



**Politechnika Łódzka**

**Wydział Fizyki Technicznej, Informatyki  
i Matematyki Stosowanej**

***Marcin Mazur***  
**242467**

**PRACA DYPLOMOWA**  
**inżynierska**  
na kierunku Informatyka Stosowana

## **Wykorzystanie oprogramowania Open-Source do współpracy z kamerami TP-Link TAPO**

Instytut Informatyki I72  
**Promotor:** dr inż. Krzysztof Lichy

ŁÓDŹ 2026



# Spis treści

<b>Streszczenie</b>	<b>3</b>
Słowa kluczowe . . . . .	3
<b>Wstęp</b>	<b>4</b>
<b>Cel i zakres pracy</b>	<b>5</b>
<b>1 Wprowadzenie technologiczne Kamer IP</b>	<b>7</b>
1.1 Zastosowanie Kamer IP . . . . .	7
1.1.1 Monitoring . . . . .	8
1.1.2 Kontrola Dostępu . . . . .	8
1.1.3 Zarządzanie Procesami Biznesowymi . . . . .	9
1.1.4 Technologie Smart . . . . .	9
1.1.5 Analiza Danych . . . . .	9
1.2 Budowa . . . . .	10
1.2.1 Budowa Fizyczna - Hardware . . . . .	10
1.2.2 Oprogramowanie - Firmware . . . . .	11
1.3 Zasada działania . . . . .	12
1.3.1 Architektura Komunikacji Sieciowej: Stos Protokółów i Standardy	12
1.3.2 Provisioning: Inicjalizacja i Uwierzytelnianie Urządzenia . . . . .	18
1.3.3 Przetwarzanie Sygnału Audiowizualnego: Od Fotonu do Pakietu Danych . . . . .	20
1.3.4 Strumieniowanie: Transmisja Danych w Czasie Rzeczywistym . .	24
1.4 Funkcje . . . . .	28
1.4.1 Obrót PTZ . . . . .	28
1.4.2 Wykrywanie obiektów i zdarzeń - AI . . . . .	29
1.4.3 Wykrywanie ruchu . . . . .	29
1.4.4 Noktowizja i termowizja . . . . .	30
1.4.5 Dwukierunkowe audio . . . . .	31
1.4.6 Zapis danych . . . . .	31
1.4.7 Integracja z Inteligentnymi Systemami . . . . .	32
1.4.8 Powiadomienia push . . . . .	33
1.5 Ograniczenia . . . . .	34
1.5.1 Ograniczenia Wynikające z Infrastruktury Sieciowej . . . . .	34
1.5.2 Luki w Zabezpieczeniach i Ryzyka dla Prywatności . . . . .	36
1.5.3 Ograniczenia Modelu Biznesowego i Uzależnienie od Producenta	39
1.6 Wnioski - Analiza . . . . .	41
<b>2 Analiza Kamery TP-Link TAPO C200</b>	<b>48</b>

2.1	Charakterystyka Ogólna i Pozycja Rynkowa . . . . .	48
2.2	Architektura Sprzętowa . . . . .	49
2.3	Architektura Oprogramowania i Protokoły Komunikacyjne . . . . .	50
2.4	Analiza Możliwości Funkcjonalnych . . . . .	53
2.5	Ograniczenia i Zjawisko „Vendor Lock-in” . . . . .	55
2.6	Aspekty Bezpieczeństwa i Prywatności . . . . .	56
<b>3</b>	<b>Metodologia i implementacja rozwiązania</b>	<b>59</b>
3.1	Metodyka Projektowa . . . . .	59
3.1.1	Double Diamond . . . . .	59
3.2	Architektura rozwiązania . . . . .	61
3.2.1	Architektura Wielowarstwowa . . . . .	62
3.2.2	Wzorzec Architektury Potokowej . . . . .	67
3.2.3	Wzorzec Architektury Opartej na Zdarzeniach . . . . .	69
3.3	Diagramy . . . . .	71
3.4	Zastosowane narzędzia i technologie . . . . .	71
3.4.1	Język Programowania . . . . .	71
3.4.2	Zarządzanie Zależnościami . . . . .	72
3.4.3	Ekosystem Konteneryzacji . . . . .	73
3.4.4	Interfejs Webowy i Protokół Komunikacji . . . . .	73
3.4.5	Biblioteki Przetwarzania Multimediów . . . . .	75
3.4.6	Kontrola Kamery i Inżynieria Wsteczna . . . . .	76
3.4.7	Narzędzie do Kompozycji i Zapisu Danych . . . . .	77
3.5	Proces implementacji rozwiązania . . . . .	78
3.5.1	Provisioning i pierwotna konfiguracja środowiska kamery . . . . .	79
3.5.2	Konfiguracja Środowiska Programistycznego . . . . .	80
3.5.3	Implementacja serwera HTTP . . . . .	81
3.5.4	Implementacja Warstwy Dostępu do Sprzętu (HAL) . . . . .	82
3.6	Podsumowanie . . . . .	85
<b>4</b>	<b>Testowanie i Analiza wyników</b>	<b>86</b>
4.1	Zakres Testów . . . . .	86
4.2	Środowisko Testowe . . . . .	86
4.3	Wyniki testów i Analiza . . . . .	86
4.4	Podsumowanie . . . . .	86
	<b>Wnioski Końcowe</b>	<b>87</b>
	Kierunki dalszego rozwoju . . . . .	87
	Podsumowanie pracy . . . . .	87

<b>Spis rysunków</b>	<b>89</b>
<b>Spis tabel</b>	<b>90</b>

## **Streszczenie**

## **Słowa kluczowe**

IoT, Kamera IP, TP-Link Tapo, Open Source, PyTapo, Docker, RTSP, Detekcja Ruchu, Flask.

# Wstęp

Globalny rynek systemów monitoringu przechodzi dynamiczną transformację, będącą efektem rozwoju **Internetu Rzeczy (IoT)**. Kamery IP stały się wszechobecnym elementem infrastruktury cyfrowej, pełniąc funkcje od podstawowego dozoru, aż po zaawansowaną analizę danych. Równolegle z postępem technologicznym, pojawia się wyzwanie o charakterze inżynierskim, jakim jest dominacja systemów opartych na **zamkniętym oprogramowaniu (proprietary software)**.

Wybór tematu pracy wynika z konieczności zaadresowania problemu **vendor lock-in** w kontekście popularnych kamer konsumenckich, na przykładzie urządzeń TP-Link Tapo. Zjawisko to, polegające na uzależnieniu pełnej funkcjonalności sprzętu od infrastruktury chmurowej i aplikacji mobilnej producenta, ogranicza **dostępność danych** oraz **możliwości integracji** z otwartymi systemami automatyki i bezpieczeństwa. Problem ten jest szczególnie istotny w kontekście **cyber bezpieczeństwa**, gdzie zamknięte i często nieaudytowane firmware może stanowić potencjalny wektor ataku.

W pracy zastosowano **metodykę Double Diamond**, dzieląc proces projektowy na fazy eksploracji i definiowania problemu (analiza protokołów kamery) oraz fazy rozwoju i dostarczania rozwiązania. Warstwa aplikacyjna została zaimplementowana w języku **Python 3.13** z wykorzystaniem **konteneryzacji Docker** dla zapewnienia izolacji i wysokiej **reprodukowalności środowiska**. Komunikacja z kamerą odbywa się poprzez bibliotekę **PyTapo**, natomiast przetwarzanie strumienia wideo RTSP realizują narzędzia **PyAV** i **OpenCV**. Taki zestaw narzędzi, osadzony w architekturze serwera **Flask** z protokołem **WebSocket's**, pozwolił na stworzenie systemu o niskim opóźnieniu (*low latency*).

Niniejsza praca ma za zadanie stanowić nie tylko dowód kompetencji inżynierskich, ale także praktyczny wkład w rozwój otwartych technologii w dziedzinie monitoringu IoT.

# Cel i zakres pracy

## Cel

**Celem głównym** niniejszej pracy inżynierskiej jest opracowanie oraz implementacja kompletnego, modułowego rozwiązania programistycznego opartego wyłącznie na **otwartym oprogramowaniu (Open Source)**, które umożliwi pełne wykorzystanie funkcjonalności kamery IP TP-Link Tapo C200 w środowisku lokalnym i niezależni użytkownika od zamkniętej infrastruktury producenta (problem *vendor lock-in*).

Osiągnięcie celu głównego jest weryfikowane poprzez realizację następujących, **konkretnych i mierzalnych** celów szczegółowych:

- Umożliwienie stabilnego **wyświetlania obrazu w czasie rzeczywistym** w przeglądarce internetowej. Weryfikacja nastąpi poprzez pomiar **opóźnienia strumienia wideo (*Latency*)** oraz wskaźnika **klatek na sekundę (FPS)**, celem osiągnięcia płynności monitoringu.
- Zdalne sterowanie kluczowymi funkcjami kamery, w tym **ruchem PTZ** (Pan/Tilt/Zoom).
- Implementacja **algorytmu wykrywania ruchu**, z poziomu serwera hostującego. Weryfikacja nastąpi poprzez analizę **efektywności algorytmów** mierzoną w kategoriach czasu przetwarzania klatki oraz minimalizacji błędów detekcji.
- Zbudowanie rozwiązania w oparciu o technologię **Docker** w celu zapewnienia **skalowalności systemu** oraz **reprodukowalności środowiska** na platformach mikserwerowych IoT (np. Raspberry Pi).
- Implementacja funkcjonalności **zapisu nagrań wideo** na serwerze hostującym z możliwością ich późniejszego **odtwarzania** poprzez interfejs webowy.



## Zakres Pracy

Zakres pracy inżynierskiej obejmuje projektowanie, implementację oraz testowanie modułowego systemu klient-serwer. Praca stanowi odpowiedź na problem *vendor lock-in* w segmencie kamer IoT, uzasadniając wybór tematu rosnącą potrzebą na otwarte i bezpieczne systemy zarządzania danymi.

## Aspekty objęte zakresem pracy

- Projekt trójwarstwowej architektury kontenerowej (Docker) dla warstwy dostępu do sprzętu, logiki biznesowej (Flask/WebSocket's) oraz warstwy prezentacji (Web Client).
- Wykorzystanie biblioteki PyTape do obsługi nieudokumentowanego API komend sterujących kamery (PTZ, tryb nocny, aktywacja usług).
- Praca skupia się na przechwytywaniu jednokierunkowego strumienia wideo i audio. Implementacja pełnej komunikacji zwrotnej w czasie rzeczywistym jest **poza zakresem** projektu.
- Przeprowadzenie **testów wydajnościowych** skupiających się na **zużyciu zasobów (CPU/RAM)** hosta podczas ciągłej analizy strumienia wideo.

## Wyłączenia z zakresu pracy

W celu zachowania osiągalności i weryfikowalności celów w ramach pracy inżynierskiej, poniższe aspekty zostały wykluczone, ze względu na ich złożoność badawczą lub techniczną:

- Protokół inicjalizacji - **Provisioning** - kamery Tapo w trybie Access Point (AP) jest nieudokumentowany, szyfrowany i opiera się na wymianie kluczy sesyjnych, co wymaga weryfikacji po stronie chmury TP-Link. Odtworzenie tego protokołu jest zadaniem na poziomie projektu badawczego i **wykracza poza ramy niniejszej pracy**. W konsekwencji, praca zakłada, że **kamera została jednorazowo skonfigurowana w sieci Wi-Fi** przy użyciu oficjalnej aplikacji mobilnej.
- Implementacja modeli **uczenia maszynowego** (np. rozpoznawanie twarzy, klasyfikacja obiektów - YOLO), ze względu na wysokie wymagania obliczeniowe i złożoność czasową, **została wykluczona**. Praca koncentruje się na detekcji ruchu opartej na różnicy klatek (OpenCV).

