**SPRAWOZDANIE**

Z PRACY

pod tytułem

„Opracowanie oprogramowania umożliwiającego uzyskanie w czasie rzeczywistym danych z rejestratora SSR-500 podczas prac obsługowych”

Opracował:

…………………………………

mgr inż. Patryk Niedbała

**WARSZAWA 2018**

[celowo pozostawiono niezapisane]

# Spis treści

[1. PODSTAWA PRAWNA 5](#_Toc525814384)

[2. CEL PRACY 5](#_Toc525814385)

[3. ZAŁOŻENIA DO PRACY 5](#_Toc525814386)

[4. PRZEBIEG I WYNIKI PRACY 6](#_Toc525814387)

[4.1. PODSTAWOWE FUNKCJONALNOŚCI APLIKACJI 6](#_Toc525814388)

[4.2. TECHNOLOGIE ZASTOSOWANE PRZY OPRACOWANIU OPROGRAMOWANIA 7](#_Toc525814389)

[4.3. PROCEDURA ODBIORU DANYCH 7](#_Toc525814390)

[4.4. PROCEDURA EKSTARCJI DANYCH 8](#_Toc525814391)

[4.5. ZASADA DZIAŁANIA APLIKACJI 10](#_Toc525814392)

[4.6. OPIS KOMPONENTOW SYSTEMOW 19](#_Toc525814393)

[4.7. PRZYSZŁE MODYFIKACJE OPROGRAMOWANIA 21](#_Toc525814394)

[5. PODSUMOWANIE I WNIOSKI 21](#_Toc525814395)

[6. ZAŁĄCZNIKI 22](#_Toc525814396)

[6.1. A. Kod źródłowy klasy CaptureForm 22](#_Toc525814397)

[6.2. B. Kod źródłowy klasy Chartform 31](#_Toc525814398)

[6.3. C. Kod źródłowy klasy Converter 40](#_Toc525814399)

[celowo pozostawiono niezapisane]

# PODSTAWA PRAWNA

Podstawą do wykonania pracy są postanowienia przedstawione w umowie nr 483/18 zawartej pomiędzy Dyrektorem Instytutu Technicznego Wojsk Lotniczych, a p. Patrykiem Niedbałą

# CEL PRACY

Celem pracy było opracowanie oprogramowania umożliwiającego uzyskanie w czasie rzeczywistym danych z rejestratora SSR-500 podczas prac obsługowych.

# ZAŁOŻENIA DO PRACY

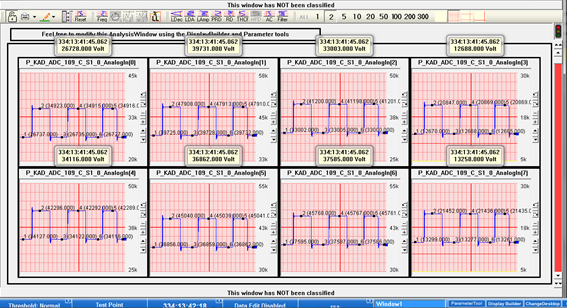
Od roku 2015 w ramach systemu SMO dla samolotów Su-22 UM3K dokonywana jest akwizycja i rejestracja danych – sygnałów z czujników tensometrycznych oraz parametrów lotu. Dane rejestrowane są przy pomocy systemu ACRA SSR-500. Wyniki rejestracji zapisywane są na karcie SD. Dane z karty SD przekazywane są okresowo do ITWL w celu przetwarzania i analizy. Dane z systemu SMO zawarte są w postaci plików PCAP (rozszerzenie .cap).

System SMO ma za zadanie wspomóc procedurę określania zużycia konstrukcji monitorowanych statków powietrznych. Podstawą tej procedury jest wyliczanie zużycia zmęczeniowego na podstawie zapisów zarejestrowanych przez zamontowany na samolotach tensometryczny system pomiarowy. Analizy zużycia dokonuje się na podstawie zapisów sygnału tensometrów oraz parametrów lotu uzyskanych z systemu GPS.

Weryfikacja poprawności działania instalacji Systemu Monitorowania Obciążeń na samolotach Su-22UM3K jest sprawdzana m.in. w czasie obsług okresowych po 100 godzinach lotu na podstawie metodyki MB-14.31.2015 dotyczącej okresowych sprawdzeń metrologicznych elementów systemu monitorowania obciążeń. W jej trakcie dokonywane jest również sprawdzenie poprawności działania czujników tensometrycznych zainstalowanych na samolocie. W celu sprawdzenia poprawności ich działania, należy podłączyć pulpit PWK-1, który zapewnia włączenie rejestratora SSR‑500 bez potrzeby włączania rejestratora pokładowego oraz włączenie obwodów kalibracji wszystkich kanałów pomiarowych rejestratora w module tensometrycznym MT-1. Następnie wykorzystując program „GS Works 8” na dedykowanym komputerze służącym do wizualizacji danych z SSR-500 w czasie rzeczywistym dokonywane jest sprawdzenie poprawności działania czujników tensometrycznych. Okno programu GS Works 8 przedstawiono na Rys.1

Ze względu na znaczące koszty, ograniczenia licencyjne oraz niewielkie możliwości modyfikacji oprogramowania GS Works 8, podjęto decyzje o rozpoczęciu prac nad oprogramowaniem, które w przyszłości umożliwi zastąpienie stosowanej do tej pory aplikacji w trakcie wykonywanych przeglądów. Dlatego w celu zastąpienia dotychczas stosowanego oprogramowania należało opracować aplikację, która umożliwi uzyskanie w czasie rzeczywistym danych z rejestrator SSR-500 w czasie prac obsługowych. W wyniku automatycznej analizy uzyskiwane są wyekstrahowane sygnały dogodne do dalszej analizy.

Dodatkowym pomocniczym wynikiem jest wykonywany w czasie rzeczywistym zapis graficzny dla każdego z wyekstrahowanych sygnałów. Ekstrakcja odbywa się na podstawie obowiązującego w danym momencie pliku konfiguracyjnego – oprogramowanie dopuszcza jego zmianę. Opracowana aplikacja powinna również spełniać podstawowe funkcjonalności umożlwiające sprawdzenie poprawności działania czujników tensometrycznych na samolocie i poprawne wykonanie prac obsługowych.



Rys 1 Widok okna programu GS Works 8

# PRZEBIEG I WYNIKI PRACY

## PODSTAWOWE FUNKCJONALNOŚCI APLIKACJI

Na podstawie metodyki MB-14.31.2015 dotyczącej okresowych sprawdzeń metrologicznych elementów systemu monitorowania obciążeń samolotów Su-22UM3K oraz ogólnych postanowień dotyczących oprogramowania opracowano podstawowe funkcjonalności aplikacji:

* Odbiór pakietów IPv4 zgodnych z protokołem UDP wysyłanych z rejestratora SSR-500
* Ekstrakcja pakietów na wartości kanałów sygnałowych z rozróżnieniem oddzielnych kanałów i zachowaniem ciągłości sygnału na podstawie wskazanego pliku XML
* Opcja skalowania wyekstrahowanych wartości na podstawie wskazanej tablicy skalowań
* Podgląd odebranych pakietów umożlwiający zweryfikowanie poprawności odbieranych pakietów
* Zapis odbieranych pakietów do pliku
* Zapis wyekstrahowanych danych w postaci sygnałów do pliku
* Wizualizacja wybranych kanałów sygnałowych
* Możliwość dopasowania parametrów wykresu do aktualnie odbieranych danych
* Wyświetlanie dowolnej wartości sygnału w postaci liczbowej
* Wizualizacja wyekstrahowanych sygnałów na podstawie wskazanego pliku .cap

## TECHNOLOGIE ZASTOSOWANE PRZY OPRACOWANIU OPROGRAMOWANIA

Oprogramowanie opracowano przy użyciu języka programowania C# .NET z wykorzystaniem środowiska Visual Studio 2017. Oprogramowanie pracuje w systemie operacyjnym MS WINDOWS. Oprogramowanie posiada otwartą architekturę – jego podstawowa wersja zawarta jest w projekcie (solucji) programu Visual Studio – rozszerzenie \*.sln.

Oprogramowanie ma budowę modułową podstawowe moduły zawierające różne części funkcjonalności zawarte są w oddzielnych projektach Visual Studio (pliki \*.csproj). Każdy z projektów kompiluje się do oddzielnej biblioteki dll. Dodatkowo, oprogramowanie posiada prosty moduł graficzny GUI, zwarty w oddzielnym projekcie \*.csproj. Aplikacja zawierająca interfejs graficzny została skompilowana do pliku wykonywalnego LiveCaptureForm.exe, który wraz z dołączonymi modułami w postaci bibliotek dll umożliwia uruchomienie aplikacji na komputerze docelowym. Skompilowana aplikacja oraz kody źródłowe oprogramowania zawiera Załącznik 1.

Do opracowania niektórych algorytmów wykorzystano zewnętrzne biblioteki - SharpPcap (do odczytu ramek danych). Do obsługi i odświeżania interfejsu graficznego wykorzystano mechanizm zdarzeń (Events) będący częścią platformy .NET. Kody źródłowe zawarto w załącznikach A,B i C oraz na dołączonej płycie DVD.

## PROCEDURA ODBIORU DANYCH

W systemie SMO dane rejestrowane są przez system rejestracji ACRA SSR-500 produkcji firmy Curtiss-Wright. Elementem systemu odpowiedzialnym za rejestracje analogowych sygnałów tensometrycznych jest karta rozszerzeń KAD/ADC/109/S1. SSR-500 to kompaktowe, lekkie, wytrzymałe rejestratory ze zintegrowanym gromadzeniem danych. Dostarczony jest port Ethernet na panelu tylnym do konfiguracji i monitorowania w czasie rzeczywistym. Komunikacja z rejestratorem za pomocą komputera PC odbywa się poprzez wejście Ethernet. Urządzenie pracuje w trybie multicast w której informacja pochodząca od jednego nadawcy, dociera do dowolnej liczby odbiorców. Dane wyjściowe mają postać serii plików .cap zawierających sekwencje kolejno zapisanych ramek danych (w postaci strumienia bitów), każda ramka posiada sygnaturę czasową. Model danych opiera się na architekturze TCP/IP – warstwa łączenia danych to MAC (Ethernet) na którą składają się pakiety IPv4 (w warstwie transportowej – protokół UDP). UDP przesyła pakiety od źródła do odbiorcy, nie wymagając potwierdzenia i retransmisji, gdy pakiety zostaną zgubione. Strumienie multimedialne muszą być przesyłane w czasie rzeczywistym, nie ma więc czasu na retransmisje. Każdy pakiet stracony bądź opóźniony oznacza problemy z jakością odbieranego strumienia danych. Dlatego aplikacje odbierające ruch muszą dysponować technologią buforowania.

Przechwytywanie pakietów to proces zbierania wszystkich pakietów danych, które przechodzą przez dany interfejs sieciowy. Przechwytywanie pakietów sieciowych w aplikacjach jest możliwością, która pozwala pisać aplikacje stanowiące monitory sieci, analizatory pakietów i narzędzia bezpieczeństwa.

