stworzono środowisko testowe z interfejsem sieciowym posiadającym ograniczoną przepustowość. Przykłady konfiguracji przedstawiają przygotowanie symulacji dla pary kompresorów (bzip2, x264). Symulację działania dla pary (wavelet, x264) testowano w taki sam sposób.

Schemat stanowiska testowego przedstawia rysunek 43. Składa się ono z dwóch maszyn wirtualnych odpalonych na komputerze osobistym. Maszyna wirtualna vm-sender odpowiada komputerowi znajdującemu się w miejscowości Gardawice, uruchomiono na niej komponent Sender oraz narzędzie symulujące komponent LabVIEW. Przesył plików odbywa się za pośrednictwem sieci 172.16.0.0/24. Sieć 192.168.0.0 służy do konfiguracji obu maszyn ze stacji roboczej za pomocą protokołu SSH.



Rys. 43. Schemat stanowiska testowego

W celu odwzorowania warunków stanowiska roboczego ograniczono przepustowość sieci 172.16.0.0/24. W tym celu użyto programu „tc”, który pozwala manipulować parametrami nadawania pakietów. Skrypt użyty do ustawienia docelowej prędkości łączka przedstawia rysunek 44. Przyjmuje on trzy argumenty pozycyjne:

1. Modyfikowany interfejs sieciowy
2. Oczekiwana prędkość przesyłu
3. Oczekiwany czas przesyłu pakietu po sieci

```

1 #!/usr/bin/env bash
2 sudo tc qdisc del dev $1 root &>/dev/null
3 sudo tc qdisc add dev $1 root tbf rate $2 latency $3 burst 1540
4 tc qdisc show dev $1

```

Rys. 44. Kod skryptu set\_interface\_rates.sh służącego ograniczeniu przepustowości łączka

Na obu maszynach wirtualnych skrypt wywołano z następującymi parametrami

```
./set_interface_rates.sh enp0s9 58kbps 200ms
```

<sup>1</sup>

Przepustowość łączka po użyciu skryptu przetestowano za pomocą programu iPerf3. Wynik jego działania na maszynie vm-sender przedstawia rysunek 45. Prędkość nadawania oszacowana została na 55.2 kilobajta na sekundę, odbioru zaś na 53.7 kilobajta na sekundę. Wartości te są niższe niż zmierzone na stanowisku pomiarowym.

```
woyski@vm-sender:/d/linux_workspace/mgr/implementacja$ iperf3 -f KBytes -c 172.16.0.1 -p 31337 -t 100 -i 50
Connecting to host 172.16.0.1, port 31337
[ 4] local 172.16.0.2 port 51198 connected to 172.16.0.1 port 31337
[ ID] Interval      Transfer     Bandwidth    Retr Cwnd
[ 4]  0.00-50.00 sec   2.78 MBytes  57.0 KBytes/sec   3  17.0 KBytes
[ 4] 50.00-100.00 sec   2.61 MBytes  53.5 KBytes/sec   0  17.0 KBytes
[ 4]          - - - - -
[ ID] Interval      Transfer     Bandwidth    Retr
[ 4]  0.00-100.00 sec   5.39 MBytes  55.2 KBytes/sec   3
[ 4]          - - - - -
[ 4]  0.00-100.00 sec   5.24 MBytes  53.7 KBytes/sec
                                         sender
                                         receiver
iperf Done.
```

Rys. 45. Wynik działania programu iPerf3 na maszynie vm-sender

Symulacja komponentu LabVIEW zakłada nadawanie realnych danych pomiarowych co określony interwał. Dane testowe zostały wygenerowane przez stanowisko pomiarowe 06.10.2017. Czas pomiędzy kolejnymi pomiarami został wyliczony na podstawie zbioru testowego wykorzystanego w rozdziale 2.

$$N = 165249$$

$$T = 28$$

$$\Delta T = \frac{T * 24 * 60 * s}{N} = \frac{2419200}{165249} \frac{s}{pomiary} = 14.64 \frac{s}{pomiary}$$

gdzie

$N$  – liczba wygenerowanych plików

$T$  – liczba dni

$\Delta T$  – interwał pomiędzy pomiarami

Rozruch wykonano za pomocą komendy

```
python3 sync/labview_sim/data_provider.py -hs 127.0.0.1 -p 50321 -s
client_storage --extensions lvm avi --sleep 14.64
```

Opis flag wykorzystanych do uruchomienia symulatora przedstawia tabela 15. Aplikacja nasłuchiwa na interfejsie lokalnym IP oraz porcie TCP 50321. Katalog bazowy pomiarów ustalony został na miejsce przechowywania danych pomiarowych z dnia 06.10.2017.