# 1    Wprowadzenie technologiczne Kamer IP

Rozdział ten ma za zadanie ugruntować zrozumienie złożoności systemów kamer IP i precyzyjnie wskazać na luki w otwartych standardach, które musi wypełnić zaprojektowane rozwiązanie. Współczesne systemy monitoringu wizyjnego oparte na kamerach IP stanowią kluczowy element infrastruktury bezpieczeństwa, wykraczając funkcjonalnością poza tradycyjne, analogowe systemy CCTV. Ewolucja ta jest ściśle związana z rozwojem sieci komputerowych i koncepcji IoT, gdzie urządzenia peryferyjne uzyskują zdolność do przetwarzania i autonomicznej komunikacji w ramach sieci. Z inżynierskiego punktu widzenia, kamera IP jest zaawansowanym systemem wbudowanym, łączącym optykę, cyfrowe przetwarzanie sygnału, kompresję danych oraz kompleksowy stos protokołów sieciowych.

## 1.1    Zastosowanie Kamer IP

Tabela 1.1: Główne obszary zastosowań kamer IP (na podstawie raportu Hanwha Vision, 2025)

Obszar zastosowania	Przykłady wykorzystania kamer IP
Bezpieczeństwo publiczne	Monitorowanie ulic, placów, obiektów strategicznych; automatyczne wykrywanie zagrożeń i incydentów.
Transport i logistyka	Monitoring lotnisk, dworców, portów; analiza przepływu pasażerów; automatyczne rozpoznawanie tablic rejestracyjnych.
Przemysł	Kontrola procesów produkcyjnych, wykrywanie awarii maszyn, nadzór nad pracownikami i bezpieczeństwem pracy.
Handel detaliczny	Zapobieganie kradzieżom, analiza zachowań klientów, optymalizacja układu sklepu.
Edukacja	Zwiększanie bezpieczeństwa uczniów i nauczycieli, kontrola dostępu do budynków szkolnych.
Ochrona zdrowia	Nadzór nad pacjentami i personelem, zabezpieczenie pomieszczeń szpitalnych, kontrola dostępu do stref wrażliwych.
Smart City	Analiza ruchu drogowego, inteligentne sterowanie sygnalizacją świetlną, planowanie urbanistyczne na podstawie danych z kamer.

Zastosowanie monitoringu wizyjnego opartego na kamerach IP jest obecnie wielosektorowe i dynamiczne. Urządzenia te, integrujące funkcje sensora i procesora danych, stały się podstawą **systemów analitycznych** w kluczowych obszarach gospodarki i bezpieczeństwa. W kontekście dalszego rozwoju monitoringu wizyjnego, szczególne znaczenie zyskuje **sztuczna inteligencja (AI)** i **uczenie maszynowe (ML)**. Nowoczesne algorytmy pozwalają na automatyczną detekcję zagrożeń, eliminację fałszywych alarmów oraz identyfikację i śledzenie obiektów w czasie rzeczywistym. Integracja tych zaawansowanych technik z **otwartym oprogramowaniem** — co jest celem niniejszej pracy — otwiera drogę do stworzenia bardziej zaawansowanych, konfigurowalnych i niezależnych narzędzi wspierających bezpieczeństwo oraz analitykę zdarzeń. Dzięki temu, system monitorujący może automatycznie filtrować szum wizualny i koncentrować uwagę na zdarzeniach o wysokim prawdopodobieństwie zagrożenia lub anomalii. Technologie te transformują surowe dane wideo w zorganizowane i użyteczne metadane, co jest fundamentalne dla automatyki i bezpieczeństwa.

### 1.1.1 Monitoring

Podstawowym i historycznym zastosowaniem kamery IP jest **nadzór wizyjny (monitoring)**. W odróżnieniu od analogowego CCTV, monitoring oparty na protokole internetowym umożliwia przesyłanie strumienia wideo wysokiej rozdzielczości (np. 1080p w Tapo C200) oraz metadanych poprzez standardowe sieci LAN/WLAN. Z technicznego punktu widzenia, monitoring realizowany jest poprzez ciągłe kodowanie wideo (standardy H.264/H.265), strumieniowanie za pomocą protokołów czasu rzeczywistego (**RTSP**) oraz zapis cyfrowy na nośnikach lokalnych (microSD, serwer NVR) lub w chmurze.

### 1.1.2 Kontrola Dostępu

Kamery IP są coraz częściej integrowane z systemami **Kontroli Dostępu (Access Control Systems - ACS)**. Ich rola wykracza poza zwykłe weryfikowanie tożsamości. Dzięki wykorzystaniu AI, kamery stają się kluczowym sensorem w bezdotykowej automatyzacji. Przykłady zastosowań inżynierskich obejmują:

1. **Rozpoznawanie Twarzy (Facial Recognition):** Zastosowanie algorytmów głębokiego uczenia do identyfikacji i weryfikacji osób uprawnionych, automatycznie odblokowując wejścia.
2. **Rozpoznawanie Tablic Rejestracyjnych (ANPR):** Automatyczne zezwalanie na wjazd pojazdów do strzeżonych stref (np. parkingów pracowniczych) na podstawie analizy obrazu z kamery.

Takie rozwiązania minimalizują ryzyko błędów ludzkich i zwiększają bezpieczeństwo

poprzez ciągłe logowanie zdarzeń wejścia i wyjścia, stanowiąc integralną część zabezpieczeń fizycznych i sieciowych Bou-Harb, Guevara, Alasmay i in., 2024.

### 1.1.3 Zarządzanie Procesami Biznesowymi

Wykorzystanie kamer IP w zarządzaniu procesami biznesowymi koncentruje się na optymalizacji operacyjnej poprzez zbieranie danych o efektywności i bezpieczeństwie pracy. W sektorach takich jak produkcja i logistyka, kamery są używane do:

1. **Kontroli Jakości (Quality Assurance - QA):** Monitorowanie linii produkcyjnych w celu automatycznego wykrywania defektów, niezgodności montażu lub nieprawidłowej sekwencji działań.
2. **Optymalizacji Przepływu Pracy (Workflow Optimization):** Analiza ścieżek ruchu pracowników i pojazdów w celu identyfikacji wąskich gardeł w magazynach i centrach dystrybucyjnych.

Te zastosowania wymagają wysokiej precyzji metadanych i niskiego opóźnienia, co stawia wysokie wymagania przed **algorytmami analizy brzegowej (Edge Analytics)**, które muszą działać na poziomie procesora kamery lub serwera lokalnego Abdalla, 2020.

### 1.1.4 Technologie Smart

Kamery IP są fundamentalnym elementem **ekosystemów Smart Home i Smart City**. W tych kontekstach, kamera pełni rolę czujnika behawioralnego, dostarczając danych do zautomatyzowanych systemów decyzyjnych. W budownictwie inteligentnym, Tapo C200, podobnie jak inne urządzenia IoT, jest zintegrowana za pomocą protokołów API z platformami takimi jak **Google Assistant i Amazon Alexa** (jak wskazano w dokumentacji Tapo). Przykłady zastosowań to:

1. **Automatyzacja Zdarzeniowa:** Detekcja ruchu lub dźwięku (np. wykrywanie płaczu dziecka w Tapo C200) uruchamia inne urządzenia (np. włącza światło, wysyła alert do systemu zarządzania domem).
2. **Zarządzanie Energią:** Wykrycie braku obecności osób w pomieszczeniu może prowadzić do automatycznego obniżenia temperatury lub wyłączenia niepotrzebnych urządzeń, przyczyniając się do zwiększenia efektywności energetycznej.

### 1.1.5 Analiza Danych

Kamera IP, połączona ze sztuczną inteligencją (AI), przestaje być pasywnym urządzeniem rejestrującym, a staje się aktywnym sensorem generującym **metadane struk-**

**turalne.** W kontekście systemów Big Data, strumień wideo jest intensywnie przetwarzany, stanowiąc bazę dla analityki w czasie rzeczywistym i prognozowania zdarzeń. Efektywne wykorzystanie danych wizyjnych do celów analitycznych obejmuje trzy główne poziomy inżynierskie:

1. **Ekstrakcja Danych Statystycznych:** Dotyczy pomiarów ilościowych, takich jak gęstość obiektów, liczenie przepływu (*flow counting*) oraz generowanie map ciepła (*heatmaps*) Minerva, Biru i Rotondi, 2021.
2. **Analiza Behavioralna i Wzorce Trendów:** Identyfikacja nietypowych sekwencji zdarzeń, które mogą sugerować incydent bezpieczeństwa (np. pozostawiony pakunek) **Al-Fuqaha:2015**.
3. **Analityka Predykcyjna (Predictive Analytics):** Przewidywanie potencjalnych przyszłych zdarzeń na podstawie historycznych i bieżących metadanych. Wymaga to integracji i walidacji danych z wielu źródeł IoT Alaba i in., 2017.

Możliwość pełnej i niezależnej **Analizy Danych (Data Analytics)** jest ściśle powiązana z problemem *vendor lock-in*. Uwolnienie strumienia z kamery Tapo C200 jest podstawowym warunkiem inżynierskim dla realizacji zaawansowanej analityki danych.

## 1.2 Budowa

Zrozumienie architektury kamery IP jest kluczowe dla identyfikacji ograniczeń narzuconych przez producentów i zaprojektowania skutecznego, otwartego oprogramowania.

Z perspektywy inżynierskiej, kamera IP nie jest monolitycznym urządzeniem, lecz złożonym systemem wbudowanym, składającym się ze ściśle zintegrowanych komponentów sprzętowych (hardware) i dedykowanego oprogramowania (firmware), które zarządza ich pracą. Poniższe podrozdziały szczegółowo omawiają te dwie warstwy.

### 1.2.1 Budowa Fizyczna - Hardware

Warstwa sprzętowa stanowi fizyczny fundament kamery, odpowiadając za akwizycję, przetwarzanie i transmisję danych audiowizualnych. W celu dogłębnej analizy, jej kluczowe komponenty zostaną omówione w dedykowanych podrozdziałach.

**1.2.1.1 Matryca (Przetwornik obrazu)** Matryca, nazywana również przetwornikiem lub sensorem obrazu, jest kluczowym elementem półprzewodnikowym, który inicjuje cały proces wizyjny. Jej fundamentalnym zadaniem jest konwersja energii fotonów (światła) padających na jej powierzchnię na mierzalny sygnał elektryczny. Proces ten, znany jako wewnętrzne zjawisko fotoelektryczne, stanowi podstawę cyfrowego przetwarzania obrazu Howe i El-Ghoroury, 2014. We współczesnych kamerach IP, w tym

w analizowanym modelu TP-Link Tapo C200, dominującą technologią jest \*\*CMOS (Complementary Metal-Oxide-Semiconductor)\*\*. Przetworniki CMOS wyparły starszą technologię CCD (Charge-Coupled Device) głównie ze względu na niższy koszt produkcji, mniejsze zużycie energii oraz możliwość integracji dodatkowych obwodów logicznych (np. przetworników analogowo-cyfrowych, układów redukcji szumów) bezpośrednio na tej samej płytce krzemowej, co jest zgodne z architekturą System-on-a-Chip (SoC) Fossum, 1997. Z inżynierskiego punktu widzenia, jakość generowanego obrazu jest determinowana przez następujące parametry techniczne matrycy:

- **Rozdzielczość (Resolution):** Określa liczbę pikseli, z których składa się obraz, np. 1920x1080 (Full HD) w kamerze Tapo C200. Wyższa rozdzielczość pozwala na zarejestrowanie większej liczby szczegółów, ale jednocześnie generuje większy strumień danych, co stanowi wyzwanie dla przepustowości sieci i zasobów obliczeniowych.
- **Rozmiar fizyczny i rozmiar piksela:** Fizyczny rozmiar matrycy (np. 1/2.9 cala) w połączeniu z jej rozdzielczością definiuje rozmiar pojedynczego piksela. Większe piksele są w stanie przechwycić więcej światła, co przekłada się na lepszą jakość obrazu w warunkach słabego oświetlenia i wyższy stosunek sygnału do szumu (SNR).
- **Czułość (Sensitivity):** Mierzona w luksach (lux), określa minimalną ilość światła potrzebną do wygenerowania użytecznego obrazu. Jest to parametr krytyczny dla funkcji noktowizyjnych, gdzie matryca musi efektywnie współpracować z oświetlaczem podczerwieni (IR).
- **Zakres dynamiki (Dynamic Range):** Zdolność matrycy do jednoczesnego rejestrowania bardzo jasnych i bardzo ciemnych obszarów w tej samej scenie. Szeroki zakres dynamiki (WDR) jest kluczowy w scenach o dużym kontraście, np. w pomieszczeniu z oknem w słoneczny dzień.