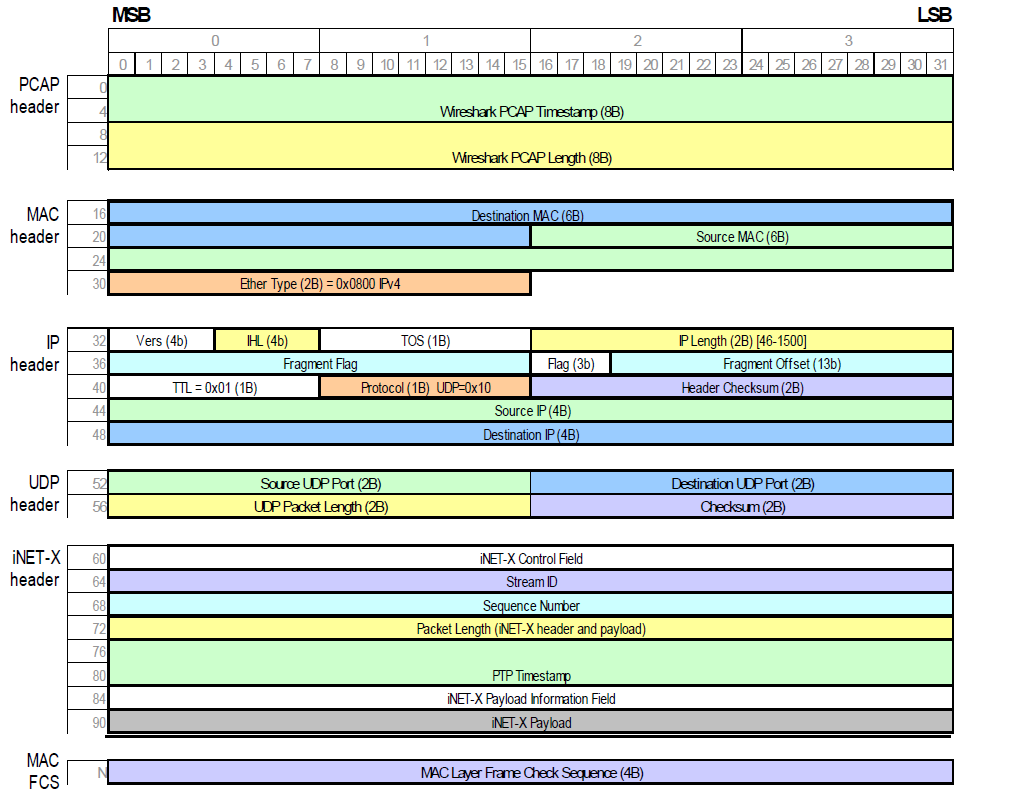
Biblioteka *WinPcap* dla Windows to najczęściej używane sterownik przechwytywania pakietów, które zapewniają interfejs API do monitorowania sieci niskiego poziomu. Wśród aplikacji używających *WinPcap, jako* podsystemu przechwytywania pakietów są słynne TcpDump i Wireshark. Biblioteka *WinPcap* dostarcza narzędzia umożlwiające przechwytywanie pakietów oraz ich analizę, dlatego została ona wykorzystana w opracowanym oprogramowaniu. Odpowiednikiem dla środowiska .NET jest biblioteka *SharpPcap* .NET zapewniająca rozwiązania zawarte w bibliotece *WinPcap* dostępne dla aplikacji .NET.

W celu rozpoczęcia przechwytywania pakietów należy zacząć od wyboru urządzenia na komputerze odpowiedzialnego za odbiór pakietów, który wybieramy z listy dołączonych kart sieciowych. *SharpPcap* zapewnia w tym celu klasę *CaptureDeviceList.* Ta klasa jest pojedynczą instancją przechowującą buforowaną listę kart sieciowych typu *ICaptureDevice.* W szczególności właściwości *Name* i *Description* zawierają odpowiednio nazwę i czytelny opis odpowiedniego urządzenia. Wybrane urządzenie zostaje następnie skonfigurowane odpowiednio do monitorowania sieci i posłuży do pozyskiwania przechwyconych pakietów. Po skonfigurowaniu połączenia i dobraniu odpowiednich filtrów, umożlwiającą optymalizacje przechwytywanych pakietów, można rozpocząć wyszukiwanie pakietów.

Napisana aplikacja powinna zapewnić możliwość odbioru w czasie rzeczywistym, tak by wszystkie wysyłane pakiety były zapisywane bez strat kolejnych pakietów. W tym celu do odbioru danych został wykorzystany obiekt typu *EventHandler* odpowiadający za przechwycenie kolejnych pakietów i zbuforowaniu ich poprzez przekazanie do utworzonej kolejki pakietów. Pakiety znajdujące się w kolejce zostaną podane ekstrakcji w osobnym wątku. Takie rozwiązaniu umożliwia jednoczesne przetwarzanie zbuforowanych pakietów oraz nasłuchiwanie kolejnych pakietów pojawiających się na sieci, co minimalizuje straty na odbiorze.

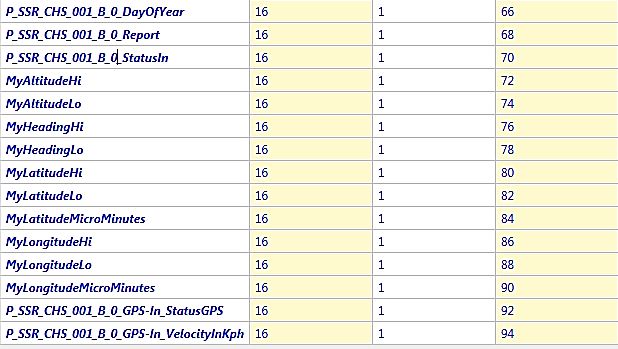
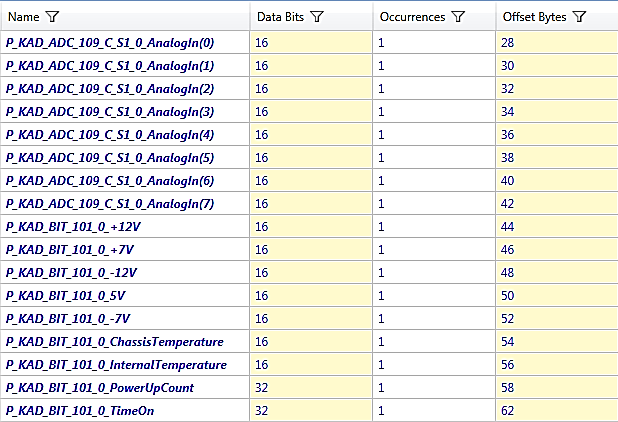
## PROCEDURA EKSTRAKCJI DANYCH

Model danych opiera się na architekturze TCP/IP – warstwa łączenia danych to MAC (Ethernet) na którą składają się pakiety IPv4 (w warstwie transportowej – protokół UDP). W warstwie aplikacji ramki wykorzystany jest format firmy ACRA control – ramka iNET‑X, wersja „placed”. W ramce zawarte są sumy kontrolne, oraz sygnatury czasowe – w tym podstawa czasu pochodząca z systemu GPS oraz z samego rejestratora. Konwersja odebranych pakietów polega na sekwencyjnym przejściu przez kolejne formaty, t.j – pakiet Ethernet przekształcany jest na pakiet IPv4, pakiet IPv4 przekształcany jest na pakiet UDP, pakiet UDP jest ostatecznie przekształcony na format iNET-X placed. Dla poprawnie pozyskanej ramki iNET-X placed, dokonywana jest ekstrakcja próbek sygnałów ze strumienia bitów zawartego w warstwie danych pakietu. Odpowiedni sygnały pozyskiwany z paczki bitów, której położenie (offset) i rozmiar określony jest w pliku konfiguracyjnym .xml.



Rys 2 Ramka ACRA iNET-X zapisana w pliku PCAP (.cap)

Do ekstrackji ramek z pakietów wykorzystano biblioteki opracowane w ramach pracy „Opracowanie oprogramowania do automatycznej analizy poprawności danych rejestrowanych w Systemie Monitorowania Obciążeń samolotów Su-22 UM3K”. Oprogramowanie składa się z kilku modułów. Do ekstrakcji sygnałów z pakietów wykorzystano klasę *Convert* zawartą w module *Smo.Startup* opracowaną w celu przetwarzania pakietów na wartości sygnałów. Na podstawie dostarczonych plików konfiguracyjnych xml opracowywana jest lista parametrów (sygnałów) do ekstrakcji – rozpatrywane są dwa pliki xml – jeden ze zmienną konfiguracją ramki systemu SSR-500, a drugi (InstrumentSettings.xml) ze stałą konfiguracją rejestrów systemu ACRA SSR-500 (konfiguracja wbudowanych parametrów) – na podstawie tych plików uzyskiwane są odpowiednie offsety konieczne do zdekodowania zapisów w każdej z ramek danych odpowiadających poszczególnym chwilom czasowym (próbkom sygnału). Wynikiem przetworzenia wszystkich ramek jest zbiór czasowych sygnałów dla każdej z rejestrowanych w systemie wielkości – są to dane takie jak parametry lotu, sygnały tensometryczne, wewnętrzna temperatura, sumy kontrolne, dwustanowe wskaźniki błędów, i inne. Przykładową strukture wyekstrahowanych danych przedstawiono na Rys 2.



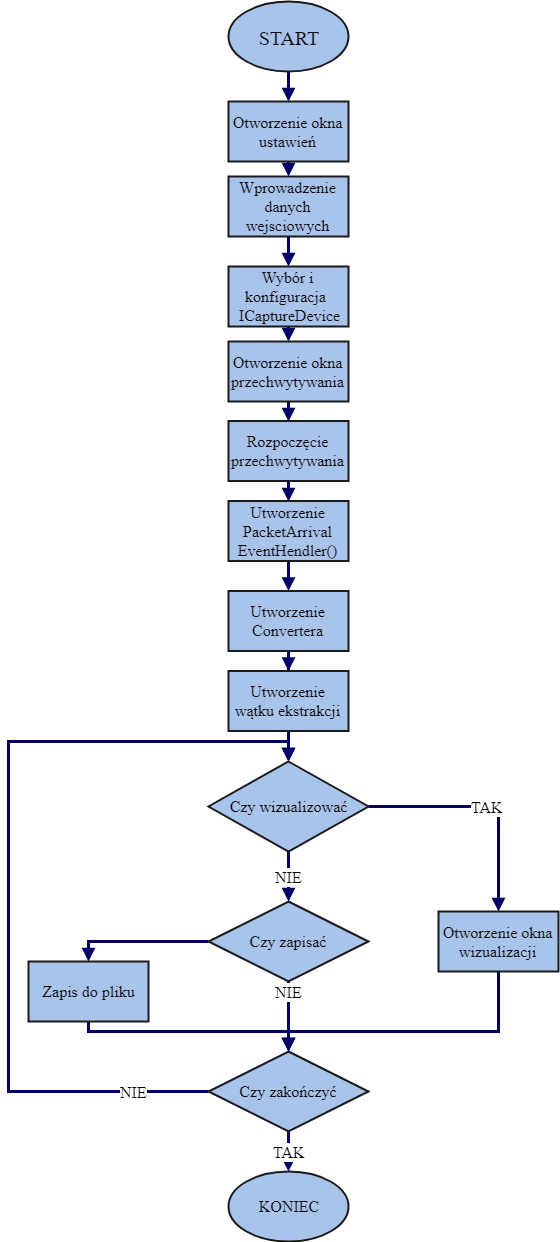
Rys 3 Przykładową strukture wyekstrahowanych danych

## ZASADA DZIAŁANIA APLIKACJI

Oprogramowanie składa się z kilku modułów zawierających opracowaną aplikacje z interfejsem GUI, wykorzystywane moduły oraz biblioteki do ekstrakcji i przechwytywania pakietów przedstawionych w poprzednich rozdziałach. Danymi wejściowymi dla aplikacji są numery portu i IP rejestratora, nazwa urządzenia sieciowego wykorzystywanego do przechwytywania pakietów, schematy konfiguracyjne w formacie \*.xml (2 pliki), oraz tabele skalowania w formacie \*.tsv. Ogólny algorytm działania aplikacji przedstawiono na Rys 4.