Tab. 15. Flagi rozruchowe symulatora

Flaga	Wartość	Opis
-hs	127.0.0.1	Adres IP symulatora
-p	50321	Port TCP symulatora
-s	client_storage	Katalog bazowy pomiarów
-extensions	lvm avi	Rozgłaszone rozszerzenia plików
-sleep	14.64	Czas pomiędzy kolejnymi rozgłoszeniami

Komponent Receiver został uruchomiony za pomocą następującej komendy

```
python3 server.py -c server.json
```

Konfigurację testową komponentu Sender przedstawia rysunek 46. Nasłuchiwa on pod adresem 127.16.0.1:50123 odpowiadającym sieci testowej, katalog bazowy pomiarów przyjął wartość server\_storage. Ponadto pliki lvm zostaną zdekompresowane po odebraniu.

```

1 {
2   "host": "172.16.0.1",
3   "port": 50123,
4   "storage_root": "server_storage",
5
6   "decompression": {
7     "lvm": "bzip2"
8   }
9 }
```

**Rys. 46.** Konfiguracja testowa komponentu Receiver

Komponent Sender został uruchomiony za pomocą następującej komendy

```
python3 client.py -c client.json
```

1

Konfigurację testową komponentu Sender przedstawia rysunek 47. Wartości par (host, port) klauzul „server” i „provider” odpowiadają konfiguracjom symulatora oraz Receivera.

Symulator i komponent Sender współdzielą katalog, w którym zostały uruchomione, stąd katalog bazowy pomiarów - client\_storage - wskazuje w obu przypadkach na tą samą lokalizację. Schemat kompresji zgodnie z wnioskami z sekcji 2.5.

```

1 {
2   "server": {
3     "host": "172.16.0.1",
4     "port": 50123,
5     "retry_time": 30,
6     "processor": "compression"
7   },
8   "provider": {
9     "host": "127.0.0.1",
10    "port": 50321,
11    "extensions": ["lvm", "avi"],
12    "retry_time": 30,
13    "separator": "\n"
14  },
15  "storage_root": "client_storage",
16  "stage_queue_path": "state_storage/stage.queue",
17  "cache_path": "state_storage/client.cache",
18
19  "compression": {
20    "avi": "x264",
21    "lvm": "bzip2",
22    "mp4": null
23  }
24 }
```

**Rys. 47.** Konfiguracja testowa komponentu Sender

Długość kolejki plików do wysłania odczytywano bezpośrednio z kolejki dyskowej. Rys. 48 przedstawia skrypt w języku Python, za pomocą którego monitorowano stan kolejki dyskowej. Przyjmuje on dwa argumenty

1. Ścieżkę do pliku przechowującego stan kolejki
2. Ścieżkę do logu stanu kolejki

W trakcie działania wykonuje on zapytanie `SELECT count(*) FROM queue`, które zlicza ilość wpisów w tabeli, każdy wpis odpowiada pojedynczemu elementowi w kolejce. Wartość następnie jest zapisywana w formacie CSV do pliku razem z czasem wykonania pomiaru.

Skrypt został uruchomiony z następującymi parametrami.

```
python3 measure.py state_storage/stage.queue state_storage/queue.log
```