Zrozumienie tych parametrów jest niezbędne do oceny ograniczeń sprzętowych kamery i świadomego projektowania algorytmów przetwarzania obrazu, takich jak detekcja ruchu, które muszą operować na danych wyjściowych dostarczanych przez matrycę.

### 1.2.2 Oprogramowanie - Firmware

Firmware to dedykowane oprogramowanie wbudowane w pamięć Flash kamery, które pełni rolę systemu operacyjnego i warstwy aplikacyjnej. Jest ono "pomostem" między fizycznym sprzętem a funkcjonalnością dostępną dla użytkownika **Arshon:Firmware**. Firmware realizuje kluczowe zadania, takie jak:

- **Inicjalizacja sprzętu (Bootloader):** Pierwszy program uruchamiany po włączeniu zasilania, który testuje i konfiguruje wszystkie komponenty sprzętowe.
- **Zarządzanie procesami:** Alokacja zasobów CPU i pamięci dla zadań takich jak kompresja wideo (np. do formatu H.264), obsługa strumienia RTSP, czy analiza obrazu.
- **Obsługa stosu sieciowego:** Implementacja protokołów komunikacyjnych (TCP/IP, Wi-Fi, HTTP), które umożliwiają połączenie z siecią lokalną i internetem.
- **Interfejs API:** Udostępnienie (lub, jak w przypadku Tapo, ukrycie) interfejsu programistycznego, który pozwala na sterowanie kamerą.

## 1.3 Zasada działania

Niniejszy podrozdział stanowi dogłębną analizę mechanizmów operacyjnych, które definiują funkcjonalność nowoczesnej kamery IP. Urządzenie to, dalekie od bycia prostym peryferium, jest w rzeczywistości zaawansowanym, autonomicznym systemem wbudowanym, w którym zbiegają się dziedziny inżynierii sprzętowej, oprogramowania firmware oraz złożonych protokołów sieciowych. Celem tej analizy jest dekonstrukcja logiki działania kamery na cztery fundamentalne, wzajemnie powiązane domeny. Rozpocniemy od zbadania jej architektury komunikacyjnej, czyli stosu protokołów, który umożliwia jej funkcjonowanie jako węzła w sieci. Następnie przeanalizujemy krytyczny proces provisioningu, czyli bezpiecznego włączania urządzenia do infrastruktury sieciowej. W trzeciej części prześledzimy wewnętrzny potok przetwarzania danych, od momentu konwersji zjawisk fizycznych – fotonów światła i fal akustycznych – na surowe dane cyfrowe, aż po ich kompresję do formatu gotowego do transmisji. Na końcu szczegółowo omówimy mechanizmy strumieniowania, które pozwalają na przesyłanie tych danych w czasie rzeczywistym. Zrozumienie synergii między wyspecjalizowanym układem scalonym (SoC), sensorami, oprogramowaniem układowym i protokołami sieciowymi jest kluczowe do pełnego pojęcia, jak kamera IP realizuje swoje zadania – od prostego monitoringu po zaawansowaną analitykę danych.

### 1.3.1 Architektura Komunikacji Sieciowej: Stos Protokołów i Standardy

Zdolność kamery IP do komunikacji z innymi urządzeniami i systemami jest jej cechą definiującą, odróżniającą ją od tradycyjnych systemów analogowych. Ta komunikacja nie jest monolitycznym procesem, lecz jest zarządzana przez hierarchiczny zbiór reguł i protokołów, zorganizowanych w warstwową architekturę. Zrozumienie tej architektury jest fundamentalne dla pojęcia, w jaki sposób kamera wysyła polecenia, odbiera konfigurację i transmituje strumień wideo.

**1.3.1.1 Fundamenty: Czworwarstwowy Model TCP/IP** Podstawą całej komunikacji w internecie, a co za tym idzie, również w kamerach IP, jest model TCP/IP. Jest to uniwersalna, sprawdzona w praktyce struktura, która dzieli złożone zadanie komunikacji sieciowej na cztery zarządzalne, abstrakcyjne warstwy. Każda warstwa odpowiada za określony zestaw funkcji i komunikuje się wyłącznie z warstwami bezpośrednio powyżej i poniżej niej. Taka modularna budowa upraszcza projektowanie systemów sieciowych i zapewnia interoperacyjność między urządzeniami różnych producentów.

- **Warstwa Dostępowa (Link Layer):** Najniższa warstwa, odpowiedzialna za fizyczne przesyłanie danych w ramach jednej sieci lokalnej (np. przez Wi-Fi lub kabel Ethernet).
- **Warstwa Internetowa (Internet Layer):** Odpowiada za adresowanie i routing pakietów danych pomiędzy różnymi sieciami, umożliwiając komunikację globalną.
- **Warstwa Transportowa (Transport Layer):** Zarządza komunikacją między konkretnymi aplikacjami na urządzeniach końcowych, zapewniając niezawodność lub szybkość transmisji.
- **Warstwa Aplikacji (Application Layer):** Najwyższa warstwa, zawierająca protokoły, z którymi bezpośrednio interaguje oprogramowanie kamery i aplikacje klienckie (np. protokoły do strumieniowania wideo czy sterowania).

Wszystkie protokoły i standardy omówione w dalszej części tego podrozdziału działają w ramach jednej z tych czterech warstw, tworząc spójny i funkcjonalny stos komunikacyjny.

**1.3.1.2 Warstwa Dostępowa (Link Layer): Fizyczne Połączenie i Bezpieczeństwo** Ta warstwa stanowi fizyczny fundament komunikacji, definiując, w jaki sposób bity danych są przesyłane przez medium transmisyjne. W przypadku konsumenckich kamer IP, takich jak analizowany model Tapo, dominującym medium jest sieć bezprzewodowa.

**Standardy IEEE 802.11 (Wi-Fi)** Kamery IP powszechnie wykorzystują standardy z rodziny IEEE 802.11, znane komercyjnie jako Wi-Fi. Najczęściej spotykane w urządzeniach IoT są standardy 802.11n (Wi-Fi 4) oraz 802.11ac (Wi-Fi 5). Standardy te określają kluczowe parametry transmisji, takie jak techniki modulacji sygnału, wykorzystywane pasma częstotliwości oraz maksymalne teoretyczne przepustowości.

- **IEEE 802.11n (Wi-Fi 4):** Jest to starszy, ale wciąż bardzo popularny standard w urządzeniach IoT ze względu na niższy koszt implementacji. Może on operować zarówno w zatłoczonym paśmie 2.4 GHz, jak i w mniej obciążonym paśmie 5 GHz, oferując przepustowości wystarczające dla strumieni wideo w rozdzielczości Full HD.



- **IEEE 802.11ac (Wi-Fi 5):** Nowszy standard, działający wyłącznie w paśmie 5 GHz, co zapewnia mniejsze zakłócenia od innych urządzeń (np. kuchenek mikrofalowych, telefonów bezprzewodowych). Oferuje znacznie wyższe prędkości i jest preferowany w środowiskach o dużej gęstości sieci bezprzewodowych.

Wybór odpowiedniego standardu i pasma ma bezpośredni wpływ na stabilność i jakość połączenia kamery, co jest kluczowe dla nieprzerwanego monitoringu.

**Mechanizmy Bezpieczeństwa (WPA2/WPA3)** Ponieważ dane przesyłane są bezprzewodowo, ich zabezpieczenie przed nieautoryzowanym dostępem jest absolutnie krytyczne. Służą do tego protokoły szyfrowania.

- **WPA2 (Wi-Fi Protected Access 2):** Przez wiele lat był to złoty standard bezpieczeństwa sieci bezprzewodowych. WPA2 wykorzystuje silny algorytm szyfrowania AES (Advanced Encryption Standard), który zapewnia solidną ochronę danych. Jest on powszechnie stosowany i kompatybilny z praktycznie wszystkimi urządzeniami.
- **WPA3 (Wi-Fi Protected Access 3):** Jest to najnowszy i najbezpieczniejszy standard, wprowadzony w 2018 roku. WPA3 oferuje kilka kluczowych ulepszeń w stosunku do WPA2, w tym silniejszą ochronę przed atakami typu "brute-force" na hasła (dzięki mechanizmowi Simultaneous Authentication of Equals) oraz zapewnienia tzw. "forward secrecy", co oznacza, że nawet jeśli atakujący zdobędzie hasło do sieci, nie będzie w stanie odszyfrować przechwyconej wcześniej komunikacji. Chociaż WPA3 jest standardem preferowanym, niektóre starsze urządzenia IoT mogą mieć problemy z kompatybilnością, co zmusza do korzystania z trybu mieszanego WPA2/WPA3 lub pozostania przy WPA2. Zabezpieczenie transmisji na tej warstwie jest pierwszą i fundamentalną linią obrony przed podsłuchem i nieautoryzowanym dostępem do strumienia wideo z kamery.

**1.3.1.3 Warstwa Internetowa (Internet Layer): Adresacja i Routing** Gdy kamera uzyska bezpieczne połączenie z siecią lokalną, warstwa internetowa wchodzi do gry, aby nadać jej unikalny adres i umożliwić komunikację z urządzeniami spoza jej bezpośredniego otoczenia.

**Protokół IP i Adresacja** Sercem tej warstwy jest Protokół Internetowy (IP), którego głównym zadaniem jest przypisanie każdemu urządzeniu w sieci unikalnego adresu IP. Adres ten działa jak adres pocztowy, pozwalając na precyzyjne kierowanie (routing) pakietów danych do i z kamery.

**Dynamiczna vs. Statyczna Konfiguracja IP** Kamera może uzyskać swój adres IP na dwa sposoby:

- **DHCP (Dynamic Host Configuration Protocol):** Jest to metoda domyślna w większości sieci domowych i konsumenckich. Po podłączeniu do sieci, kamera automatycznie wysyła zapytanie do routera, który z puli dostępnych adresów przydziela jej jeden na określony czas. Jest to rozwiązanie wygodne i nie wymagające konfiguracji przez użytkownika. Jego wadą jest to, że adres IP kamery może się zmienić po jej ponownym uruchomieniu, co może utrudnić dostęp do niej z innych systemów.
- **Statyczny adres IP:** W bardziej profesjonalnych zastosowaniach często konfiguruje się kamerę ze stałym, niezmiennym adresem IP. Użytkownik ręcznie przypisuje adres spoza puli DHCP routera. Gwarantuje to, że kamera będzie zawsze dostępna pod tym samym, znanym adresem, co jest kluczowe dla integracji z systemami NVR (Network Video Recorder) czy platformami automatyki domowej.

**1.3.1.4 Warstwa Transportowa (Transport Layer): Niezawodność kontra Szybkość** Warstwa transportowa odpowiada za komunikację pomiędzy konkretnymi procesami działającymi na kamerze i na urządzeniu klienckim. To na tym poziomie podejmowana jest fundamentalna decyzja architektoniczna dotycząca sposobu przesyłania danych, która determinuje charakter całej komunikacji. Kamera IP musi obsługiwać dwa diametralnie różne typy komunikacji, co prowadzi do swoistej "podwójnej osobowości" jej stosu sieciowego. Z jednej strony musi działać jak niezawodny serwer webowy, a z drugiej – jak stacja nadawcza czasu rzeczywistego. Do obsługi tych dwóch ról wykorzystywane są dwa różne protokoły warstwy transportowej: TCP i UDP.

**TCP (Transmission Control Protocol)** TCP jest protokołem połączeniowym, co oznacza, że przed przesłaniem jakichkolwiek danych, nawiązuje formalną sesję między klientem a serwerem za pomocą trójetapowego procesu "uścisku dłoni" (three-way handshake). Jego główną cechą jest niezawodność. TCP gwarantuje, że każdy wysłany pakiet dotrze do odbiorcy, a pakiety zostaną złożone w prawidłowej kolejności. Osiąga to poprzez mechanizmy potwierdzeń (acknowledgements) i retransmisji utraconych pakietów. Ta niezawodność ma jednak swoją cenę – dodatkowe dane kontrolne i potencjalne opóźnienia związane z retransmisjami sprawiają, że TCP jest wolniejszy od UDP.