Główne operacje wykonywane przez oprogramowanie to:

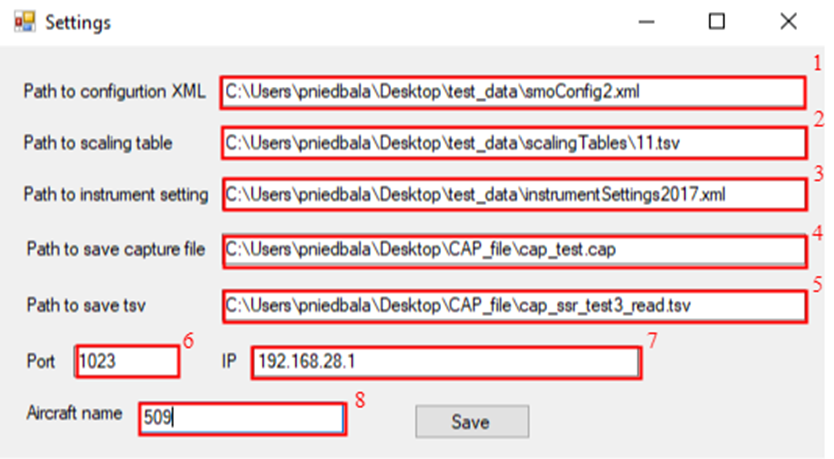
* Przechwytywanie pakietów UDP ramka po ramce w czasie rzeczywistym
* Ekstrakcja pakietów na wartości sygnałów
* Wizualizacja graficzna odebranych danych
* Zapis i odczyt parametrów z plików .cap



* Rys 4 Ogólny algorytm działania aplikacji

Po uruchomieniu się aplikacji w pierwszym kroku zostaje otworzone okno ustawień, którego wygląd przedstawiono na Rys 5 wraz z oznaczonymi kolejnymi oknami tekstowymi, zawierającymi ścieżki do wprowadzenia danych wejściowych dla aplikacji z podanymi przykładowymi wartościami. W kolejnych oknach należy wpisać:

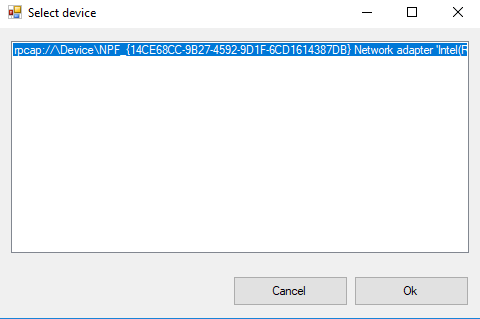
1. Ścieżkę do pliku konfiguracyjnego w formacie \*.xml ze zmienną konfiguracją ramki systemu SSR-500
2. Ścieżkę do tablicy skalowań w formacie \*tsv.
3. Ścieżkę do pliku konfiguracyjnego w formacie \*.xml dotyczących ustawień urządzenia ze stałą konfiguracją rejestrów systemu ACRA SSR-500
4. Ścieżkę i nazwę pliku, do którego zostaną zapisane przechwycone pakiety
5. Ścieżkę i nazwę pliku, do którego zostaną zapisane wartości sygnałowe wyekstrahowanych pakietów.
6. Numer portu używanego do transmisji przez rejestrator
7. Numer IP używany przez rejestrator
8. Numer samolotu, z którego rejestrowane są dane



Rys 5 Okno ustawień

Ustawienia domyślnie są pobierane z pliku przechowującego poprzednie wprowadzone wartości, ma to na celu skrócenie czasu wprowadzania ustawień przy każdym uruchomieniu aplikacji. Po podaniu odpowiednich ścieżek i wartości, zmiany zatwierdzamy przyciskiem „Save” w celu zapisania wprowadzonych zmian do pliku. Po wyjściu z okna ustawień program pobiera ze wskazanych plików odpowiednie dane stanowiące dane wejściowe dla poprawnego działania aplikacji

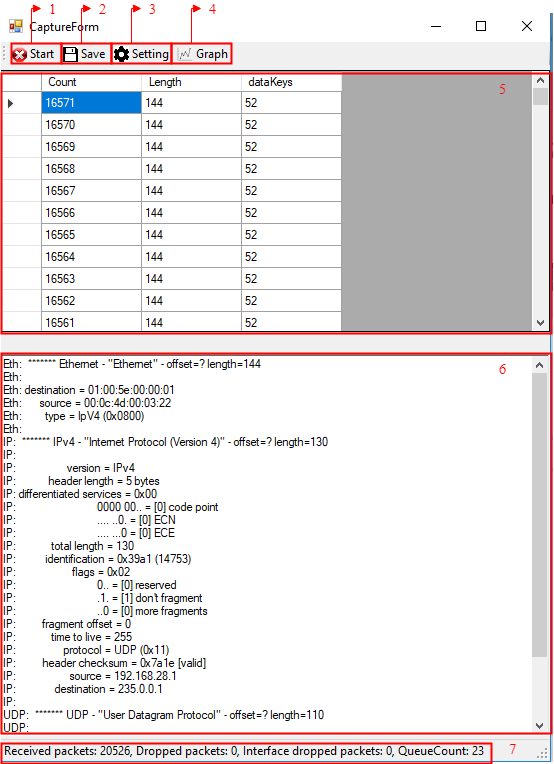
W kolejnym kroku zostaje otwarte okno wyboru urządzenia *ICaptureDevice*, które posłuży do przechwytywania nadchodzących pakietów. Wygląd okna przedstawiono na Rys 6. Głównym elementem formularza Windows jest okno tekstowe, na którym zostają rozpisane wszystkie zlokalizowane karty sieciowe podłączone do komputera. Użytkownik ma za zadnie wybrać urządzenie, które umożliwi komunikacje z rejestratorem i posłuży w dalszej części do przechwytywania pakietów. Po dokonaniu wyboru zatwierdzamy decyzje przyciskiem „OK”. Następnie wybrana karta zostanie odpowiednio skonfigurowana, jako urządzenie *ICaptureDevice* służące do odbioru danych z rejestratora.



Rys 6 Okno wyboru ICaptureDevice

Po wybraniu i skonfigurowaniu urządzenia *ICaptureDevice* zostaje otworzone okno przechwytywania, którego wygląd przedstawiono na Rys 7. Na rysunku oznaczono również główne elementy formularza Windows oznaczone kolejnymi numerami:

1. Przycisk rozpoczęcia przechwytywania
2. Przycisk zapisu
3. Przycisk ustawień
4. Przycisk okna wizualizacji
5. Tabela odebranych pakietów
6. Okno tekstowe z parametrami wskazanych pakietów
7. Wskaźniki ilości odebranych pakietów, straconych czy znajdujących się aktualnie w kolejce



Rys 7 Okno przechwytywania

Po skonfigurowaniu połączenia z *ICaptureDevice* i dobraniu odpowiednich filtrów umożlwiającą optymalizacje przechwytywanych pakietów można rozpocząć wyszukiwanie pakietów. Napisana aplikacja powinna zapewnić możliwość odbioru w czasie rzeczywistym, tak aby wysyłane pakiety były zapisywane bez strat kolejnych pakietów. W tym celu do odbioru nadesłanych pakietów został wykorzystany handler *PacketArtivalEventHandler(),* w którym zostały zaimplementowane funkcje odbioru nadesłanych pakietów. Wszystkie przechwycone pakiety zostają zbuforowane poprzez przekazanie do utworzonej kolejki pakietów. Takie rozwiązaniu umożliwia jednoczesne przetwarzanie zbuforowanych pakietów oraz nasłuchiwanie kolejnych pakietów pojawiających się na sieci co minimalizuje straty na odbiorze.

Następnie zostaje wywołany konstruktor klasy *Converter* zawierający metody, które umożliwią ekstrakcje pakietów na wartości sygnałowe. Danymi wejściowymi dla konstruktora są odczytane pliki konfiguracyjne z rozszerzeniem \*xml oraz tablica skalowań. Na podstawie tych plików uzyskiwane są odpowiednie offsety konieczne do zdekodowania zapisów w każdej z ramek danych odpowiadających poszczególnym chwilom czasowym (próbkom sygnału). Utworzony obiekt *Converter* zostaje następnie przekazany do nowo utworzonego wątku, który będzie odpowiedzialny za ekstrakcje przechwyconych pakietów.

W nowo utworzonym wątku następuje na podstawie metody *DecodePacket* klasy *Converter* dekodowanie kolejnych pakietów, które aktualnie zostały zbuforowane w kolejce. Wspomniana metoda wykorzystuje m. in. procedury zaimplementowane w klasie *Reader* modułu *SMO.Reader.* Na podstawie odpowiednich przesunięć (z plików konfiguracyjnych xml) odczytywane są odpowiednie grupy bitów odpowiadające poszczególnym rejestrowanym sygnałom/parametrom. Dodatkowo w wykorzystywanej metodzie są zaimplementowane procedury mające zweryfikować poprawność nadesłanej ramki tak by pakiety z błędnymi danymi nie miały wpływu na kształt zarejestrowanego sygnału. Odczytane wartości zostają zapisane do obiektu typu słownik przechowującego wszystkie wyekstrahowane z pakietów wartości sygnałowe. Równocześnie, po każdej ekstrakcji, zostają zapisane do tablicy wartości czasu, w którym przechwycono dany pakiet z rejestratora. Otrzymana tablica będzie stanowiła podstawę czasu dla wizualizacji odebranych parametrów. Następnie wykorzystując funkcje *SharpPcap* zostają wyznaczone statystyki odebranego pakietu, zawierające takie informacje jak długość odebranego pakietu, wersja wykorzystywanego protokołu, adresy urządzenia źródła oraz docelowego itp, na podstawie których można zweryfikować poprawność odbieranych pakietów.