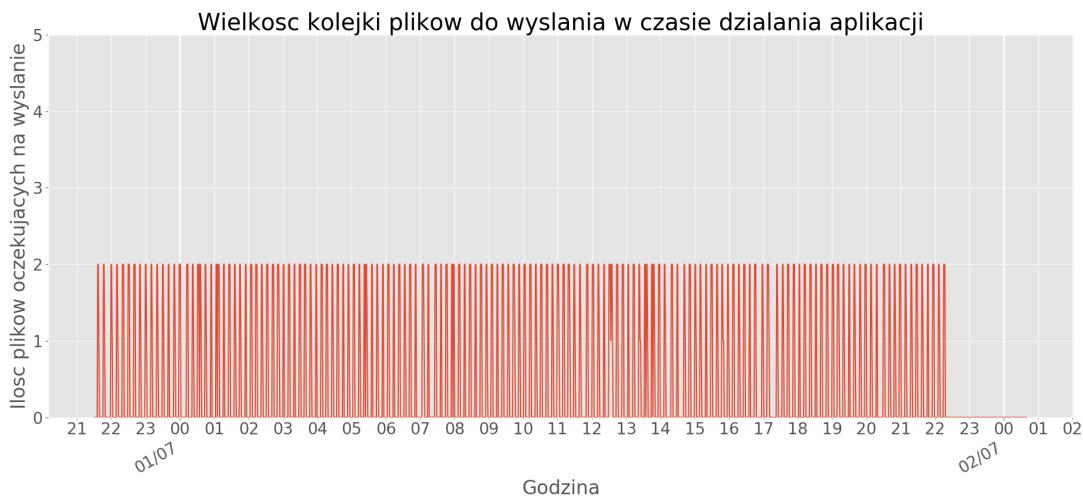
```

1 import sqlite3
2 import time
3 import csv
4 import sys
5
6
7 def main():
8     while True:
9         with sqlite3.connect(sys.argv[1]) as con, open(sys.argv[2], 'a') as fd:
10             ((count,)) = con.execute("SELECT count(*) FROM queue").fetchall()
11             writer = csv.writer(fd, delimiter=',')
12             writer.writerow([time.strftime("%Y/%m/%d %H:%M:%S"), count])
13             time.sleep(10)
14
15 main()
16

```

Rys. 48. Skrypt monitorujący stan kolejki

Wynik przesłania zestawu danych testowych przedstawia rysunek 49. Ilość punktów pomiarowych ograniczono w celu uwidocznienia zmian w długości kolejki. Nadawanie danych rozpoczęto około godziny 21.30 dnia 01.07.2018. Długość kolejki w czasie symulacji nie wyniosła więcej niż 2 plików. Test trwał około 24 godzin. Przepustowość na poziomie 55 kilobajtów jest wystarczająca do bieżącego przesyłu danych pomiarowych.



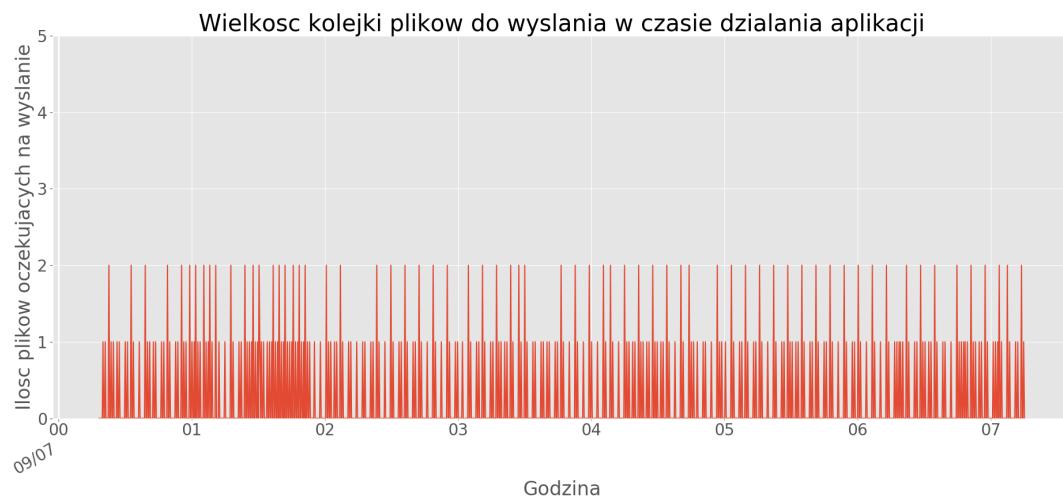
Rys. 49. Długość kolejki plików do wysłania w ujęciu godzinowym, przepustowość łącza 55 kB/s

Wynik testu przy przepustowości 15 kB/s przedstawia rysunek 50. Po 5 godzinach działania kolejka urosła do 350 plików. Prędkość ta jest niewystarczająca do nadawania bez opóźnień.



Rys. 50. Długość kolejki plików do wysłania w ujęciu godzinowym, przepustowość łącza 15 kB/s

Wynik testu pary (wavelet, x264) przedstawia rysunek 51. Symulację przeprowadzono, przy przepustowości łącza 12 kB/s. Pomimo najwyższej z badanych przepustowości nadawanie odbywa się na bieżąco. Duża ilość wartości zero widocznych na wykresie sugeruje, że schemat ten może pracować również przy niższych przepustowościach.