W kamerze IP, TCP jest absolutnie niezbędny do zadań, gdzie integralność danych jest krytyczna. Są to między innymi:

- **Dostęp do interfejsu konfiguracyjnego:** Gdy użytkownik łączy się z kamerą

przez przeglądarkę internetową, cała komunikacja odbywa się za pomocą protokołu HTTP, który działa na bazie TCP.

- **Wysyłanie poleceń sterujących:** Komendy wysyłane przez API, takie jak obrót kamery (PTZ), zmiana ustawień obrazu czy aktywacja trybu nocnego, muszą dotrzeć w całości i bezbłędnie. Utrata nawet jednego pakietu w poleceniu "obróć do lewo" mogłaby spowodować nieprzewidywalne zachowanie urządzenia. Dlatego te funkcje również opierają się na niezawodności TCP.

**UDP (User Datagram Protocol)** UDP jest protokołem bezpołączeniowym, często określanym jako "best-effort". Nie nawiązuje on formalnej sesji i nie gwarantuje dostarczenia pakietów ani ich prawidłowej kolejności. Po prostu wysyła datagramy do odbiorcy, nie czekając na potwierdzenie. Brak tych mechanizmów kontrolnych sprawia, że UDP ma znacznie mniejszy narzut (mniejsze nagłówki) i charakteryzuje się bardzo niskimi opóźnieniami, co czyni go idealnym do zastosowań czasu rzeczywistego.

W kamerze IP, UDP jest protokołem z wyboru do transportu głównego ładunku – strumienia audio i wideo. W przypadku transmisji na żywo, utrata pojedynczej klatki wideo (jednego lub kilku pakietów) jest zazwyczaj niezauważalna dla ludzkiego oka. Znacznie gorszym doświadczeniem dla użytkownika byłoby zatrzymanie obrazu na kilkaset milisekund w oczekiwaniu na retransmisję utraconego pakietu, co miałoby miejsce przy użyciu TCP. Dlatego strumieniowanie mediów jest realizowane za pomocą protokołu RTP, który niemal zawsze działa na bazie szybkiego, ale zawodnego UDP.

Ta dwoistość architektoniczna jest kluczowa dla zrozumienia działania kamery. Jej oprogramowanie firmware musi jednocześnie zarządzać stabilnymi, niezawodnymi połączeniami TCP dla poleceń i konfiguracji, oraz emitować ciągły, wysokonkładowy strumień datagramów UDP z danymi audiowizualnymi. Efektywne zarządzanie tymi dwoma trybami komunikacji przez procesor SoC jest jednym z głównych wyzwań inżynierskich w projektowaniu wydajnych urządzeń IoT.

Tabela 1.2: Porównanie protokołów TCP i UDP w kontekście kamery IP.

Cecha	Protokół TCP ( <i>Transmission Control Protocol</i> )	Protokół UDP ( <i>User Datagram Protocol</i> )
Typ Połączenia	Połączeniowy (wymaga nawiązania sesji)	Bezpołączeniowy (wysyła dane bez nawiązywania sesji)
Niezawodność	Gwarantowana dostawa, kontrola błędów i kolejności	„Best-effort” (brak gwarancji dostawy i kolejności)
Prędkość	Wolniejszy (większy narzut, retransmisje)	Szybszy (minimalny narzut, brak retransmisji)
Nagłówek	Większy (20 bajtów)	Mniejszy (8 bajtów)
Główne Zastosowanie w Kmerze IP	Sterowanie i konfiguracja (HTTP API), dostęp webowy	Strumieniowanie mediów w czasie rzeczywistym (RTP)

**1.3.1.5 Warstwa Aplikacji (Application Layer): Usługi Sieciowe i Sterowanie** Najwyższa warstwa stosu TCP/IP zawiera protokoły, które realizują konkretne funkcje widoczne dla użytkownika i innych systemów. To tutaj logika biznesowa kamery jest wystawiana na zewnątrz w postaci usług sieciowych.

**HTTP API (Application Programming Interface)** Wiele kamer IP, zwłaszcza tych przeznaczonych do integracji, udostępnia interfejs programistyczny aplikacji (API) oparty na protokole HTTP. Działa on jak zdalny panel sterowania, umożliwiając autoryzowanym aplikacjom wysyłanie żądań HTTP (np. GET lub POST) do określonych adresów URL na serwerze webowym kamery w celu wykonania akcji lub odczytania stanu. Przykładowo, wysłanie żądania do `http://<adres_ip_kamery>/api/ptz?action=pan_left&speed=50` mogłoby spowodować obrót kamery w lewo z określoną prędkością. Takie API jest kluczowe dla integracji kamery z zewnętrznymi systemami, takimi jak platformy automatyki domowej (np. Home Assistant) czy niestandardowe oprogramowanie do zarządzania wideo. Jest to mechanizm, który projekt opisany w niniejszej pracy inżynierskiej ma na celu wykorzystać do sterowania kamerą Tapo.

**NTP (Network Time Protocol)** Choć często pomijany, NTP odgrywa fundamentalną rolę w prawidłowym funkcjonowaniu kamery IP. Jest to protokół służący do synchronizacji wewnętrznego zegara urządzenia z wysoce precyzyjnymi, globalnymi ser-

werami czasu. Dokładny i zsynchronizowany czas jest niezbędny z kilku powodów:

- **Prawidłowe znaczniki czasu (timestamps):** Każda klatka wideo i fragment audio muszą być opatrzone precyzyjnym znacznikiem czasu. Jest to kluczowe dla synchronizacji obrazu z dźwiękiem podczas odtwarzania oraz dla analizy zdarzeń, gdzie dokładna kolejność i czas ich wystąpienia mają znaczenie kryminalistyczne.
- **Logowanie zdarzeń:** Wszelkie zdarzenia systemowe, takie jak wykrycie ruchu, utrata połączenia czy próby logowania, są zapisywane w logach systemowych. Spójny czas we wszystkich logach w sieci jest niezbędny do diagnostyki i analizy bezpieczeństwa.
- **Bezpieczeństwo:** Wiele mechanizmów bezpieczeństwa, takich jak walidacja certyfikatów cyfrowych (używanych np. w HTTPS), opiera się na sprawdzaniu, czy bieżąca data i godzina mieszczą się w okresie ważności certyfikatu. Błędnie ustawiony zegar mógłby uniemożliwić kamerze nawiązywanie bezpiecznych połączeń.

Kamera okresowo wysyła zapytania do serwerów NTP, aby skorygować ewentualne dryfowanie swojego wewnętrznego zegara, zapewniając dokładność rzędu milisekund w sieciach lokalnych.

### 1.3.2 Provisioning: Inicjalizacja i Uwierzytelnianie Urządzenia

Zanim nowa kamera IP stanie się funkcjonalnym i zaufanym elementem sieci, musi przejść przez krytyczny proces zwany provisioningiem. Nie jest to jedynie techniczna konfiguracja, ale fundamentalny proces ustanawiania cyfrowej tożsamości urządzenia i zakotwiczenia zaufania, który przekształca anonimowy sprzęt prosto z pudełka w zweryfikowany i bezpieczny węzeł sieciowy.

**1.3.2.1 Definicja i Cel Provisioningu** Provisioning (w polskim kontekście często nazywany inicjalizacją, udostępnianiem lub aprowizacją) to kompleksowy proces bezpiecznego wprowadzania nowego urządzenia do środowiska sieciowego. Obejmuje on cały cykl życia, od pierwszego uruchomienia, poprzez konfigurację, uwierzytelnienie, aż po zarządzanie operacyjne i ewentualne wycofanie z użytku. Głównym celem provisioningu jest zapewnienie, że tylko autoryzowane, bezpieczne i prawidłowo skonfigurowane urządzenia uzyskują dostęp do sieci i jej zasobów. Jest to pierwsza i najważniejsza linia obrony w ekosystemie Internetu Rzeczy (IoT), gdzie potencjalnie miliony urządzeń mogą stanowić wektor ataku, jeśli nie zostaną prawidłowo zweryfikowane.

Proces ten opiera się na fundamentalnej zasadzie bezpieczeństwa znanej jako Zero Trust. Sieć nie ufa żadnemu urządzeniu domyślnie, nawet jeśli znajduje się ono fizycz-

nie w jej zasięgu. Każde urządzenie musi najpierw udowodnić swoją tożsamość i uzyskać autoryzację, zanim zostanie dopuszczone do komunikacji. W przypadku kamer IP, których strumień wideo jest daną wrażliwą, solidny proces provisioningu jest absolutnie kluczowy.

**1.3.2.2 Przykładowy Proces Provisioningu dla Kamery Wi-Fi** Proces provisioningu dla typowej konsumenckiej kamery Wi-Fi, takiej jak modele TP-Link Tapo, jest zaprojektowany tak, aby był jak najprostszy dla użytkownika końcowego, jednocześnie realizując niezbędne kroki bezpieczeństwa. Zazwyczaj odbywa się on za pośrednictwem dedykowanej aplikacji mobilnej producenta i można go podzielić na trzy główne etapy.

**1. Rejestracja (Enrollment)** Pierwszym krokiem jest zarejestrowanie fizycznego urządzenia w systemie zarządzania producenta.

- **Tryb Access Point (AP):** Po pierwszym podłączeniu do zasilania, kamera nie próbuje łączyć się z żadną istniejącą siecią. Zamiast tego, uruchamia własną, tymczasową sieć Wi-Fi o niewielkim zasięgu, działając w trybie punktu dostępowego (Access Point). Ta sieć jest zazwyczaj otwarta lub zabezpieczona prostym, domyślnym hasłem.
- **Połączenie z Aplikacją:** Użytkownik, postępując zgodnie z instrukcjami w aplikacji mobilnej, łączy swój smartfon z tą tymczasową siecią Wi-Fi emitowaną przez kamerę. W tym momencie smartfon i kamera znajdują się w tej samej, izolowanej sieci, co pozwala aplikacji na bezpośrednie "odkrycie" kamery i nawiązanie z nią bezpiecznej komunikacji.
- **Identyfikacja Urządzenia:** Aplikacja odczytuje unikalny identyfikator sprzętowy kamery (np. adres MAC lub numer seryjny) i rejestruje go na koncie użytkownika w chmurze producenta. Ten krok formalnie przypisuje to konkretne urządzenie do tego konkretnego użytkownika.

**2. Konfiguracja (Configuration)** Po nawiązaniu bezpośredniego połączenia, aplikacja mobilna przekazuje kamerze niezbędne dane konfiguracyjne, aby mogła ona funkcjonować w docelowej sieci.

- **Przekazanie Poświadczeń Sieciowych:** Najważniejszym elementem tego etapu jest bezpieczne przekazanie kamerze nazwy (SSID) i hasła do domowej sieci Wi-Fi użytkownika. Aplikacja szyfruje te dane i wysyła je bezpośrednio do kamery.
- **Ustawienia Dodatkowe:** W tym kroku mogą być również przekazywane inne ustawienia, takie jak nazwa kamery (np. "Salon"), strefa czasowa, a także może

zostać zainicjowana automatyczna aktualizacja oprogramowania firmware do najnowszej wersji.

**3. Uwierzytelnianie (Authentication)** Jest to kulminacyjny i najważniejszy z punktu widzenia bezpieczeństwa etap provisioningu.