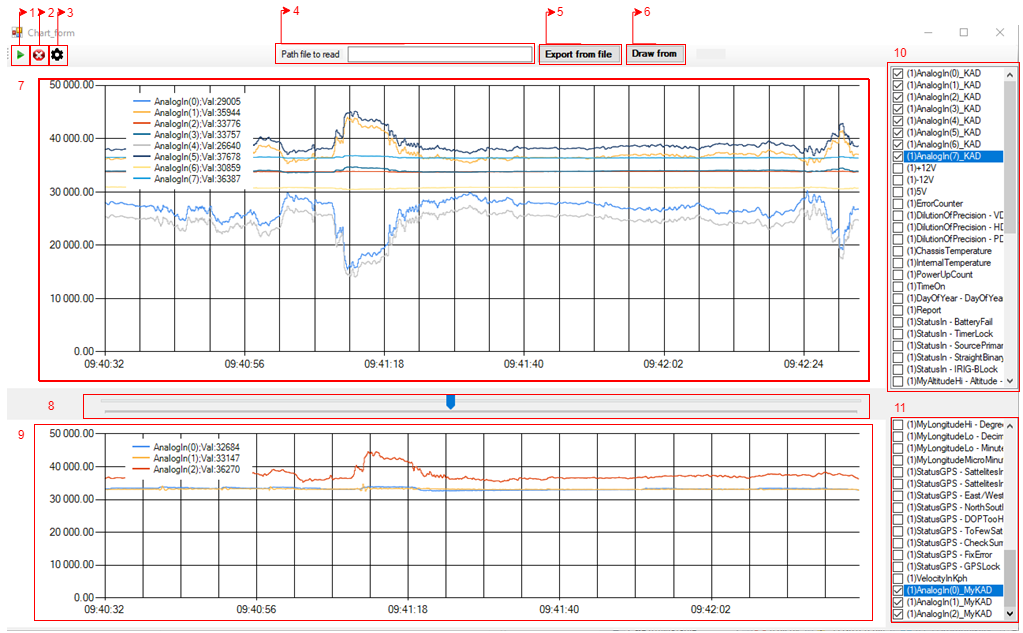
Po utworzeniu niezbędnych obiektów oraz wątków odpowiedzianych za przetwarzanie odbieranych informacji, w wątku głównym są wykonywane metody obsługujące zdarzenia typu *Event* odpowiadające naciśnięciu przycisku czy odebraniu ramki. W przypadku, kiedy zostanie przechwycony i przetworzony nowy pakiet, następuje aktualizacja tablicy pakietów w głównym oknie aplikacji. W każdej iteracji zostaje dodany nowy wiersz zawierający takie informacje jak numer kolejnego odebranego pakietu (*Count*), długość pakietu (*Length*) oraz z ilością wyekstrahowanych wartości sygnałowych (*dataKeys*).

Po wyborze jednego z wierszy w oknie tekstowym pojawiają się informacje o odczytanych parametrach danego pakietu. Dodatkowo, w dolnej części formularza Windows umieszczono wskaźniki mające za zadanie informować o: ilości wszystkich przechwyconych pakietów w sieci (*Recived packets*), pominiętych pakietach (*Dropped packets*), kolizji pakietów (*Interferance dropped packets*) czy ilości pakietów zbuforowanych aktualnie w kolejce (*QueueCounts*).

Przechwytywanie kolejnych pakietów może zostać zatrzymane bądź ponownie wznowione poprzez przycisk oznaczony na Rys 7 numerem 1. Kolejny przycisk z napisem „Save” służy do uruchomienia zapisu rejestrowanych ramek do plików .cap oraz .tsv. W momencie wciśnięcia przycisku rozpoczęcie zapisu jest sygnalizowane pojawieniem się czerwonej ikony. Wszystkie pakiety przechwycone od tego momentu zostają zapisane do pliku .cap, wyekstrahowane z nich wartości sygnałów zostają również zapisane do pliku tekstowego .tsv. Ponowne wciśniecie przycisku spowoduje zakończenie zapisu. Następny przycisk z podpisem ‘Settings” powoduje pojawienie się okna ustawień identycznego jak przy uruchomieniu aplikacji. Umożliwia to wprowadzenie modyfikacji ustawień danych wejściowych do programu takich jak wybrane pliki .xml, ścieżki zapisu plików, tablice skalowań czy numery portu i IP rejestratora. Kolejny przycisk z podpisem „Graph” umożliwia otwarcie okna odpowiedzialnego za wizualizacje wyekstrahowanych wartości sygnałowych. Wygląd okna wizualizacji przedstawiono na Rys 8

Na Rys 8 oznaczono podstawowe elementy okna wizualizacji, na które składają się kolejno:

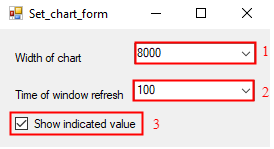
1. Przycisk rozpoczęcia wizualizacji
2. Przycisk wstrzymania wizualizacji
3. Przycisk okna ustawień wizualizacji
4. Okno tekstowe wyboru pliku do odczytu
5. Przycisk pobrania danych z pliku
6. Przycisk wizualizacji oznaczonych parametrów
7. Okno wykresu 1
8. Wskaźnik aktualnie podglądanej wartości
9. Okno wykresu 2
10. Lista parametrów do wywietlenia dla wykresu 1
11. Lista parametrów do wyświetlenia dla wykresu 2



Rys 8 Okno wizualizacji

Okno wizualizacji ma za zadanie wykonać wykresy serii kolejnych odebranych sygnałów, parametru wybranego przez użytkownika, w zakresie który wybierze. Do uruchomienia wykonywania wykresu w czasie rzeczywistym oraz jego zatrzymania służą dwa przyciski oznaczone na Rys 8 kolejno numerami 1 i 2. Po otworzeniu okna wizualizacji w trakcie przechwytywania kolejnych pakietów, w oknie wyboru parametrów oznaczonym na Rys 8 numerem 10 zostanie wyświetlona lista nazw kolejnych wyekstrahowanych sygnałów z przechwyconych pakietów. W celu wyświetlenia wybranych parametrów należy zaznaczyć pole wyboru przy danej nazwie sygnału. Przy wyborze parametrów należy pamietać, iż w jednym odebranym pakiecie ilość próbek danego sygnału może być różna w zależności od parametru. Powoduje to, iż ilość próbek pewnych sygnałów jest większa niż pozostałych, co skutkuje tym, że nie mogą zostać wyświetlone w tym samym momencie na tym samym wykresie. Informacje o różnicy w ilości odbieranych próbek są podane przy nazwie każdego sygnału, dodatkowo wprowadzono komunikaty ostrzegawcze mające poinformować użytkownika o błędnie wybranych parametrach.

Następnie po wciśnięciu przycisku oznaczony numerem 1 w oknie wykresu zostanie wygenerowany przebieg wybranego sygnału, który będzie cyklicznie odświeżany o nowe odbierane dane. Wybranie parametrów z listy parametrów do wyświetlenia oznaczonej numerem 11 spowoduje wyświetlenie ich na polu wykresu oznaczonym numerem 9. W trakcie wyświetlania kolejnych wartości istnieje możliwość przybliżenia fragmentu wykresu poprzez oznaczenie obszaru, który ma wypełnić okno wykresu. W celu podejrzenia danej wartości na wykresie służy wskaźnik przesuwny oznaczony numerem 8. Po zaznaczeniu opcji wyświetlania wartości w oknie ustawień wizualizacji oraz przesunięciu suwaka, w legendzie wykresu pokażą się wartości odpowiadające wartości sygnału, w miejscu wskazanym przez wskaźnik przesuwny. Inną opcją podglądu wartości jest zatrzymanie wykresu i wskazanie w dany punkt wykresu myszką. Pojawi się wtedy okno zawierające dane wskazanego punktu wykresu. Zmiany ustawień pramaterów wykresu można dokonać poprzez okna ustawień wizualizacji które otwiera się poprzez przycisk z numerem 3 wygląd okna przedstawiono na Rys 9. Należy również pamiętać, iż ilość wybranych parametrów wyświetlanych na wykresie, jak również czas odświeżania wykresu oraz ilość punktów wyświetlanych na wykresie wpływają na wydajność procesu wizualizacji. Zaleca się dobierać parametry wykresu tak by uzyskać jak najwyższą wydajność z zachowaniem pożądanej funkcjonalności.



Rys 9 Okno ustawień wizualizacji

Na okno ustawień wizualizacji składają się:

1. Okno wyboru szerokości wykresu wskazująca ile ostatnich odebranych próbek sygnału zostanie wyświetlona na wykresie
2. Czas odświeżania się wykresu podany w milisekundach
3. Pole wyboru możliwości wyświetlania wartości na wykresie wskazanej przez zamieszczony na oknie wizualizacji suwak

Dodatkową opcją zaimplementowaną w programie jest możliwość wyświetlania wykresów na podstawie wskazanych plików .cap. W oknie tekstowym oznaczonym na Rys 8 numerem 4 wskazuje się ścieżkę dostępu do pliku .cap zawierającym parametry sygnałów zgodnych z zadanym plikiem konfiguracyjnym. Uruchamiając przycisk oznaczony numerem 5 zostają pobrane kolejne pakiety zapisane na pliku i zostają z nich wyekstrahowane parametry sygnałów, których lista pojawia się w oknie oznaczonym numerem 10. W celu wizualizacji pobranych sygnałów należy wybrać te parametry z listy, które mają zostać wygenerowane na wykresie i następnie wcisnąć przycisk z napisem „Draw from”. Wtedy w oknie wykresu oznaczonym numerem 7 zostaną wygenerowane wykresy wybranych sygnałów tak jak w przypadku wizualizacji w czasie rzeczywistym.

## OPIS KOMPONENTÓW SYSTEMÓW

Poniżej opisano moduły aplikacji oraz zawarte w nich klasy/pliki.