Rys. 51. Długość kolejki plików do wysłania w ujęciu godzinowym, kompresja (wavelet, x264), przepustowość łącza 12 kB/s

## 4.2. Zachowanie systemu w warunkach roboczych

W warunkach symulacyjnych pliki nadawane były co stały interwał, a przepustowość sieci posiadała stałą wartość. Odbiega to od warunków rzeczywistych, w których oba te parametry są zmienne. W celu zbadania zachowania aplikacji przeprowadzono próbne rozruchy na środowisku docelowym.

Konfiguracje komponentów Sender i Receiver przedstawiają rysunki 52 i 53. Dekompresję danych pominięto w celu zaoszczędzenia przestrzeni dyskowej na maszynie docelowej. Komponent Receiver nasłuchiwał pod adresem 10.8.0.42 na porcie 50123, wartość ta została

skonfigurowana również w klauzuli „server” komponentu Sender. Adres komponentu LabVIEW ustawiony został na 127.0.0.1 (interfejs lokalny) port 6341. Komponent Sender po zerwaniu połączenia oczekuje 30 sekund przed próbą ponownego nawiązania.

```

1 {
2   "server": {
3     "host": "10.8.0.42",
4     "port": 50123,
5     "retry_time": 30,
6     "processor": "compression"
7   },
8   "provider": {
9     "host": "127.0.0.1",
10    "port": 6341,
11    "extensions": ["lvm", "avi"],
12    "retry_time": 30,
13    "separator": "\n"
14  },
15  "storage_root": "../pomiary",
16  "stage_queue_path": "state_storage/stage.queue",
17  "cache_path": "state_storage/client.cache",
18
19  "compression": {
20    "avi": "x264",
21    "lvm": "bzip2",
22    "mp4": null
23  }
24 }
```

Rys. 52. Konfiguracja komponentu Sender

```

1 {
2   "host": "10.8.0.42",
3   "port": 50123,
4   "storage_root": "/home/tomaszjonak/server_storage",
5
6   "decompression": {}
7 }
```

Rys. 53. Konfiguracja komponentu Receiver

Monitorowanie stanu kolejki przeprowadzono za pomocą skryptu przedstawionego na rysunku 54. Uzupełnia on pomiar o wielkość pliku kolejki. Wielkość tą zmierzono w celu ustalenia jak długość kolejki wpływa na zużycie przestrzeni dyskowej przez komponent Sender.

```

1 import sqlite3
2 import time
3 import csv
4 import sys
5 import pathlib as pl
6
7
8 def main():
9     queue_path = pl.Path(sys.argv[1])
10    log_path = pl.Path(sys.argv[2])
11    while True:
12        try:
13            with sqlite3.connect(str(queue_path)) as con, log_path.open('a') as fd:
14                ((count,),) = con.execute("SELECT count(*) FROM queue").fetchall()
15                writer = csv.writer(fd, delimiter=',')
16                writer.writerow([time.strftime("%Y/%m/%d %H:%M:%S"), count, queue_path.stat().st_size])
17        except Exception as e:
18            print(e)
19        finally:
20            time.sleep(float(sys.argv[3]))
21
22 main()
```

Rys. 54. Skrypt monitorujący długość kolejki i wielkość pliku kolejki

Wynik monitorowania stanu kolejki w warunkach roboczych przedstawia rysunek 55. Pomiar uruchomiono o godzinie 20.40 dnia 04.07.2018. Do godziny piątej pliki nadawane były

bez opóźnień. Od godziny piątej do dwudziestej pierwszej kolejka rośnie, przypuszcza się, że spowodowane jest to zwiększym natężeniem ruchu drogowego. Pomiędzy północą a piątą rano 05.07 długość kolejki spada.

Wykres przyjmuje specyficzny kształt pomiędzy godziną siódmą a dziewiątą 05.07.2018. W okolicach godziny 6.40 komputer w miejscowości Gardawice uruchomił się ponownie, komponent Sender został wyłączony. Manualne uruchomienie komponentu nastąpiło o godzinie 8.40. Za skok ilości plików w kolejce odpowiedzialny jest wątek SyncThread, po nawiązaniu połączenia z komponentem LabVIEW przeszukał on dysk twardy i uzupełnił kolejkę o brakujące pliki.