- **Restart i Połączenie z Siecią Docelową:** Po otrzymaniu konfiguracji, kamera kończy działanie w trybie AP i restartuje się. Następnie próbuje połączyć się z domową siecią Wi-Fi, używając otrzymanych poświadczeń.
- **Uwierzytelnianie w Chmurze:** Po pomyślnym połączeniu z siecią lokalną i uzyskaniu dostępu do internetu, kamera nawiązuje połączenie z serwerami chmurowymi producenta. W tym momencie następuje kluczowy proces uwierzytelniania. Kamera przedstawia serwerowi swój unikalny, wbudowany fabrycznie certyfikat cyfrowy (np. w standardzie X.509). Ten certyfikat działa jak niezaprzeczalny, kryptograficzny dowód tożsamości.
- **Weryfikacja i Udzielenie Dostępu:** Serwer chmurowy weryfikuje ten certyfikat, sprawdzając, czy pochodzi on z zaufanego źródła (czyli od samego producenta) i czy odpowiada urządzeniu, które zostało wcześniej zarejestrowane na koncie użytkownika. Jak zauważono w dokumentacji źródłowej, cały ten proces jest szyfrowany i wymaga weryfikacji po stronie chmury TP-Link, co świadczy o jego złożoności i krytycznym znaczeniu. Dopiero po pomyślnej weryfikacji certyfikatu, serwer uznaje kamerę za w pełni uwierzytelnioną i zaufaną. Od tego momentu kamera uzyskuje pełen dostęp do usług chmurowych (np. zdalnego podglądu, powiadomień push) i staje się w pełni funkcjonalnym urządzeniem.

Ten wieloetapowy proces, choć dla użytkownika sprowadza się do kilku kliknięć w aplikacji, jest w rzeczywistości starannie zaprojektowaną sekwencją operacji kryptograficznych i sieciowych. Wbudowany fabrycznie certyfikat pełni rolę "cyfrowego aktu urodzenia" kamery, jednoznacznie i niepodrabialnie poświadczając jej pochodzenie. Pomyślne uwierzytelnienie w chmurze jest momentem, w którym ta tożsamość zostaje oficjalnie potwierdzona, a urządzenie otrzymuje "pozwolenie na pracę" w sieci. Jest to fundamentalny mechanizm, który chroni zarówno użytkownika, jak i całą infrastrukturę IoT przed wprowadzeniem do niej fałszywych lub skompromitowanych urządzeń.

### 1.3.3 Przetwarzanie Sygnału Audiowizualnego: Od Fotonu do Pakietu Danych

Sercem kamery IP jest jej zdolność do przekształcania zjawisk fizycznych – światła i dźwięku – w ustrukturyzowany, skompresowany strumień danych cyfrowych, gotowy do transmisji przez sieć. Proces ten nie jest prostą konwersją, lecz złożonym, wieloetapowym potokiem przetwarzania (pipeline), realizowanym w czasie rzeczywistym przez

wyspecjalizowane komponenty sprzętowe wewnątrz układu System-on-a-Chip (SoC). Architektura SoC jest tu kluczowa; zamiast obciążać uniwersalny procesor (CPU) zadaniami intensywnymi obliczeniowo, deleguje je do dedykowanych, wysoce wydajnych bloków sprzętowych. Dzięki temu kamera działa nie jak tradycyjny komputer, ale jak wyspecjalizowana "rafineria danych", której jedynym celem jest nieustanne przekształcanie ogromnego strumienia surowych danych sensorycznych w zoptymalizowany, użyteczny produkt końcowy – skompresowany strumień AV.

**1.3.3.1 Ścieżka Przetwarzania Obrazu (Image Pipeline)** Droga, jaką przebywa informacja wizualna od obiektywu do interfejsu sieciowego, jest najbardziej złożonym procesem wewnątrz kamery.

**1. Akwizycja w Matrycy CMOS** Wszystko zaczyna się w przetworniku obrazu, którym w nowoczesnych kamerach jest niemal wyłącznie matryca CMOS (Complementary Metal-Oxide-Semiconductor).

- **Konwersja fotonów na ładunek:** Gdy światło przechodzi przez obiektyw, fotony uderzają w siatkę milionów światłoczułych elementów na matrycy, zwanych fotodiodami. Każda fotodioda, pod wpływem energii fotonów, generuje ładunek elektryczny, którego wielkość jest wprost proporcjonalna do intensywności padającego na nią światła.
- **Filtr Bayera:** Fotodiody same w sobie są "ślepe" na kolory – mierzą jedynie natężenie światła (luminancję). Aby uzyskać informację o kolorze, powierzchnia matrycy jest pokryta mozaiką mikroskopijnych filtrów w trzech podstawowych kolorach: czerwonym (R), zielonym (G) i niebieskim (B). Najczęściej stosowany jest tzw. filtr Bayera, w którym na każdy kwadrat 2x2 piksele przypadają dwa filtry zielone, jeden czerwony i jeden niebieski. Wynika to z faktu, że ludzkie oko jest najbardziej wrażliwe na światło zielone. W rezultacie, na wyjściu z matrycy otrzymujemy surowy, "mozaikowy" obraz, w którym każdy piksel ma informację tylko o jednym kolorze.
- **Odczyt i digitalizacja:** W przeciwieństwie do starszych matryc CCD, w technologii CMOS każda fotodioda (lub mała grupa) ma swój własny, zintegrowany wzmacniacz i obwody odczytu. Pozwala to na szybki, bezpośredni odczyt wartości ładunku z każdego piksela i jego konwersję na sygnał cyfrowy (proces A/D) jeszcze na poziomie samego sensora lub w jego bezpośrednim sąsiedztwie.

**2. Przetwarzanie w ISP (Image Signal Processor)** Surowy, zdigitalizowany obraz w formacie Bayera jest następnie przekazywany do dedykowanego koprocesora – Procesora Sygnału Obrazu (ISP). ISP to potężny, wyspecjalizowany układ, często będący



częścią głównego SoC, który w czasie rzeczywistym wykonuje serię skomplikowanych operacji w celu przekształcenia surowych danych w pełnowartościowy, estetyczny obraz wideo. Potok przetwarzania w ISP (ISP Pipeline) obejmuje następujące kluczowe etapy:

Tabela 1.3: Etapy przetwarzania w potoku ISP.

Etap	Opis
<b>Akwizycja Danych Surowych (<i>Bayer</i>)</b>	Otrzymanie zdigitalizowanego, mozaikowego obrazu z matrycy CMOS, gdzie każdy piksel reprezentuje natężenie tylko jednego z trzech kolorów ( <i>R, G lub B</i> ).
<b>Demosaicing (<i>Interpolacja Kolorów</i>)</b>	Algorytm rekonstruuje pełną informację o kolorze ( <i>RGB</i> ) dla każdego piksela poprzez interpolację brakujących wartości na podstawie kolorów sąsiednich pikseli.
<b>Redukcja Szumów</b>	Zastosowanie zaawansowanych filtrów w celu usunięcia szumu cyfrowego, który powstaje zwłaszcza przy słabym oświetleniu ( <i>wysokie ISO</i> ).
<b>Automatyczna Korekcja (<i>AWB/AE</i>)</b>	Analiza całej sceny w celu automatycznego dostosowania <b>balansu bieli</b> ( <i>AWB</i> ) dla naturalnego odwzorowania kolorów oraz <b>ekspozycji</b> ( <i>AE</i> ) dla optymalnej jasności obrazu.
<b>Ulepszanie Obrazu</b>	Zastosowanie operacji takich jak korekcja gamma, regulacja kontrastu, nasycenia kolorów oraz wyostrażanie krawędzi w celu poprawy ogólnej jakości wizualnej.
<b>Konwersja Przestrzeni Kolorów</b>	Przekształcenie obrazu z przestrzeni kolorów <i>RGB</i> na format bardziej odpowiedni do kompresji wideo, najczęściej <b>YCbCr</b> , który oddziela informację o jasności ( <i>Y</i> ) od informacji o kolorze ( <i>Cb, Cr</i> ).

Po przejściu przez potok ISP, mamy do czynienia z pełnokolorowym, skorygowanym, ale wciąż nieskompresowanym strumieniem wideo. Strumień ten, nawet dla rozdzielczości 1080p przy 30 klatkach na sekundę, ma ogromną przepływność (rzędu 1.5 Gb/s), co czyni go niemożliwym do przesłania przez typową sieć domową.

**3. Kompresja Wideo (H.264/H.265)** Ostatnim etapem przetwarzania obrazu jest jego drastyczna kompresja. Przetworzony, nieskompresowany strumień wideo (w for-

macie YCbCr) jest kierowany do kolejnego wyspecjalizowanego bloku sprzętowego w SoC – sprzętowego kodera wideo. W nowoczesnych kamerach są to kodery implementujące standardy H.264 (AVC) lub H.265 (HEVC).

- **Zasada działania:** Kodery te wykorzystują zaawansowane techniki w celu redukcji redundancji przestrzennej (wewnątrz pojedynczej klatki) i temporalnej (pomiędzy kolejnymi klatkami). Analizują obraz w poszukiwaniu podobnych bloków i zamiast przesyłać pełną informację o każdym z nich, przesyłają tylko informację o różnicach i wektorach ruchu.
- **Sprzęt vs. Oprogramowanie:** Realizacja kompresji H.264 w czasie rzeczywistym jest zadaniem niezwykle wymagającym obliczeniowo. Próba wykonania jej programowo na głównym CPU kamery byłaby zbyt wolna i energochłonna. Dlatego kluczowe jest użycie dedykowanego bloku sprzętowego, który wykonuje te operacje wielokrotnie szybciej i przy znacznie niższym zużyciu energii.

Na wyjściu z kodera otrzymujemy skompresowany elementarny strumień wideo (elementary stream), którego przepływność jest zredukowana stukrotnie lub więcej (np. do kilku Mb/s), co umożliwia jego efektywną transmisję przez sieć.

**1.3.3.2 Ścieżka Przetwarzania Dźwięku (Audio Pipeline)** Proces przetwarzania dźwięku jest mniej złożony niż obrazu, ale podąża za podobną logiką konwersji i kompresji.

**1. Akwizycja w Mikrofonie MEMS** Dźwięk jest przechwytywany przez mikrofon wykonany w technologii MEMS (Micro-Electro-Mechanical Systems). Fale dźwiękowe wprawiają w drgania miniaturową membranę wewnątrz mikrofonu. W najpopularniejszych mikrofonach pojemnościowych, te drgania zmieniają pojemność elektryczną, co jest przekształcane na analogowy sygnał elektryczny.

**2. Digitalizacja i Konwersja PDM do PCM** Współczesne mikrofony MEMS są urządzeniami wysoce zintegrowanymi i często zawierają w swojej obudowie przetwornik analogowo-cyfrowy (ADC).

- **Modulacja Sigma-Delta:** ADC w mikrofonie to zazwyczaj modulator sigma-delta, który z bardzo wysoką częstotliwością (rzędu kilku MHz) próbuje przybliżyć wartość sygnału analogowego, generując na wyjściu jednobitowy strumień danych zwany PDM (Pulse Density Modulation). Gęstość impulsów w tym strumieniu odpowiada amplitudzie oryginalnego sygnału audio.
- **Konwersja do PCM:** Strumień PDM jest następnie przesyłany do głównego układu SoC. Tam, dedykowany blok cyfrowego przetwarzania sygnałów (DSP) stosuje

filtr dolnoprzepustowy (aby usunąć szum kwantyzacji przeniesiony na wysokie częstotliwości przez modulator) i proces decymacji (zmniejszenia częstotliwości próbkowania). W rezultacie jednobitowy strumień PDM o wysokiej częstotliwości jest konwertowany na standardowy, wielobitowy (np. 16-bitowy) strumień PCM (Pulse Code Modulation) o typowej częstotliwości próbkowania dla audio (np. 8, 16 lub 44.1 kHz). PCM to nieskompresowana, cyfrowa reprezentacja dźwięku.

**3. Kompresja Audio (AAC)** Podobnie jak w przypadku wideo, surowy strumień audio PCM ma zbyt dużą przepływność do efektywnej transmisji. Jest on więc kierowany do kodera audio, który kompresuje go przy użyciu stratnego kodeka, najczęściej AAC (Advanced Audio Coding).

- **Kodowanie percepcyjne:** AAC wykorzystuje model psychoakustyczny do analizy dźwięku i usuwania tych jego składowych, które są niesłyszalne lub maskowane przez inne, głośniejsze dźwięki dla ludzkiego ucha. Pozwala to na znaczną redukcję rozmiaru danych przy minimalnej odczuwalnej utracie jakości. AAC jest standardem w wielu zastosowaniach strumieniowych, w tym na platformach takich jak YouTube czy w urządzeniach Apple, i oferuje lepszą jakość przy tej samej przepływności w porównaniu do starszego formatu MP3.