**LiveCaptureWinform** – moduł główny do przechwytywania pakietów i ich wizualizacji

*\CaptureForm.cs* – klasa odpowiadająca za przechwytywanie pakietów i ich ekstrakcje zawierająca interfejs graficzny

*\Chart\_Form.cs* – - klasa wizualizacji otrzymanych wartości sygnałowych

*\DeviceListForm.cs –* interfejs graficzny wyboru karty sieciowej

*\Set\_chart\_form.cs –* interfejs graficzny ustawień właściwości wykresu

*\Settings\_form.cs* – interfejs graficzny ustawień aplikacji

*\Warning\_Form.cs –*interfejs graficzny komunikatu ostrzegawczego

*\Warnigng\_Form2.cs –* interfejs graficzny drugiego komunikatu ostrzegawczego

*\Program.cs -* punkt początkowy dla niezależnej aplikacji (.exe)

**SmoStartup** – moduł dll do konfiguracji i odczytu przechwyconych pakietów

*Converter.cs* – główna klasa modułu służąco do konwersji przechwyconych pakietów

*LiveConverter.cs* – pomocnicza klasa modułu służąco do konfiguracji na podstawie dostarczonych danych wejściowych

*InjectorBootstraper.cs* – klasa pomocnicza do odczytu xml konfiguracyjnych

*LiveAircraftDataProvider.cs –* klasa pomocnicza do konfiguracji początkowej

**SmoReader** – moduł dll do odczytu surowych danych

*\ScaleParameter.cs* – struktura danych wykorzystywana podczas skalowania sygnału

*\Signal.cs* – struktura danych zawierająca pojedynczy sygnał

*\Infrastructure\GlobalSettings.cs* – globalne ustawienia

*\Utils\Filtering.cs* – klasa pomocnicza zawierająca algorytmy filtrowania

*\Utils\MathUtils.cs* – klasa pomocnicza zawierająca pomocnicze operacje matematyczne

*\Utils\TimeFormatter.cs* – klasa pomocnicza zawierająca narzędzia do formatowania znaczników czasowych

*\ConfigParser.cs* – klasa służąca do przetwarzania plików konfiguracyjnych xml – w celu uzyskania offsetów niezbędnych do przeprowadzania odczytu surowych zapisów

*\Reader.cs* – główna klasa w oprogramowaniu służąca do odczytu surowych danych pakiet po pakiecie

*\StrainGaugeScaler.cs* – klasa zawierająca algorytm skalowania sygnałów na podstawie tabel \*.tsv

**Scaling converter** – moduł pomocniczy do konwersji tablic skalowania

**SharpPcap** – moduł dll do przechwytywania pakietów

## PRZYSZŁE MODYFIKACJE OPROGRAMOWANIA

Przeprowadzone prace stanowią pierwszy etap prac nad opracowaniem oprogramowania, które umożliwi w przyszłości zastąpienie programu GS Works 8 w trakcie przeglądów okresowych oraz usprawni proces sprawdzania poprawności działania czujników zamontowanych na samolocie. Na podstawie pierwszej serii testów określono zalecane modyfikacje oprogramowania w celu poprawy jego funkcjonalności.

* możliwość wyekstrahowania wartości sygnału z dwóch różnych pakietów na podstawie jednego pliku konfiguracyjnego przy rozróżnieniu ramek danych na podstawie ID pakietu
* zwiększenie efektywności wyświetlania w przypadku bardzo dużych strumieni danych
* możliwość eksportu wydzielonej części wykresu do pliku .tsv oraz w postaci.png
* możliwość ukrycia dolnego okna w trakcie wyświetlania danych z pliku
* możliwość odczytu danych z pliku .tsv
* dodanie markera na wykresie skazującego precyzyjnie punkt, którego wartość jest wyświetlana
* możliwość ukrycia bądź przesunięcia legendy wykresu
* zmapowanie wartości na wykresach na liczby całkowite
* możliwość zadania wartości min, max oraz kroku na osiach wykresu

Wyznaczone funkcjonalności zostały określone na podstawie pierwszej serii testów oprogramowania. W celu określania kolejnych modyfikacji umożlwiających rozszerzenie funkcjonalności oprogramowania zostaną przeprowadzone kolejne serie testów aplikacji.

# PODSUMOWANIE I WNIOSKI

1. W opracowaniu przedstawiono dokumentację oprogramowania umożliwiającego uzyskanie w czasie rzeczywistym danych z rejestratora SSR-500 podczas prac obsługowych.
2. Wykazano, że opracowana aplikacja posiada podstawowe funkcjonalności opracowane na podstawie metodyki MB-14.31.2015.
3. Wskazano technologie oraz biblioteki zastosowane przy opracowaniu oprogramowania
4. Oprogramowanie ma budowę modułową podstawowe moduły zawierające różne części funkcjonalności zawarte są w oddzielnych bibliotekach dll oraz plik wykonywalny LiveCaptureForm.exe zawierający część funkcjonalności oraz interfejs graficzny.
5. W opisie przedstawiono proces przechwytywania pakietów IPv4 i zaznaczono istotne elementy tego procesu
6. Wyszczególniono proces ekstrakcji danych mających na celu uzyskanie z pakietów IPv4 wartości sygnałów z rejestratora SSR-500
7. Opisano możliwości aplikacji, omówiono ogólny algorytm działania aplikacji oraz przedstawiono instrukcje korzystania z opracowanego oprogramowania
8. Dokumentacja ukazuje schemat zależności pomiędzy częściami składowymi systemu, oraz ukazuje podstawowe procedury wykorzystywane w systemie.
9. Wskazane jest rozszerzenie funkcjonalności oprogramowania, w celu możliwości zastąpienia licencjonowanego programu GS Works 8 przy wykonywaniu obsług okresowych zgodnie z metodyką MB-14.31.2015

# ZAŁĄCZNIKI

W załącznikach podano kod źródłowy klas spełniających główną role w trakcie pracy aplikacji: klasa *CaptureForm*, klasa *ChartForm* oraz klasa *Convert*

## A. Kod źródłowy klasy CaptureForm

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Collections;

using SharpPcap;

using SharpPcap.LibPcap;

using SharpPcap.AirPcap;

using SharpPcap.WinPcap;

using PacketDotNet;

using Smo.Startup;

using Smo.Common.Entities;

using Smo.Common.Infrastructure;

using Smo.Common.Public.Repositories;

using SmoReader.Entities;

namespace WinformsExample

{

public partial class CaptureForm : Form

{

public static string path\_scalingtable = "";

public static string path\_configxml = "";

public static string path\_savetsv = "";

public static string path\_instrumentsetting = "";

public static string port = "";

public static string path\_savecap = "";

public static string IP = "";

public static string aircraftname = "";

private Settings settings = new Settings();

private Chart\_form chartform = new Chart\_form();

public static Dictionary<string, List<ValueType>> extractedSamples = new Dictionary<string, List<ValueType>>();

public static Dictionary<string, List<ValueType>> temp\_exSam= new Dictionary<string, List<ValueType>>();

public static bool start\_chart = false;

public static List<DateTime> timearray = new List<DateTime>();

public static List<DateTime> timearray\_128 = new List<DateTime>();

public static List<DateTime> timearray\_256 = new List<DateTime>();

public static int data\_delay = 100;

/// <summary>

/// When true the background thread will terminate

/// </summary>

/// <param name="args">

/// A <see cref="System.String"/>

/// </param>

private bool BackgroundThreadStop;

/// <summary>

/// Object that is used to prevent two threads from accessing

/// PacketQueue at the same time

/// </summary>

/// <param name="args">

/// A <see cref="System.String"/>

/// </param>

private object QueueLock = new object();

/// <summary>

/// The queue that the callback thread puts packets in. Accessed by

/// the background thread when QueueLock is held

/// </summary>

private List<RawCapture> PacketQueue = new List<RawCapture>();

/// <summary>

/// The last time PcapDevice.Statistics() was called on the active device.

/// Allow periodic display of device statistics

/// </summary>

/// <param name="args">

/// A <see cref="System.String"/>

/// </param>

private DateTime LastStatisticsOutput;

/// <summary>

/// Interval between PcapDevice.Statistics() output

/// </summary>

/// <param name="args">

/// A <see cref="System.String"/>

/// </param>

private TimeSpan LastStatisticsInterval = new TimeSpan(0, 0, 2);

private System.Threading.Thread backgroundThread;

private DeviceListForm deviceListForm;

private static ICaptureDevice device;

private static CaptureFileWriterDevice captureFileWriter;

private static bool save\_file = false;

private static bool save\_tsv = false;

public CaptureForm()

{

InitializeComponent();

Application.ApplicationExit += new EventHandler(Application\_ApplicationExit);

}

void Application\_ApplicationExit(object sender, EventArgs e)

{

}

private void CaptureForm\_Load(object sender, EventArgs e)

{

settings = new Settings();

settings.ShowDialog();

deviceListForm = new DeviceListForm();

deviceListForm.OnItemSelected += new DeviceListForm.OnItemSelectedDelegate(deviceListForm\_OnItemSelected);

deviceListForm.OnCancel += new DeviceListForm.OnCancelDelegate(deviceListForm\_OnCancel);

timearray.Add(DateTime.Now);

using (StreamReader sr = new StreamReader("actual\_settings\\settings.txt"))

{

path\_configxml = sr.ReadLine();

path\_scalingtable = sr.ReadLine();

path\_instrumentsetting = sr.ReadLine();

path\_savecap = sr.ReadLine();

path\_savetsv = sr.ReadLine();

port = sr.ReadLine();

IP= sr.ReadLine();

aircraftname = sr.ReadLine();

}

}

void deviceListForm\_OnItemSelected(int itemIndex)

{

// close the device list form

deviceListForm.Hide();

StartCapture(itemIndex);

}

void deviceListForm\_OnCancel()

{

Application.Exit();

}

public class PacketWrapper

{

public RawCapture p;

public int Count { get; private set; }

//public PosixTimeval Timeval { get { return p.Timeval; } }

// LinkLayers LinkLayerType { get { return p.LinkLayerType; } }

public int Length { get { return p.Data.Length; } }

//public int ip\_dest { get { return p.; } }

public int dataKeys{ get; private set; }

public PacketWrapper(int count, RawCapture p,int dataKeys)

{

this.Count = count;

this.dataKeys = dataKeys;

this.p = p;

}

}

private PacketArrivalEventHandler arrivalEventHandler;

private CaptureStoppedEventHandler captureStoppedEventHandler;

private void Shutdown()

{

if (device != null)

{

device.StopCapture();

device.Close();

device.OnPacketArrival -= arrivalEventHandler;

device.OnCaptureStopped -= captureStoppedEventHandler;

device = null;

// ask the background thread to shut down

BackgroundThreadStop = true;

// wait for the background thread to terminate

backgroundThread.Join();

// switch the icon back to the play icon

startStopToolStripButton.Image = global::WinformsExample.Properties.Resources.play\_icon\_enabled;

startStopToolStripButton.ToolTipText = "Select device to capture from";

}

}

private void StartCapture(int itemIndex)

{

packetCount = 0;

device = CaptureDeviceList.Instance[itemIndex];

packetStrings = new Queue<PacketWrapper>();

bs = new BindingSource();

dataGridView.DataSource = bs;

LastStatisticsOutput = DateTime.Now;

var converter = Converter.BuildLiveConverter(aircraftname, path\_configxml, path\_scalingtable, path\_instrumentsetting);

// start the background thread

BackgroundThreadStop = false;

backgroundThread = new System.Threading.Thread(() =>BackgroundThread(converter));

backgroundThread.Start();

// setup background capture

arrivalEventHandler = new PacketArrivalEventHandler((sender, e) => device\_OnPacketArrival(this, e, converter, extractedSamples));

device.OnPacketArrival += arrivalEventHandler;

captureStoppedEventHandler = new CaptureStoppedEventHandler(device\_OnCaptureStopped);

device.OnCaptureStopped += captureStoppedEventHandler;

device.Open();