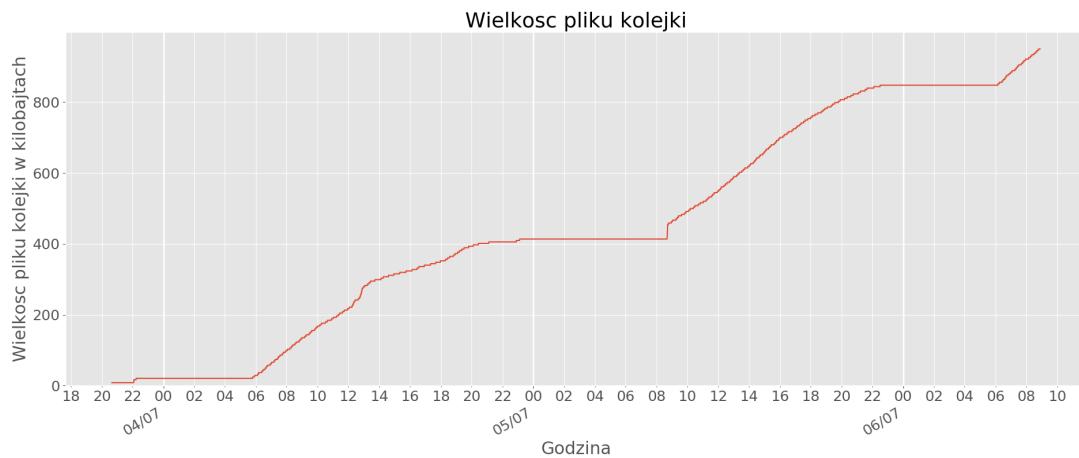
Na podstawie kształtu wykresu można przypuszczać tendencję wzrostową długości kolejki. Spadek w godzinach nocnych nie kompensuje wzrostu długości w ciągu dnia. Wybrany schemat kompresji okazał się niewystarczający, a pomiar przepustowości łącza zbyt krótki aby otrzymać realistyczne dane.



Rys. 55. Długość kolejki plików do wysłania w ujęciu godzinowym

Trend wzrostowy długości kolejki może doprowadzić do przepełnienia dysku, jest to sytuacja niepożądana, w przypadku zbyt szybkiego zajmowania przestrzeni dyskowej zastosowanie komponentu Sender przestaje przynosić zysk. Jednym z celów działania systemu przesyłu jest zapobiegnięcie przepełnieniu.

Wykres wielkości pliku kolejki w czasie prowadzonego pomiaru przedstawia rysunek 56. Wielkość pliku wzrasta wraz ze zwiększaniem się ilości plików w kolejce. Interesującym jest fakt, że w momencie spadku długości kolejki wielkość pliku nie zmniejsza się. Sytuacja ta jest widoczna na wykresie pomiędzy godziną 23 dnia 04.07, a 9 dnia 05.07. Przypuszczalnie silnik bazy danych SQLite, wykorzystany w implementacji kolejki zakłada, że wielkość bazy będzie rosnąć i zwalnianie miejsca nie przynosi korzyści.



Rys. 56. Wielkość pliku kolejki w ujęciu godzinowym



## 5. Podsumowanie

W ramach pracy przeprowadzono analizę pomiarów generowanych przez stanowisko pomiarowe. Oszacowano przepustowość łącza, za pomocą którego stanowisko komunikuje się z uczelnią, a także dobrano metody kompresji poszczególnych danych pomiarowych na podstawie analizy sprawności. Zaprojektowany został system przesyłowy, bazując na uproszczonej wersji protokołu HTTP, przeprowadzono analizę możliwych awarii oraz zaproponowano metody ich obsługi. Działanie aplikacji zostało zweryfikowane w środowisku symulacyjnym a następnie wdrożone. Ponadto zaimplementowano w języku Python oraz zweryfikowano działanie algorytmu falkowego.

### Kierunki dalszego rozwoju

W trakcie prac nad projektem zauważono, że model współbieżności oparty na wątkach nie jest dostosowany do napotkanego problemu. Przypuszcza się, że skuteczniejsze wykorzystanie zasobów sieciowych można uzyskać stosując podejście asynchroniczne. Jest ono oferowane przez bibliotekę asyncio języka Python oraz język Go.

Kompresja jest jedną z metod przetwarzania strumieni danych. Proponowane jest uogólnienie tego podejścia i umożliwienie zintegrowania aplikacji z dowolnym „pośrednim” programem przetwarzającym dane. W ten sposób można umożliwić wykonanie analizy profili magnetycznych na stanowisku pomiarowym, a wysyłać jedynie jej wyniki.