**1.3.3.3 Synchronizacja i Muksowanie** Ostatnim krokiem wewnątrz SoC, zanim dane trafią do karty sieciowej, jest połączenie oddzielnych, skompresowanych strumieni wideo (H.264) i audio (AAC) w jeden spójny strumień. Proces ten, zwany multipleksowaniem (muksowaniem), polega na przeplataniu pakietów audio i wideo w ramach jednego kontenera. Kluczowe jest przy tym osadzenie w strumieniu precyzyjnych znaczników czasu (timestamps) dla każdego pakietu, co pozwoli aplikacji klienckiej na idealne zsynchronizowanie odtwarzania obrazu i dźwięku. Po tym etapie, gotowy, zsynchronizowany strumień danych jest przekazywany do interfejsu sieciowego w celu opakowania go w pakiety RTP i wysłania w sieć.

### **1.3.4 Strumieniowanie: Transmisja Danych w Czasie Rzeczywistym**

Po przetworzeniu i skompresowaniu danych audiowizualnych, ostatnim zadaniem kamery jest ich efektywna transmisja do klienta przez sieć. Proces ten, znany jako strumieniowanie (streaming), opiera się na zestawie wyspecjalizowanych protokołów warstwy aplikacji, które zarządzają sesją i transportują dane w sposób zoptymalizowany pod kątem czasu rzeczywistego.

**1.3.4.1 Separacja Sterowania i Danych: Rola RTSP i RTP** Fundamentalną zasadą architektoniczną w strumieniowaniu na żywo jest rozdzielenie płaszczyzny sterowania (control plane) od płaszczyzny danych (data plane). Oznacza to, że protokół

używany do zarządzania sesją (np. uruchamiania i zatrzymywania strumienia) jest inny niż protokół używany do faktycznego przesyłania pakietów z wideo i audio. To rozdzielanie pozwala na optymalizację każdego z tych zadań z osobna: sterowanie wymaga niezawodności, a przesyłanie danych – szybkości i niskich opóźnień.

- **RTSP (Real-Time Streaming Protocol):** Pełni rolę "sieciowego pilota zdalnego sterowania". Jest to protokół warstwy aplikacji, który służy do nawiązywania, kontrolowania i kończenia sesji strumieniowej. Klient używa komend RTSP, aby "powiedzieć" kamerze, co ma robić – np. "zaczynaj nadawać", "zatrzymaj na chwilę" czy "zakończ transmisję". Ponieważ utrata polecenia sterującego byłaby problematyczna, komunikacja RTSP odbywa się zazwyczaj za pośrednictwem niezawodnego protokołu TCP. Co istotne, RTSP nie transportuje samych danych multimedialnych.
- **RTP (Real-time Transport Protocol):** Jest to protokół odpowiedzialny za transport danych. Jego zadaniem jest opakowanie skompresowanych danych wideo (H.264) i audio (AAC) w pakiety RTP i przesłanie ich do klienta. Aby zminimalizować opóźnienia, RTP niemal zawsze działa na bazie szybkiego protokołu UDP. Każdy pakiet RTP zawiera informacje niezbędne do prawidłowego odtworzenia strumienia po stronie klienta.

**1.3.4.2 Nawiązywanie Sesji Strumieniowej: Uścisk Dłoni RTSP** Zanim na ekranie klienta pojawi się pierwszy obraz, musi on przeprowadzić z kamerą negocjacje za pomocą protokołu RTSP. Ten proces, często nazywany "uściskiem dłoni"(handshake), przebiega w kilku krokach i jest niezbędny do ustalenia parametrów transmisji.

Tabela 1.4: Podstawowe komendy protokołu RTSP.

Komenda	Nadawca	Cel
<b>DESCRIBE</b>	Klient	Żądanie od serwera (kamery) opisu dostępnych strumieni multimedialnych. Odpowiedź zawiera dane w formacie <b>SDP</b> , informujące np. o istnieniu strumienia wideo H.264 i audio AAC.
<b>SETUP</b>	Klient	Konfiguracja transportu dla każdego strumienia z osobna. Klient informuje serwer, na których portach <b>UDP</b> będzie nasłuchiwał na pakiety <i>RTP</i> (dane) i <i>RTCP</i> (dane kontrolne).
<b>PLAY</b>	Klient	Polecenie dla serwera, aby rozpoczął transmisję pakietów <i>RTP</i> na wcześniej uzgodnione porty.
<b>PAUSE</b>	Klient	Wstrzymanie transmisji strumienia bez zrywania sesji. Sesję można wznowić komendą <b>PLAY</b> .
<b>TEARDOWN</b>	Klient	Zakończenie sesji strumieniowej i zwolnienie zasobów po stronie serwera.

Przebieg negocjacji:

1. **DESCRIBE:** Klient (np. odtwarzacz VLC) wysyła do kamery żądanie DESCRIBE, pytając o zawartość dostępną pod danym adresem RTSP (np. `rtsp://192.168.1.100/stream1`). Kamera odpowiada, wysyłając opis w formacie SDP (Session Description Protocol), który informuje klienta, że dostępne są dwa strumienie: jeden wideo (zakończony w H.264) i jeden audio (zakodowany w AAC).
2. **SETUP:** Klient, chcąc odbierać oba strumienie, wysyła dwa osobne żądania SETUP – jedno dla strumienia wideo i jedno dla audio. W każdym żądaniu SETUP klient podaje kamerze numery portów, na których będzie nasłuchiwał na przychodzące pakiety RTP (z danymi) oraz RTCP (z informacjami kontrolnymi).

3. **PLAY:** Po pomyślnym skonfigurowaniu obu strumieni, klient wysyła jedno polecenie PLAY. Jest to sygnał dla kamery, aby rozpoczęła wysyłanie pakietów RTP z danymi wideo i audio na porty wskazane przez klienta w krokach SETUP. Od tego momentu rozpoczyna się właściwe strumieniowanie.

**1.3.4.3 Transport Danych z Użyciem RTP** Gdy sesja jest już ustanowiona, kamera zaczyna wysyłać ciągły strumień pakietów RTP. Struktura tych pakietów jest kluczowa dla prawidłowego odtworzenia mediów po stronie klienta. Najważniejsze pola w nagłówku RTP to:

- **Payload Type (Typ Ładunku):** 7-bitowe pole, które identyfikuje format danych w pakiecie. Dzięki niemu klient wie, czy dany pakiet zawiera dane wideo H.264, audio AAC, czy inny typ mediów. Pozwala to na skierowanie pakietu do odpowiedniego dekodera.
- **Sequence Number (Numer Sekwencyjny):** 16-bitowy licznik, który jest inkrementowany o jeden dla każdego wysłanego pakietu RTP. To pole jest absolutnie krytyczne. Pozwala klientowi wykryć utratę pakietów (jeśli w sekwencji pojawi się luka) oraz przywrócić prawidłową kolejność pakietów, które mogły dotrzeć do celu w złej kolejności z powodu różnych dróg w sieci.
- **Timestamp (Znacznik Czasu):** 32-bitowe pole, które odzwierciedla moment próbkowania danych zawartych w pakiecie. Jest ono generowane na podstawie wewnętrznego zegara kamery. Znaczniki czasu są niezbędne do synchronizacji różnych strumieni (np. wideo i audio), do obliczania i kompensowania opóźnień sieciowych (tzw. jitter) oraz do zapewnienia płynnego odtwarzania.

**1.3.4.4 Monitorowanie Jakości Strumienia (RTCP)** Równolegle z wysyłaniem danych przez RTP, działa protokół RTCP (Real-time Transport Control Protocol). Jest to protokół towarzyszący RTP, który służy do przesyłania informacji kontrolnych i statystyk dotyczących sesji. W przeciwieństwie do jednokierunkowego przepływu danych RTP (z kamery do klienta), komunikacja RTCP jest dwukierunkowa. Klient okresowo wysyła do kamery raporty (Receiver Reports) zawierające informacje o jakości odbioru, takie jak liczba utraconych pakietów, miara jittera czy czas podróży w obie strony (round-trip time). Te informacje zwrotne są niezwykle cenne. Zaawansowana kamera lub serwer strumieniujący może na ich podstawie dynamicznie dostosowywać parametry transmisji, na przykład obniżając bitrate (jakość) strumienia wideo w odpowiedzi na wykryte przeciążenie sieci, aby zapewnić ciągłość transmisji kosztem jakości obrazu. RTCP jest więc mechanizmem zapewniającym adaptacyjność i odporność strumieniowania na zmienne warunki sieciowe.

## 1.4 Funkcje

Współczesna kamera IP jest czymś znacznie więcej niż pasywnym rejestratorem obrazu. Ewolucja technologiczna przekształciła ją w aktywne, wielofunkcyjne urządzenie sensoryczne, którego możliwości wykraczają daleko poza tradycyjny monitoring. Zdolność do zdalnego sterowania, inteligentnej analizy obrazu i dźwięku, działania w trudnych warunkach oświetleniowych oraz integracji z szerszymi ekosystemami cyfrowymi definiuje jej nowoczesną tożsamość. Niniejszy podrozdział stanowi przegląd kluczowych funkcji, które decydują o wszechstronności i wartości inżynierskiej tych urządzeń.

### 1.4.1 Obrót PTZ

Funkcjonalność PTZ (Pan-Tilt-Zoom) jest jedną z najbardziej charakterystycznych cech, która odróżnia kamery dynamiczne od statycznych. Jest to zdolność do mechanicznego sterowania polem widzenia kamery w trzech osiach, co znacząco rozszerza jej możliwości operacyjne.

- **Pan (Obrót poziomy):** Odnosi się do ruchu kamery w płaszczyźnie poziomej, od lewej do prawej, co pozwala na skanowanie szerokich panoram.
- **Tilt (Pochylenie pionowe):** Oznacza ruch w płaszczyźnie pionowej, w górę i w dół, umożliwiając obserwację obiektów na różnych wysokościach.
- **Zoom (Powiększenie):** Zdolność do zmiany ogniskowej obiektywu w celu przybliżenia lub oddalenia obrazu. Należy rozróżnić dwa typy zoomu:
  - **Zoom optyczny:** Realizowany przez fizyczny ruch soczewek w obiektywie. Zmienia on powiększenie bez utraty jakości obrazu, co jest kluczowe dla identyfikacji szczegółów z dużej odległości, takich jak twarze czy tablice rejestracyjne.
  - **Zoom cyfrowy:** Jest to w rzeczywistości powiększenie fragmentu już przechwyconego obrazu, co prowadzi do interpolacji pikseli i nieuchronnej degradacji jakości.

Od strony technicznej, mechanizm PTZ opiera się na precyzyjnych, miniaturowych silnikach krokowych, które wykonują polecenia otrzymywane z oprogramowania sterującego. Sterowanie odbywa się zdalnie za pomocą dedykowanych aplikacji, oprogramowania VMS (Video Management System) lub fizycznych kontrolerów z joystickiem. Komunikacja ta jest realizowana za pomocą różnych protokołów, od starszych standardów szeregowych jak RS-485, po nowoczesne protokoły sieciowe, takie jak ONVIF (Open Network Video Interface Forum) czy własnościowe API producenta oparte na HTTP.

### 1.4.2 Wykrywanie obiektów i zdarzeń - AI

Integracja sztucznej inteligencji (AI) i uczenia maszynowego (ML) bezpośrednio w kamerze (tzw. Edge AI) jest jedną z najważniejszych innowacji w dziedzinie monitoringu. Dzięki potężnym procesorom wbudowanym w układy SoC, kamery zyskały zdolność do analizowania obrazu w czasie rzeczywistym, przekształcając się z pasywnych rejestratorów w inteligentne sensory.