//string filter = "udp port 1027 and host 192.168.28.1";

string part1 = "";

string part2 = "";

string filter = "";

if (port != "") { part1 = String.Concat("udp port ", port); }

else { part1 = ""; }

if (IP != "") { part2 = String.Concat("host ", IP); }

else { part1 = ""; }

if (part1 == "") { filter = part2; }

else if (part2 == "") {filter = part1; }

else { filter = String.Concat(part1, " and ", part2); }

//device.Filter = "host 192.168.0.179";

device.Filter = filter;

// force an initial statistics update

captureStatistics = device.Statistics;

UpdateCaptureStatistics("0");

// start the background capture

device.StartCapture();

// disable the stop icon since the capture has stopped

startStopToolStripButton.Image = global::WinformsExample.Properties.Resources.stop\_icon\_enabled;

startStopToolStripButton.ToolTipText = "Stop capture";

}

{

if (status != CaptureStoppedEventStatus.CompletedWithoutError)

{

MessageBox.Show("Error stopping capture", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private Queue<PacketWrapper> packetStrings;

private int packetCount;

private BindingSource bs;

private ICaptureStatistics captureStatistics;

private bool statisticsUiNeedsUpdate = false;

void device\_OnPacketArrival(object sender, CaptureEventArgs e, LiveConverter converter, Dictionary<string, List<ValueType>> extractedSamples)

{

// print out periodic statistics about this device

var Now = DateTime.Now; // cache 'DateTime.Now' for minor reduction in cpu overhead

var interval = Now - LastStatisticsOutput;

if (interval > LastStatisticsInterval)

{

Console.WriteLine("device\_OnPacketArrival: " + e.Device.Statistics);

captureStatistics = e.Device.Statistics;

statisticsUiNeedsUpdate = true;

LastStatisticsOutput = Now;

}

// lock QueueLock to prevent multiple threads accessing PacketQueue at

// the same time

lock (QueueLock)

{

PacketQueue.Add(e.Packet);

}

}

private void CaptureForm\_Shown(object sender, EventArgs e)

{

deviceListForm.Show();

}

private void toolStripButton1\_Click(object sender, EventArgs e)

{

if (device == null)

{

deviceListForm.Show();

}

else

{

Shutdown();

}

}

/// <summary>

/// Checks for queued packets. If any exist it locks the QueueLock, saves a

/// reference of the current queue for itself, puts a new queue back into

/// place into PacketQueue and unlocks QueueLock. This is a minimal amount of

/// work done while the queue is locked.

///

/// The background thread can then process queue that it saved without holding

/// the queue lock.

/// </summary>

private void BackgroundThread(LiveConverter converter)

{

while (!BackgroundThreadStop)

{

bool shouldSleep = true;

lock (QueueLock)

{

if (PacketQueue.Count != 0)

{

shouldSleep = false;

}

}

if (shouldSleep)

{

System.Threading.Thread.Sleep(data\_delay - 99);

}

else // should process the queue

{

List<RawCapture> ourQueue;

lock (QueueLock)

{

// swap queues, giving the capture callback a new one

ourQueue = PacketQueue;

PacketQueue = new List<RawCapture>();

}

Console.WriteLine("BackgroundThread: ourQueue.Count is {0}", ourQueue.Count);

temp\_exSam = new Dictionary<string, List<ValueType>>();

foreach (var packet in ourQueue)

{

// Here is where we can process our packets freely without

// holding off packet capture.

//

// NOTE: If the incoming packet rate is greater than

// the packet processing rate these queues will grow

// to enormous sizes. Packets should be dropped in these

// cases

if (save\_file == true)

{

captureFileWriter.Write(packet);

save\_tsv = true;

}

var result = converter.DecodePacket(packet);

temp\_exSam = new Dictionary<string, List<ValueType>>();

result.Samples.ForEach(s =>

{

if (extractedSamples.ContainsKey(s.Key))

extractedSamples?[s.Key].Add(s.Value);

else

extractedSamples[s.Key] = new List<ValueType>() { s.Value };

if (temp\_exSam.ContainsKey(s.Key))

temp\_exSam?[s.Key].Add(s.Value);

else

temp\_exSam[s.Key] = new List<ValueType>() { s.Value };

});

timearray.Add(DateTime.Now);

if (timearray.Count() > 1)

{

TimeSpan delta = TimeSpan.FromTicks((timearray[timearray.Count() - 1].Subtract(timearray[timearray.Count() - 2]).Ticks) / 128);

timearray\_128.Add(timearray[timearray.Count() - 2] + delta);

for (int i = 0 ; i < 127;i++)

{

timearray\_128.Add(timearray\_128.Last() + delta);

}

}

if (timearray.Count() > 1)

{

TimeSpan delta = TimeSpan.FromTicks((timearray[timearray.Count() - 1].Subtract(timearray[timearray.Count() - 2]).Ticks) / 256);

timearray\_256.Add(timearray[timearray.Count() - 2] + delta);

for (int i = 0; i < 255; i++)

{

timearray\_256.Add(timearray\_128.Last() + delta);

}

}

int dataKeys = extractedSamples.Keys.Count();

var packetWrapper = new PacketWrapper(packetCount, packet,dataKeys);

this.BeginInvoke(new MethodInvoker(delegate

{

packetStrings.Enqueue(packetWrapper);

}

));

packetCount++;

//zmien\_label1(packetCount.ToString());

var time = packet.Timeval.Date;

var len = packet.Data.Length;

Console.WriteLine("BackgroundThread: {0}:{1}:{2},{3} Len={4}",

time.Hour, time.Minute, time.Second, time.Millisecond, len);

}

if (save\_tsv == true && save\_file == false)

{

string headerLine = "";

foreach (string key in extractedSamples.Keys)

{

headerLine = headerLine + key + "\t";

}

var keys\_test = new List<string>(extractedSamples.Keys);

var test = keys\_test[0];

int len = extractedSamples[test].Count;

//string dumpTextPath = "C:\\Users\\pniedbala\\Desktop\\test\_data\\509\_valid\_file1\\data\_read2.tsv";

using (StreamWriter sw = File.CreateText(path\_savetsv))

{

sw.WriteLine(headerLine);

for (int i = 0; i < len; i++)

{

string valuesLine = "";

foreach (string key in extractedSamples.Keys)

{

valuesLine = valuesLine + extractedSamples[key][i] + "\t";

}

sw.WriteLine(valuesLine);

}

}

}

this.BeginInvoke(new MethodInvoker(delegate

{

bs.DataSource = packetStrings.Reverse();

}

));

if (statisticsUiNeedsUpdate)

{

UpdateCaptureStatistics(ourQueue.Count.ToString());

statisticsUiNeedsUpdate = false;

}

if (start\_chart)

{

try

{

this.Invoke((MethodInvoker)delegate { chartform.UpdateChart(); });

//UpdateChart();

}

catch

{

}

}

}

}

}

private void UpdateCaptureStatistics(string QueueCount)

{

captureStatisticsToolStripStatusLabel.Text = string.Format("Received packets: {0}, Dropped packets: {1}, Interface dropped packets: {2}, QueueCount: {3} ",

captureStatistics.ReceivedPackets,

captureStatistics.DroppedPackets,

captureStatistics.InterfaceDroppedPackets,

QueueCount);

}

private void CaptureForm\_FormClosing(object sender, FormClosingEventArgs e)

{

Shutdown();

}

private void splitContainer1\_Panel1\_Paint(object sender, PaintEventArgs e)

{

}

private void dataGridView\_SelectionChanged(object sender, EventArgs e)

{

if (dataGridView.SelectedCells.Count == 0)

return;

var packetWrapper = (PacketWrapper)dataGridView.Rows[dataGridView.SelectedCells[0].RowIndex].DataBoundItem;

var packet = Packet.ParsePacket(packetWrapper.p.LinkLayerType, packetWrapper.p.Data);

packetInfoTextbox.Text = packet.ToString(StringOutputType.VerboseColored);

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void openFileDialog1\_FileOk(object sender, CancelEventArgs e)

{

}

private void textBox1\_Click(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

}

private void toolStripButton1\_Click\_1(object sender, EventArgs e)

{

if (save\_file == false)

{

string capFile = path\_savecap;

captureFileWriter = new CaptureFileWriterDevice(capFile);

save\_file = true;

System.Drawing.Image img = System.Drawing.Image.FromFile("C:\\Users\\pniedbala\\Pictures\\icons\\rec5.png");

toolStripButton1.Image = img;

}

else

{

save\_file = false;

captureFileWriter.Close();

System.Drawing.Image img2 = System.Drawing.Image.FromFile("C:\\Users\\pniedbala\\Pictures\\icons\\sav2.jpg");

toolStripButton1.Image = img2;

}

}

private void toolStripButton2\_Click(object sender, EventArgs e)

{

settings = new Settings();

settings.Show();

}

private void toolStripButton3\_Click(object sender, EventArgs e)

{

chartform = new Chart\_form();

chartform.Show();

}

}

}

## B. Kod źródłowy klasy Chartform

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

//using System.Threading.Tasks;

using System.Threading;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

using SharpPcap;

using SharpPcap.LibPcap;

using SharpPcap.AirPcap;

using SharpPcap.WinPcap;

using PacketDotNet;

using Smo.Startup;

namespace WinformsExample

{

public partial class Chart\_form : Form

{

private double[] ChartArray = new double[1000];

private bool stop\_chart = false;

private Set\_chart\_form set\_chart\_form = new Set\_chart\_form();

private Thread Time;

public static int chart\_width = 8000;

public static bool show\_last\_value = false;

Warning\_Form warnform = new Warning\_Form();

Warningform2 warnform2 = new Warningform2();

public static int view\_point = 0;

public static Dictionary<string, List<ValueType>> extractedSamples = new Dictionary<string, List<ValueType>>();

public static Dictionary<string, List<ValueType>> temp\_exSam = new Dictionary<string, List<ValueType>>();

public static List<DateTime> timearrayf = new List<DateTime>();

public static List<DateTime> timearray\_128f = new List<DateTime>();

public static List<DateTime> timearray\_256f = new List<DateTime>();

public Chart\_form()

{

InitializeComponent();

var sample = CaptureForm.extractedSamples;

List<string> keys = new List<string>(sample.Keys);

//dodawanie chasklistbox;

dodawanie\_checklistbox(keys,checkedListBox1);

dodawanie\_checklistbox(keys,checkedListBox2);

chart1.Series.Clear();

chart1.Titles.Add("opis").Name = "opis";

chart1.Titles["opis"].Visible = false;

chart1.ChartAreas[0].CursorX.IsUserEnabled = true;

chart1.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;

chart1.ChartAreas[0].CursorY.IsUserEnabled = true;

chart1.ChartAreas[0].CursorY.IsUserSelectionEnabled = true;

chart1.ChartAreas[0].AxisX.MajorGrid.Interval = 500d;

//chart1.ChartAreas[0].AxisX.IsReversed = true;