Kompleksowa metoda synchronizacji jest wymagana, ponieważ w katalogu pomiarów mogą znajdować się pliki już wysłane. Wprowadzenie na stałe do aplikacji mechanizmu usuwania nadanych plików likwiduje ten problem. W takim podejściu każdy plik, który znajduje się w katalogu pomiarów wymaga wysłania.

Symulacje wskazują, że zastosowanie algorytmu falkowego na stanowisku pomiarowym umożliwi bieżący przesył danych. W celu weryfikacji tej hipotezy, należy przeprowadzić próbę instalacji biblioteki PyWavelets z uprawnieniami administratorskimi, bądź zaimplementować algorytm z użyciem innej biblioteki.

## Wnioski

Stanowisko pomiarowe przeprowadza średnio dziennie 5.6 tysiąca pomiarów. Wynikiem tego jest ponad 11 tysięcy plików z pomiarami o łącznej wadzie 9 gigabajtów. Wielkość se-

kwencji wideo zależy od pory dnia. Przypuszczalnie jest to spowodowane zmianami naświetlenia stanowiska pomiarowego. Łącze sieciowe zapewniające transfer danych na uczelnię posiada przepustowość na poziomie zdecydowanie niższym niż 58 kilobajtów na sekundę, którą zmierzono w trakcie pracy.

Zbadano trzy metody kompresji bezstratnej oraz dwie stratne. Kompresja wideo z użyciem standardu H.264 przyniosła bardzo dobre rezultaty. Współczynnik kompresji oferowany przez algorytm falkowy deklasuje pozostałe proponowane metody.

W ramach pracy zaprojektowano dwie aplikacje, które w kooperacji realizują ciągły i bezstratny przesył danych pomiarowych. System przesyłowy został uruchomiony na środowisku pomiarowym. Obserwacja jego działania prowadzi do wniosku, że obecnie zaproponowany schemat kompresji nie jest wystarczający, aby realizować przesył danych na bieżąco.

Osiągnięcie tego celu wymaga dalszych prac nad sposobem przygotowania danych do nadawania. Proponowane jest przeprowadzenie analizy sygnałowej w miejsce kompresji i nadawanie jej wyników.

# Bibliografia

- [1] J. Postel, “Transmission control protocol,” Internet Requests for Comments, RFC Editor, STD 7, September 1981, <http://www.rfc-editor.org/rfc/rfc793.txt>. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc793.txt>
- [2] D. Salomon G. Motta, *Handbook of Data Compression*, 5th ed. Springer Publishing Company, Incorporated, 2009.
- [3] D. Huffman, “Canonical forms for information-lossless finite-state logical machines,” *IRE Transactions on Circuit Theory*, vol. 6, no. 5, s. 41–59, May 1959.
- [4] G. K. Wallace, “The jpeg still picture compression standard,” *Commun. ACM*, vol. 34, no. 4, s. 30–44, Apr. 1991. [Online]. Available: <http://doi.acm.org/10.1145/103085.103089>
- [5] I. E. Richardson, *The H.264 Advanced Video Compression Standard*, 2nd ed. Wiley Publishing, 2010.
- [6] Z. Marszałek, “Model polowy czujnika pętlowego indukcyjnego,” s. 19–26, 01 2011.
- [7] “iperf/iperf3,” <https://iperf.fr/>, data dostepu: 2018-06-30.
- [8] G. Manzini, “The burrows-wheeler transform: Theory and practice,” in *Mathematical Foundations of Computer Science 1999*, M. Kutyłowski, L. Pacholski, T. Wierzbicki, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, s. 34–47.
- [9] “Lzma compression,” <https://dxr.mozilla.org/mozilla-central/source/other-licenses/7zstub/src/DOC/lzma.txt>, data dostepu: 2018-05-20.
- [10] D. Salomon, *Data compression: the complete reference*. Springer Science & Business Media, 2004.
- [11] ——, *Data compression: the complete reference*. Springer Science & Business Media, 2004.
- [12] “Linux programmer’s manual - inotify(7),” <http://man7.org/linux/man-pages/man7/inotify.7.html>, data dostepu: 2018-05-13.

- [13] “Readdirorychangesw function,” [https://msdn.microsoft.com/en-us/library/aa365465\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa365465(v=vs.85).aspx), data dostepu: 2018-05-13.
- [14] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, T. Berners-Lee, “Hypertext transfer protocol – http/1.1,” Internet Requests for Comments, RFC Editor, RFC 2616, June 1999, <http://www.rfc-editor.org/rfc/rfc2616.txt>. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2616.txt>