**1.4.2.1 Detekcja i klasyfikacja obiektów** W przeciwieństwie do prostej detekcji ruchu, algorytmy AI oparte na głębokich sieciach neuronowych (np. YOLO, SSD) potrafią identyfikować i klasyfikować konkretne obiekty w polu widzenia kamery. Kamera jest w stanie odróżnić człowieka od pojazdu, zwierzęcia czy poruszającej się na wietrze gałęzi. Główną korzyścią jest drastyczna redukcja fałszywych alarmów, co pozwala operatorom skupić się na realnych zagrożeniach.

**1.4.2.2 Wykrywanie zdarzeń i analiza behawioralna** Zaawansowane modele AI idą o krok dalej, rozpoznając nie tylko obiekty, ale również ich zachowania i zdarzenia. Przykłady obejmują:

- **Przekroczenie wirtualnej linii (Line Crossing):** Wykrycie obiektu przecinającego zdefiniowaną w kadrze linię.
- **Wykrywanie wtargnięcia (Intrusion Detection):** Alarmowanie, gdy obiekt wejdzie do określonej, zabronionej strefy.
- **Wykrywanie wałęsania się (Loitering Detection):** Identyfikacja osoby lub pojazdu przebywającego w danym obszarze dłużej niż ustalony czas.
- **Klasyfikacja dźwięku:** Niektóre kamery potrafią analizować również sygnał audio, rozpoznając dźwięki takie jak tłuczone szkło, krzyk czy strzał z broni palnej.

Analityka brzegowa (Edge Analytics) oznacza, że te skomplikowane obliczenia odbywają się na samej kamerze, co minimalizuje opóźnienia, zmniejsza obciążenie sieci i serwerów oraz zwiększa prywatność, ponieważ często tylko metadane (np. "wykryto osobę o godzinie 14:32") są wysyłane do chmury, a nie cały strumień wideo.

### 1.4.3 Wykrywanie ruchu

Jest to bardziej podstawowa, ale wciąż fundamentalna funkcja, dostępna w niemal każdej kamerze IP. Jej celem jest identyfikacja jakiegokolwiek zmiany w obserwowanej scenie, która może wskazywać na ruch. W przeciwieństwie do detekcji obiektów opartej na AI, tradycyjne metody detekcji ruchu są prostsze obliczeniowo i nie "rozumieją", co jest źródłem ruchu. Najczęściej stosowane są dwie techniki:



- **Różnica międzyklatkowa (Frame Differencing):** Algorytm ten porównuje kolejne klatki wideo piksel po pikselu. Jeśli różnica w wartościach pikseli w określonym obszarze przekroczy zdefiniowany próg, system uznaje to za ruch. Jest to metoda bardzo szybka, ale podatna na fałszywe alarmy spowodowane np. zmianami oświetlenia.
- **Odejmowanie tła (Background Subtraction):** Ta bardziej zaawansowana technika polega na stworzeniu statystycznego modelu tła (tego, jak scena wygląda, gdy nic się w niej nie porusza). Każda nowa klatka jest porównywana z tym modelem, a znaczące różnice są klasyfikowane jako obiekty pierwszego planu, czyli ruch. Metoda ta jest bardziej odporna na globalne zmiany oświetlenia, ale może być mylona przez powolne zmiany w tle lub poruszające się obiekty, które są jego częścią (np. falujące na wietrze drzewa).

Wykrycie ruchu jest najczęściej wykorzystywane jako wyzwalacz (trigger) dla innych akcji, takich jak rozpoczęcie nagrywania na karcie SD lub wysłanie powiadomienia push do użytkownika.

#### 1.4.4 Noktowizja i termowizja

Zdolność do "widzenia" w ciemności jest kluczową funkcją kamer bezpieczeństwa. Realizowana jest ona głównie za pomocą dwóch odrębnych technologii: noktowizji w podczerwieni oraz termowizji.

**1.4.4.1 Noktowizja w podczerwieni (IR Night Vision)** Jest to najpopularniejsza technologia stosowana w kamerach konsumenckich i profesjonalnych. Jej działanie opiera się na oświetleniu sceny za pomocą diod LED emitujących światło w paśmie bliskiej podczerwieni (IR), które jest niewidoczne dla ludzkiego oka, ale doskonale "widziane" przez matrycę kamery. Kluczowym elementem jest tutaj mechaniczny filtr odcinający podczerwień (IR Cut Filter).

- **W dzień:** Filtr jest umieszczony między obiektywem a matrycą i blokuje światło podczerwone, które zniekształciłoby kolory. Dzięki temu obraz ma naturalne, wierne barwy.
- **W nocy:** Gdy czujnik światła wykryje niski poziom oświetlenia, filtr jest mechanicznie odsuwany. Jednocześnie aktywowane są diody IR, a kamera przełącza się w tryb monochromatyczny (czarno-biały), który jest znacznie bardziej czuły na światło podczerwone. Pozwala to na uzyskanie wyraźnego obrazu nawet w całkowitej ciemności.

**1.4.4.2 Termowizja (Thermal Imaging)** Jest to zupełnie inna, bardziej zaawansowana technologia. Kamera termowizyjna nie potrzebuje żadnego źródła światła. Zamiast tego, jej specjalny sensor (mikrobolometr) wykrywa promieniowanie ciepłe (daleką podczerwień) emitowane przez wszystkie obiekty, których temperatura jest wyższa od zera absolutnego. Obraz jest tworzony na podstawie różnic temperatur – cieplejsze obiekty, takie jak ludzie czy zwierzęta, są wyraźnie widoczne na tle chłodniejszego otoczenia. Główne zalety termowizji to:

- Działanie w absolutnej ciemności i trudnych warunkach atmosferycznych (mgła, dym, deszcz).
- Wysoka skuteczność w wykrywaniu intruzów na dużych odległościach i w ukryciu (np. w zaroślach).
- Mniejsza liczba fałszywych alarmów, ponieważ nie reaguje na cienie, odbicia światła czy ruch obiektów nieożywionych.

Jednak kamery termowizyjne są znacznie droższe i zazwyczaj oferują niższą rozdzielczość, co uniemożliwia identyfikację szczegółów, takich jak rysy twarzy.

#### 1.4.5 Dwukierunkowe audio

Funkcja dwukierunkowego audio przekształca kamerę z pasywnego urządzenia nasłuchowego w interaktywny interkom. Dzięki wbudowanemu mikrofonowi i głośnikowi, użytkownik może nie tylko słyszeć dźwięk z otoczenia kamery, ale również mówić przez nią, a jego głos zostanie odtworzony przez głośnik urządzenia.

Ta dwukierunkowa komunikacja jest realizowana cyfrowo, a dane audio w obie strony są przesyłane przez tę samą sieć IP, co strumień wideo. Z technicznego punktu widzenia, implementacja tej funkcji często opiera się na protokołach Voice over IP (VoIP), takich jak SIP (Session Initiation Protocol) do nawiązywania i zarządzania sesją oraz RTP (Real-time Transport Protocol) do transportu pakietów audio w czasie rzeczywistym.

Zastosowania tej funkcji są bardzo szerokie:

- **Komunikacja:** Rozmowa z domownikami, dziećmi czy zwierzętami domowymi.
- **Weryfikacja:** Rozmowa z gościem lub kurierem stojącym przed drzwiami.
- **Odstraszanie:** Możliwość werbalnego ostrzeżenia potencjalnego intruza, co często jest skutecznym środkiem prewencyjnym.

#### 1.4.6 Zapis danych

Kamery IP oferują kilka elastycznych metod zapisu i archiwizacji materiału wideo, co pozwala dostosować rozwiązanie do konkretnych potrzeb w zakresie bezpieczeństwa, budżetu i infrastruktury sieciowej.

**1.4.6.1 Zapis lokalny na karcie microSD** Wiele kamer, zwłaszcza z segmentu konsumenckiego, jest wyposażonych w gniazdo na kartę pamięci microSD. Umożliwia to zapis nagrań bezpośrednio na urządzeniu, bez potrzeby korzystania z zewnętrznych rejestratorów czy połączenia z internetem. Jest to rozwiązanie idealne do zapisu zdarzeń wyzwalanych ruchem w lokalizacjach o ograniczonej łączności sieciowej. Główną wadą jest ryzyko utraty nagrań w przypadku kradzieży lub fizycznego uszkodzenia samej kamery.

**1.4.6.2 Rejestrator sieciowy (NVR)** Network Video Recorder (NVR) to dedykowane urządzenie w sieci lokalnej, którego zadaniem jest odbieranie strumieni wideo z wielu kamer IP i zapisywanie ich na wbudowanych dyskach twardej. NVR stanowi centralny punkt zarządzania systemem monitoringu, oferując dużą pojemność zapisu, możliwość ciągłego nagrywania 24/7 oraz zaawansowane funkcje odtwarzania i wyszukiwania. Jest to standardowe rozwiązanie w profesjonalnych systemach bezpieczeństwa.

**1.4.6.3 Zapis w chmurze (Cloud Storage)** W tym modelu strumień wideo z kamery jest przesyłany przez internet i zapisywany na serwerach dostawcy usługi. Główne zalety to:

- **Zdalny dostęp:** Nagrania są dostępne z dowolnego miejsca na świecie za pośrednictwem aplikacji mobilnej lub przeglądarki internetowej.
- **Bezpieczeństwo danych:** Materiał jest bezpieczny nawet w przypadku kradzieży lub zniszczenia kamery.
- **Brak lokalnego sprzętu:** Eliminuje potrzebę zakupu i utrzymania NVR.

Wadą tego rozwiązania jest uzależnienie od stałego połączenia z internetem, miesięczne koszty subskrypcji oraz potencjalne obawy dotyczące prywatności danych.

## **1.4.7 Integracja z Inteligentnymi Systemami**

Siła kamer IP leży w ich zdolności do bycia częścią większego, zintegrowanego ekosystemu. Otwartość protokołów sieciowych umożliwia komunikację z szeroką gamą innych urządzeń i platform programistycznych. Kluczowe technologie umożliwiające integrację to:

- **RTSP (Real-Time Streaming Protocol):** Standardowy protokół, który pozwala zewnętrznym aplikacjom i urządzeniom (np. odtwarzaczom wideo, systemom NVR) na dostęp do strumienia wideo z kamery. Jest to fundamentalny element interoperacyjności.

- **ONVIF (Open Network Video Interface Forum):** Globalny standard mający na celu ujednolicenie komunikacji między urządzeniami do nadzoru wideo różnych producentów. Kamera zgodna z ONVIF może być łatwo zintegrowana z systemem VMS lub NVR innej firmy, co daje użytkownikowi swobodę wyboru komponentów systemu.
- **API (Application Programming Interface):** Wielu producentów udostępnia własne API, które pozwala na programistyczną kontrolę zaawansowanych funkcji kamery, niedostępnych w standardzie ONVIF. To właśnie takie API jest wykorzystywane w niniejszej pracy do sterowania kamerą Tapo.

Dzięki tym mechanizmom, kamera IP może stać się inteligentnym czujnikiem w systemie automatyki domowej. Przykładowo, wykrycie ruchu przez kamerę w ogrodzie po zmroku może automatycznie włączyć oświetlenie zewnętrzne, a obraz z kamery przy drzwiach może być wyświetlany na inteligentnym ekranie po naciśnięciu dzwonka.

#### 1.4.8 Powiadomienia push

Powiadomienia push to mechanizm natychmiastowego informowania użytkownika o zdarzeniach wykrytych przez kamerę, bez konieczności ciągłego obserwowania obrazu na żywo. Architektura tego systemu opiera się na współpracy kilku elementów:

1. **Wykrycie zdarzenia:** Kamera wykrywa zdarzenie, takie jak ruch, dźwięk lub detekcja obiektu przez AI.
2. **Komunikacja z serwerem:** Kamera (lub jej oprogramowanie) wysyła informację o zdarzeniu do serwera producenta w chmurze.
3. **Wysyłanie do bramki push:** Serwer producenta kontaktuje się z dedykowaną bramką powiadomień dla danego systemu operacyjnego – APNs (Apple Push Notification service) dla urządzeń z systemem iOS lub FCM (Firebase Cloud Messaging) dla urządzeń z systemem Android.
4. **Dostarczenie do urządzenia:** Bramka APNs/FCM dostarcza powiadomienie na odpowiednie urządzenie mobilne użytkownika.
5. **Wyświetlenie alertu:** System operacyjny telefonu wyświetla powiadomienie na ekranie, często wraz z krótkim opisem i zrzutem ekranu ze zdarzenia, co pozwala użytkownikowi na natychmiastową reakcję.