SetZoom(chart1);

chart2.Series.Clear();

chart2.Titles.Add("opis").Name = "opis";

chart2.Titles["opis"].Visible = false;

chart2.ChartAreas[0].CursorX.IsUserEnabled = true;

chart2.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;

chart2.ChartAreas[0].CursorY.IsUserEnabled = true;

chart2.ChartAreas[0].CursorY.IsUserSelectionEnabled = true;

chart2.ChartAreas[0].AxisX.MajorGrid.Interval = 500d;

//chart2.ChartAreas[0].AxisX.IsReversed = true;

SetZoom(chart2);

}

private void TimeHandler()

{

while (true)

{

Thread.Sleep(1000);

}

}

private void SetZoom(Chart chart)

{

chart.ChartAreas[0].CursorX.Interval = 0;

chart.ChartAreas[0].CursorY.Interval = 0;

chart.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;

chart.ChartAreas[0].CursorY.IsUserSelectionEnabled = true;

chart.ChartAreas[0].AxisX.ScaleView.Zoomable = true;

chart.ChartAreas[0].AxisY.ScaleView.Zoomable = true;

chart.ChartAreas[0].CursorX.SelectionColor = Color.DarkGray;

chart.ChartAreas[0].CursorY.SelectionColor = Color.DarkGray;

}

private void dodawanie\_checklistbox(List<string> keys, CheckedListBox checkedListBox)

{

checkedListBox.Items.Clear();

List<string> name\_list = new List<string>();

for (int i = keys.Count() - 1; i >= 0; i--)

{

string[] split\_key = keys[i].ToString().Split('\_');

var name = "(" + CaptureForm.temp\_exSam[keys[i]].Count.ToString() + ")" + split\_key[split\_key.Count() - 1] ;

if (name.Contains("AnalogIn")) { name = name + "\_" + split\_key[1]; }

name\_list.Add(name);

checkedListBox.Items.Insert(0, name);

}

}

private void dodawanie\_checklistbox2(List<string> keys, CheckedListBox checkedListBox)

{

checkedListBox.Items.Clear();

List<string> name\_list = new List<string>();

for (int i = keys.Count() - 1; i >= 0; i--)

{

string[] split\_key = keys[i].ToString().Split('\_');

var name = "(" + extractedSamples[keys[i]].Count.ToString() + ")" + split\_key[split\_key.Count() - 1];

if (name.Contains("AnalogIn")) { name = name + "\_" + split\_key[1]; }

name\_list.Add(name);

checkedListBox.Items.Insert(0, name);

}

}

private void StartChart()

{

try

{

chart1.Series.Clear();

chart2.Series.Clear();

List<bool> co\_eksportowac1 = new List<bool>();

for (int i = 0; i < checkedListBox1.Items.Count; i++)

{

if (checkedListBox1.GetItemCheckState(i).ToString() == "Checked")

co\_eksportowac1.Add(true);

else

co\_eksportowac1.Add(false);

}

List<bool> co\_eksportowac2 = new List<bool>();

for (int i = 0; i < checkedListBox2.Items.Count; i++)

{

if (checkedListBox2.GetItemCheckState(i).ToString() == "Checked")

co\_eksportowac2.Add(true);

else

co\_eksportowac2.Add(false);

}

//chart1.Series["Series1"].Points.Clear();

List<string> keys = new List<string>(CaptureForm.extractedSamples.Keys);

int probkowanie = 0;

for (int i = 0; i <= co\_eksportowac1.Count() - 1; i++)

if (co\_eksportowac1[i])

{

var probkowanie\_key = 1;

try

{

probkowanie\_key = Int32.Parse(CaptureForm.temp\_exSam[keys[i]].Count.ToString());//.Split(')').First().Split('(').Last();

}

catch

{

return;

}

if (probkowanie == 0)

{ probkowanie = probkowanie\_key; }

if (probkowanie != probkowanie\_key) { warnform.ShowDialog(); break; }

chart1.Series.Add(keys[i]).ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.FastLine; ;

chart1.Series[keys[i]].ToolTip = "#VALY, #INDEX";

chart1.Series[keys[i]].XValueType = ChartValueType.DateTime;

chart1.Series[keys[i]].IsXValueIndexed = true;

chart1.Series[keys[i]].SmartLabelStyle.Enabled = false;

//chart1.Series[keys[i]].IsValueShownAsLabel = true;

}

chart1.ChartAreas[0].AxisX.LabelStyle.Format = "HH:mm:ss";

chart1.ChartAreas[0].AxisY.LabelStyle.Format = "N2";

probkowanie = 0;

for (int i = 0; i <= co\_eksportowac2.Count() - 1; i++)

if (co\_eksportowac2[i])

{

var probkowanie\_key = Int32.Parse(CaptureForm.temp\_exSam[keys[i]].Count.ToString());//.Split(')').First().Split('(').Last();

if (probkowanie == 0)

{ probkowanie = probkowanie\_key; }

if (probkowanie != probkowanie\_key) { warnform.ShowDialog(); break; }

chart2.Series.Add(keys[i]).ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.FastLine; ;

chart2.Series[keys[i]].ToolTip = "#VALY, #INDEX";

chart2.Series[keys[i]].XValueType = ChartValueType.DateTime;

chart2.Series[keys[i]].IsXValueIndexed = true;

}

chart2.ChartAreas[0].AxisX.LabelStyle.Format = "HH:mm:ss";

chart2.ChartAreas[0].AxisY.LabelStyle.Format = "N2";

Time = new Thread(new ThreadStart(this.TimeHandler));

Time.IsBackground = true;

Time.Start();

}

catch { }

}

private void StartChart\_file()

{

chart1.Series.Clear();

chart2.Series.Clear();

List<bool> co\_eksportowac1 = new List<bool>();

for (int i = 0; i < checkedListBox1.Items.Count; i++)

{

if (checkedListBox1.GetItemCheckState(i).ToString() == "Checked")

co\_eksportowac1.Add(true);

else

co\_eksportowac1.Add(false);

}

//chart1.Series["Series1"].Points.Clear();

List<string> keys = new List<string>(extractedSamples.Keys);

int probkowanie = 0;

for (int i = 0; i <= co\_eksportowac1.Count() - 1; i++)

if (co\_eksportowac1[i])

{

var probkowanie\_key = 1;

try

{

probkowanie\_key = Int32.Parse(extractedSamples[keys[i]].Count.ToString());//.Split(')').First().Split('(').Last();

}

catch

{

return;

}

if (probkowanie == 0)

{ probkowanie = probkowanie\_key; }

if (probkowanie != probkowanie\_key) { warnform.ShowDialog(); break; }

chart1.Series.Add(keys[i]).ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.FastLine; ;

chart1.Series[keys[i]].ToolTip = "#VALY, #INDEX";

chart1.Series[keys[i]].XValueType = ChartValueType.DateTime;

chart1.Series[keys[i]].IsXValueIndexed = true;

chart1.Series[keys[i]].SmartLabelStyle.Enabled = false;

//chart1.Series[keys[i]].IsValueShownAsLabel = true;

}

}

private void toolStripButton1\_Click(object sender, EventArgs e)

{

StartChart();

CaptureForm.start\_chart = true;

stop\_chart = false;

}

public void UpdateChart()

{

if (stop\_chart == false)

{

Changechart(chart2);

Changechart(chart1);

}

}

public void Changechart(Chart chart)

{

var series = chart.Series.ToList();

List<DateTime> timebase = new List<DateTime>();

foreach (Series seria in series)

{

chart.Series[seria.ToString().Substring(7)].Points.Clear();

var chartarray = CaptureForm.extractedSamples[seria.ToString().Substring(7)];

int iter = 0;

if (chartarray.Count() < chart\_width)

{ iter = 0; }

else { iter = chartarray.Count() - chart\_width - 1; }

var time\_divide =chartarray.Count() / (CaptureForm.timearray.Count() - 1);

if (time\_divide < 3) { timebase = CaptureForm.timearray; }

if (time\_divide > 120 && time\_divide < 130) { timebase = CaptureForm.timearray\_128; }

if (time\_divide > 250 && time\_divide < 260) { timebase = CaptureForm.timearray\_256; }

for (int i = iter; i < chartarray.Count()- 1; ++i)

{

chart.Series[seria.ToString().Substring(7)].Points.AddXY(timebase[i], chartarray[i]);

//chart1.Series[seria.ToString().Substring(7)].XValueType(Data);

}

//chart.Series[seria.ToString().Substring(7)].Points.Last().Label = chartarray[chartarray.Count() - 1].ToString();

if (show\_last\_value)

{

try

{

int which\_point = (view\_point \* (chartarray.Count() - 10) / 1000) + 1;

string[] split\_key = seria.ToString().Split('\_');

chart.Series[seria.ToString().Substring(7)].LegendText = split\_key[split\_key.Count() - 1] + ";Val:" + Math.Round(chart.Series[seria.ToString().Substring(7)].Points[which\_point].YValues[0]).ToString();

//chart.Series[seria.ToString().Substring(7)].LegendText = split\_key[split\_key.Count() - 1] + ";Val:" + Math.Round(chart.Series[seria.ToString().Substring(7)].Points.Last().YValues[0]).ToString();

}

catch(Exception ex)

{

}

}

//string[] split\_key = keys[i].ToString().Split('\_')

//chart.Series[seria.ToString().Substring(7)].Points.Last;

}

}

public void Changechartfile(Chart chart)

{

var series = chart.Series.ToList();

List<DateTime> timebase = new List<DateTime>();

foreach (Series seria in series)

{

chart.Series[seria.ToString().Substring(7)].Points.Clear();

var chartarray = extractedSamples[seria.ToString().Substring(7)];

int iter = 0;

if (chartarray.Count() < chart\_width)

{ iter = 0; }

else { iter = chartarray.Count() - chart\_width - 1; }

var time\_divide = chartarray.Count() / (timearrayf.Count() - 1);

if (time\_divide < 3) { timebase = timearrayf; }

if (time\_divide > 120 && time\_divide < 130) { timebase = timearray\_128f; }

if (time\_divide > 250 && time\_divide < 260) { timebase = timearray\_256f; }

for (int i = iter; i < chartarray.Count() - 1; ++i)

{

chart.Series[seria.ToString().Substring(7)].Points.AddXY(timebase[i], chartarray[i]);

//chart1.Series[seria.ToString().Substring(7)].XValueType(Data);

}

//chart.Series[seria.ToString().Substring(7)].Points.Last().Label = chartarray[chartarray.Count() - 1].ToString();

//string[] split\_key = keys[i].ToString().Split('\_')

//chart.Series[seria.ToString().Substring(7)].Points.Last;

}

}

private void Chart\_form\_FormClosing(object sender, FormClosingEventArgs e)

{

CaptureForm.start\_chart = false;

Thread.Sleep(2000);

}

private void toolStripButton2\_Click(object sender, EventArgs e)

{

stop\_chart = true;

}

private void toolStripButton3\_Click(object sender, EventArgs e)

{

set\_chart\_form = new Set\_chart\_form();

set\_chart\_form.Show();

}

private void trackBar1\_Scroll(object sender, EventArgs e)

{

view\_point = trackBar1.Value;

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void textBox1\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog1 = new OpenFileDialog();

openFileDialog1.InitialDirectory = "c:\\";

openFileDialog1.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

openFileDialog1.FilterIndex = 2;

openFileDialog1.RestoreDirectory = true;

openFileDialog1.ShowDialog();

textBox1.Text = openFileDialog1.FileName;

}

private void button1\_Click(object sender, EventArgs e)

{

if (textBox1.Text == "")

{

warnform2.ShowDialog(); return;

}

else

{

ICaptureDevice device;

string capFile = textBox1.Text;

try

{

// Get an offline device

device = new CaptureFileReaderDevice(capFile);

// Open the device

device.Open();

}

catch (Exception ex)

{

Console.WriteLine("Caught exception when opening file" + ex.ToString());

return;

}

//var converter = Converter.BuildLiveConverter(CaptureForm.aircraftname, CaptureForm.path\_configxml, CaptureForm.path\_scalingtable, CaptureForm.path\_instrumentsetting);

device.OnPacketArrival +=

new PacketArrivalEventHandler(device\_OnPacketArrival);

device.Capture();

var keys2 = new List<string>(extractedSamples.Keys);

dodawanie\_checklistbox2(keys2, checkedListBox1);

}

}

private static int packetIndex = 0;

private void device\_OnPacketArrival(object sender, CaptureEventArgs e)

{

if (e.Packet.LinkLayerType == PacketDotNet.LinkLayers.Ethernet)

{

//var packet = PacketDotNet.Packet.ParsePacket(e.Packet.LinkLayerType, e.Packet.Data);

//var ethernetPacket = (PacketDotNet.EthernetPacket)packet;

packetIndex++;

var converter = Converter.BuildLiveConverter(CaptureForm.aircraftname, CaptureForm.path\_configxml, CaptureForm.path\_scalingtable, CaptureForm.path\_instrumentsetting);

var result = converter.DecodePacket(e.Packet);

temp\_exSam = new Dictionary<string, List<ValueType>>();

result.Samples.ForEach(s =>

{

if (extractedSamples.ContainsKey(s.Key))

extractedSamples?[s.Key].Add(s.Value);

else

extractedSamples[s.Key] = new List<ValueType>() { s.Value };

if (temp\_exSam.ContainsKey(s.Key))

temp\_exSam?[s.Key].Add(s.Value);

else

temp\_exSam[s.Key] = new List<ValueType>() { s.Value };

});

timearrayf.Add(DateTime.Now);

if (timearrayf.Count() > 1)

{

TimeSpan delta = TimeSpan.FromTicks((timearrayf[timearrayf.Count() - 1].Subtract(timearrayf[timearrayf.Count() - 2]).Ticks) / 128);

timearray\_128f.Add(timearrayf[timearrayf.Count() - 2] + delta);

for (int i = 0; i < 127; i++)

{

timearray\_128f.Add(timearray\_128f.Last() + delta);

}

}

if (timearrayf.Count() > 1)

{

TimeSpan delta = TimeSpan.FromTicks((timearrayf[timearrayf.Count() - 1].Subtract(timearrayf[timearrayf.Count() - 2]).Ticks) / 256);

timearray\_256f.Add(timearrayf[timearrayf.Count() - 2] + delta);

for (int i = 0; i < 255; i++)

{

timearray\_256f.Add(timearray\_128f.Last() + delta);

}

}

// int dataKeys = extractedSamples.Keys.Count();

if (packetIndex%1000 == 0)

{

try

{

this.Invoke((MethodInvoker)delegate { Updatelabel(); });

//UpdateChart();

}

catch

{

}

}

}

}

public void Updatelabel()

{

label3.Text = "Process pacet" + packetIndex.ToString();

Application.DoEvents();

}

private void button2\_Click(object sender, EventArgs e)

{

StartChart\_file();

Changechartfile(chart1);

}

private void label3\_Click(object sender, EventArgs e)

{

}

}

}

## C. Kod źródłowy klasy Converter

using Smo.Common;

using Smo.Common.Enums;

using Smo.Common.Infrastructure;

using Smo.Common.Models;

using Smo.Common.Utils;

using Smo.Launcher;

using SmoReader.Utils;

using System;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using System.Threading.Tasks.Dataflow;

using Smo.Common.Public.Models;

using Smo.Common.Public.Repositories;

using Smo.Reader.Standard;

using SharpPcap;

using SmoReader;

namespace Smo.Startup

{

public class ConversionBatchStatus

{

public int TotalGroupCount { get; set; }

public int TotalFileCount { get; set; }

public int ProcessedFileCount { get; set; }

public int ProcessedGroupCount { get; set; }

}

public class Converter

{

public static BatchConverter BuildBatchConverter(IAircraftDataProvider aircraftDataProvider, string instrumentSettingsXmlPath)

{

var iocBootstrapper = new InjectorBootstraper(aircraftDataProvider, instrumentSettingsXmlPath);

var launcher = iocBootstrapper.GetService(typeof(BatchConverter)) as BatchConverter;

return launcher;

}

public static LiveConverter BuildLiveConverter(

string aircraftName,

string configXmlPath,

string scalingTsvPath,

string instrumentSettingsXmlPath)

{

var liveAircraftDataProvider = new LiveAircraftDataProvider(aircraftName, configXmlPath, scalingTsvPath);

var converter = new LiveConverter(aircraftName, liveAircraftDataProvider, instrumentSettingsXmlPath);

return converter;

}

}

public class LiveConverter

{

PacketDecoder \_decoder { get; set; }

string \_aircraftName { get; set; }

public LiveConverter(string aircraftName, IAircraftDataProvider aircraftDataProvider, string instrumentSettingsXmlPath)

{

var cfgParser = new ConfigParser(aircraftDataProvider, instrumentSettingsXmlPath);

var parameterDefinitions = cfgParser.ReadConfigurationXml(aircraftName, DateTime.Now);

\_decoder = new PacketDecoder(parameterDefinitions, aircraftDataProvider);

}

public PacketReadResult DecodePacket(RawCapture packet)

{

var result = \_decoder.DecodePacket(packet, \_aircraftName);

return result;

}

}

public class BatchConverter

{

private readonly IFilePersistence \_filePersistence;

private readonly IAircraftDataProvider \_aircraftDataProvider;

IGlobalPaths \_globalPaths;

private readonly ILoggingService \_loggingService;

public ConversionBatchStatus ConversionBatchStatus = new ConversionBatchStatus();

public AnalysisBatchStatus AnalysisBatchStatus = new AnalysisBatchStatus();

public BatchConverter(

IFilePersistence filePersistence,

IAircraftDataProvider aircraftRepository,

IGlobalPaths globalPaths,

ILoggingService loggingService)

{

\_filePersistence = filePersistence;

\_aircraftDataProvider = aircraftRepository;

\_globalPaths = globalPaths;

\_loggingService = loggingService;

}

public FileGroupScanResult ScanFiles(List<string> inputPaths)

{

var fileGroupScanner = new FileGroupScanner(\_aircraftDataProvider, \_globalPaths, \_loggingService);

var fileGroupScanResult = fileGroupScanner.Scan(inputPaths);

return fileGroupScanResult;

}

public async Task<ConversionBatchResult> StartConversion(ConversionLaunchCommand command)

{

ConversionBatchStatus.TotalGroupCount = command.ScannedFileGroups.Count;

ConversionBatchStatus.TotalFileCount = command.ScannedFileGroups.SelectMany(fpr => fpr.Files).Count();

Trace.TraceInformation($"Scanning Finished. Found {ConversionBatchStatus.TotalFileCount} files in {ConversionBatchStatus.TotalGroupCount} groups");

var fileMerger = new FileMerger(

\_aircraftDataProvider,

\_globalPaths);

fileMerger.cutoffDate = command.EarliestDate;

var extractedRecords = new ConcurrentBag<RecordModel>();

var readFileMetadatas = new ConcurrentBag<FileReadMetadata>();

await AsyncHelper.AsyncParallelForEach(command.ScannedFileGroups,

async (group) =>

{

\_loggingService?.Log("Starting Parallel Scan");

var fileGroupReader = new FileGroupReader(\_aircraftDataProvider, \_globalPaths, \_loggingService);

var fileReadResults = fileGroupReader.ReadFiles(group.Files);

fileReadResults.ForEach(fr => readFileMetadatas.Add(fr.Metadata));

var mergeResult = fileMerger.MergeFiles(fileReadResults);

//now analyze

var analysisResult = new AnalysisEngine(\_aircraftDataProvider, \_loggingService).Analyze(mergeResult);

if (analysisResult.isValidForAnalysis || command.letSmallFilesThrough)

{

//write tsv

var tsvWriter = new TsvWriter(command.TsvOutputFolder, \_loggingService);

var tsvPath = tsvWriter.WriteTsv(analysisResult);

ConversionBatchStatus.ProcessedGroupCount++;

Trace.TraceInformation($"Converted {ConversionBatchStatus.ProcessedGroupCount}/{ConversionBatchStatus.TotalGroupCount} groups");

var extractedRecord = new RecordModel()

{

AircraftName = analysisResult.Metadata.AircraftName,

MetaData = analysisResult.Metadata,

InputFiles = group.Files.Select(f => f.ReadMetadata).ToList(),

TsvPath = tsvPath,

MaxValues = analysisResult.MaxValues,

MinValues = analysisResult.MinValues

};

extractedRecords.Add(extractedRecord);

}

await Task.Yield();

},

8, TaskScheduler.Current);

//GC.Collect();

Trace.TraceInformation($"Final Extracted group number: {ConversionBatchStatus.ProcessedGroupCount} of {ConversionBatchStatus.TotalGroupCount} potential groups");

//this is for better logging of file read results into the db

var inputFiles = command.ScannedFileGroups.SelectMany(g => g.Files.Select(f => f.ReadMetadata)).ToList();

var readFiles = readFileMetadatas.ToList();

var updatedInputFiles = new List<FileReadMetadata>();

inputFiles.ForEach(f =>

{

var match = readFiles.FirstOrDefault(r => r.FilePath == f.FilePath);

if (match?.IsReadSuccessful ?? false)

{

f = match;

}

else if (match != null)

{

f.Flags = match.Flags;

f.IsReadSuccessful = match.IsReadSuccessful;

}

updatedInputFiles.Add(f);

});

var allExtractedMetadatas = extractedRecords.SelectMany(r => r.InputFiles);

return new ConversionBatchResult

{

ConverterVersion = GetVersion(),

Records = extractedRecords.ToList(),

OrphanedFiles = updatedInputFiles

.Where(ifm => !allExtractedMetadatas.Any(fm => fm.FilePath == ifm.FilePath))

.ToList()

};

}

public string GetVersion()

{

return Assembly.GetAssembly(typeof(FileMerger)).GetName().Version.ToString();

}

}

}