Ten mechanizm, oparty na modelu publikacji i subskrypcji, jest niezwykle wydajny i oszczędny dla baterii urządzenia mobilnego, ponieważ nie wymaga stałego połączenia aplikacji z serwerem.

## 1.5 Ograniczenia

Pomimo dynamicznego rozwoju i szerokiego spektrum zastosowań, technologia kamer IP obciążona jest szeregiem fundamentalnych ograniczeń. Wynikają one zarówno z natury samej technologii, jak i z modeli biznesowych przyjętych przez producentów sprzętu. Pełne zrozumienie tych ograniczeń jest kluczowe dla projektowania świadomych inżyniersko, niezależnych i bezpiecznych systemów monitoringu. Niniejszy podrozdział dokonuje systematycznej analizy tych wyzwań, grupując je w trzy wzajemnie powiązane domeny: ograniczenia wynikające z infrastruktury sieciowej, luki w zabezpieczeniach i ryzyka dla prywatności oraz ograniczenia narzucane przez ekosystem producenta.

### 1.5.1 Ograniczenia Wynikające z Infrastruktury Sieciowej

Podstawową cechą definiującą kamerę IP jest jej funkcjonowanie jako węzła w sieci komputerowej. Ta fundamentalna zależność sprawia, że wydajność i niezawodność kamery są nierozdzielnie związane z jakością i przepustowością infrastruktury sieciowej, w której operuje. Ograniczenia te są szczególnie dotkliwe w typowych wdrożeniach konsumenckich, gdzie sieć domowa rzadko jest optymalizowana pod kątem ciągłej transmisji wideo w czasie rzeczywistym.

**1.5.1.1 Wymagania Dotyczące Przepustowości i Zużycie Danych** Transmisja strumienia wideo, zwłaszcza w wysokiej rozdzielczości, jest procesem wysoce zasobochłonnym, który generuje stałe i znaczące obciążenie dla sieci. Wielkość tego obciążenia nie jest stałą wartością, lecz dynamiczną funkcją czterech kluczowych zmiennych: rozdzielczości, liczby klatek na sekundę (FPS), zastosowanego kodeka kompresji oraz złożoności obserwowanej sceny.

- **Rozdzielczość (Resolution):** Wyższa rozdzielczość oznacza większą liczbę pikseli w każdej klatce, co przekłada się na bardziej szczegółowy obraz, ale jednocześnie wykładniczo zwiększa ilość danych do przesłania. Strumień wideo w rozdzielczości 1080p (Full HD) wymaga zazwyczaj przepustowości na poziomie 2-4 Mbps, podczas gdy strumień 4K (Ultra HD) może z łatwością konsumować od 8 do 15 Mbps, a nawet więcej.
- **Liczba klatek na sekundę (FPS):** Parametr ten definiuje płynność ruchu w nagraniu. Zwiększenie liczby klatek z 15 do 30 FPS podwaja ilość przesyłanych danych, co bezpośrednio przekłada się na proporcjonalny wzrost wymaganego pasma. Redukcja FPS jest skuteczną metodą ograniczenia zużycia pasma, jednak odbywa się kosztem utraty płynności, co może być krytyczne przy analizie szybkich zdarzeń.

- **Kompresja (Compression):** Wybór kodeka ma fundamentalne znaczenie dla efektywności transmisji. Nowocześniejszy standard H.265 (HEVC) jest w stanie zredukować wymagania dotyczące przepustowości nawet o 50% w porównaniu do powszechnie stosowanego H.264, przy zachowaniu porównywalnej jakości wizualnej.
- **Złożoność sceny (Scene Complexity):** Nowoczesne kodeki wideo optymalizują transmisję, kodując głównie zmiany pomiędzy kolejnymi klatkami. W rezultacie, statyczna scena, taka jak pusty korytarz, będzie generować znacznie mniejszy strumień danych niż dynamiczna scena z dużą ilością ruchu, np. wejście do sklepu w godzinach szczytu. Wysoka aktywność w kadrze może nawet podwoić chwilowe zapotrzebowanie na pasmo.

Te wymagania stają się szczególnie problematyczne w kontekście zapisu w chmurze. Większość konsumenckich planów internetowych ma charakter asymetryczny, oferując wysoką prędkość pobierania (download), ale znacznie niższą prędkość wysyłania (upload). Ponieważ kamera wysyła strumień wideo do chmury, kluczowa jest właśnie przepustowość wysyłania. Pojedyncza kamera 4K może z łatwością wysycić całe dostępne pasmo wysyłania typowego łącza domowego, uniemożliwiając lub znacznie spowalniając działanie innych usług internetowych, takich jak wideokonferencje czy wysyłanie dużych plików.

**1.5.1.2 Zależność od Stabilności i Jakości Połączenia Sieciowego** Protokół transmisji w czasie rzeczywistym (RTP), stanowiący podstawę strumieniowania wideo z kamer IP, jest zoptymalizowany pod kątem minimalizacji opóźnień, a nie gwarancji dostarczenia danych. W praktyce oznacza to, że w przypadku utraty pakietu danych w sieci, nie jest on retransmitowany, aby nie powodować zatrzymania ("zacięcia") obrazu. Ta cecha architektoniczna sprawia, że jakość strumienia jest niezwykle wrażliwa na wszelkie niedoskonałości sieci, które w środowiskach bezprzewodowych (Wi-Fi) są nie wyjątkiem, a regułą.

- **Utrata pakietów (Packet Loss):** Każdy utracony pakiet to bezpowrotnie utracony fragment informacji o obrazie lub dźwięku. Skutkuje to bezpośrednio widocznymi i słyszalnymi artefaktami: pikselozą (obraz staje się "kwadratowy"), zamrożeniem klatek (stuttering), zniekształconym lub przerywanym dźwiękiem, a także desynchronizacją obrazu i dźwięku. Badania wskazują, że poziom utraty pakietów na poziomie zaledwie 2% może już poważnie zdegradować jakość rozmowy wideo lub transmisji na żywo.
- **Niestabilność sieci Wi-Fi:** Zdecydowana większość kamer konsumenckich jest instalowana w sieciach Wi-Fi, które z natury są medium współdzielonym i podatnym na zakłócenia. Na jakość połączenia negatywnie wpływają:

- **Zakłócenia (Interference):** Sygnały z sąsiednich sieci Wi-Fi, urządzeń Bluetooth, kuchenek mikrofalowych i innych urządzeń działających w zatłoczonym paśmie 2.4 GHz mogą powodować kolizje i utratę pakietów.
- **Tłumienie sygnału:** Fizyczne przeszkody, takie jak ściany, stropy i meble, osłabiają sygnał Wi-Fi. Im dalej kamera znajduje się od routera, tym słabsze połączenie, niższa przepustowość i większe prawdopodobieństwo utraty pakietów.
- **Kongestia sieciowa (Network Congestion):** Kamera musi konkurować o dostęp do pasma z każdym innym urządzeniem w sieci domowej (komputerami, smartfonami, telewizorami). W godzinach szczytowego obciążenia, gdy wiele urządzeń aktywnie korzysta z internetu, sieć staje się przeciążona, co prowadzi do opóźnień i odrzucania pakietów.
- **Opóźnienia (Latency) i Zmienność Opóźnień (Jitter):** Opóźnienie to czas potrzebny na dotarcie pakietu od kamery do odbiorcy, a jitter to miara nieregularności tych opóźnień. Nawet jeśli pakiety nie są gubione, ale docierają w nierównych odstępach czasu, może to zakłócić płynność odtwarzania. Odbiorca (np. aplikacja w telefonie) posiada bufor kompensujący niewielki jitter, ale jego przepełnienie w wyniku dużych wahań opóźnień skutkuje zacinaniem się obrazu, podczas gdy odtwarzacz czeka na spóźnione pakiety.

W praktyce, te czynniki degradujące jakość nie działają w sposób addytywny, lecz multiplikatywny. Kamera o wysokiej rozdzielczości (generująca duży strumień danych), umieszczona w dużej odległości od routera (słaby sygnał) w zatłoczonej sieci Wi-Fi (wysokie zakłócenia i utrata pakietów), doświadczy katastrofalnego spadku jakości transmisji. To właśnie ten efekt wzmacniający wyjaśnia, dlaczego doświadczenia użytkowników z kamerami IP bywają tak niespójne i trudne do zdiagnozowania – postrzegany problem często jest wynikiem nałożenia się kilku pozornie niewielkich niedoskonałości sieciowych. Paradoksalnie, główna zaleta marketingowa kamer konsumenckich – łatwość instalacji dzięki łączności bezprzewodowej – stoi w bezpośredniej sprzeczności z ich technicznym wymaganiem posiadania stabilnej, wysokoprzepustowej i niskoprzetłoczonej sieci. Użytkownikowi sprzedawany jest produkt o wysokiej rozdzielczości, który w docelowym, typowym środowisku domowej sieci Wi-Fi, rzadko ma szansę osiągnąć swoją nominalną jakość działania.

### 1.5.2 Luki w Zabezpieczeniach i Ryzyka dla Prywatności

Jako permanentnie podłączone do sieci, często instalowane i zapominane urządzenia peryferyjne, kamery IP stanowią istotny i unikalny wektor zagrożeń cybernetycznych. Ich umiejscowienie w wrażliwych, prywatnych przestrzeniach sprawia, że konsekwen-

cje udanego ataku wykraczają daleko poza typowe incydenty bezpieczeństwa IT, bezpośrednio naruszając prywatność i fizyczne bezpieczeństwo użytkowników.

**1.5.2.1 Wektory Ataków i Powszechne Podatności** Połączenie niezabezpieczonych konfiguracji domyślnych, zaniedbań ze strony użytkowników oraz dużej powierzchni ataku czyni kamery IP głównym celem masowych, zautomatyzowanych ataków.

- **Słabe lub domyślne poświadczenia:** Głównym i najprostszym wektorem ataku jest niezmiennienie przez użytkownika fabrycznych, domyślnych danych logowania (nazwy użytkownika i hasła). Atakujący wykorzystują zautomatyzowane skanery, które przeszukują internet w poszukiwaniu urządzeń odpowiadających na standardowych portach, a następnie próbują uzyskać do nich dostęp, używając publicznie znanych, domyślnych poświadczeń dla danego modelu kamery.
- **Ekspozycja w sieci publicznej:** Błędna konfiguracja routera, w szczególności niepotrzebne przekierowanie portów (port forwarding), może wystawić interfejs administracyjny kamery bezpośrednio na publiczny internet. Takie urządzenia stają się łatwo wykrywalne za pomocą wyspecjalizowanych wyszukiwarek, takich jak Shodan, które indeksują podłączone do internetu urządzenia.
- **Wykorzystanie w botnetach:** Przejęte kamery, ze względu na ich liczbę i stałe podłączenie do sieci, są cennym zasobem do tworzenia botnetów. Złowrogi przykład botnetu Mirai pokazał, jak setki tysięcy skompromitowanych urządzeń IoT, w dużej mierze kamer IP, zostały wykorzystane do przeprowadzenia zmasowanych ataków typu DDoS (Distributed Denial of Service), które zakłóciły działanie największych serwisów internetowych. Incydent ten unaoczniał, jak indywidualne zaniedbanie bezpieczeństwa może przyczynić się do globalnej destabilizacji internetu.
- **Wykorzystanie jako proxy do działalności przestępczej:** Nowszym i bardziej podstępny zagrożeniem jest wykorzystywanie przejętych kamer jako serwerów proxy do anonimizacji działalności przestępczej. Badania naukowe dowodzą, że skompromitowane urządzenia IoT, w tym w dużej mierze kamery, są masowo wykorzystywane w infrastrukturze przestępczej do przeprowadzania ataków na instytucje finansowe, takich jak credential stuffing (automatyczne testowanie skradzionych loginów i haseł), kradzież kryptowalut czy oszustwa z użyciem kart kredytowych. Właściciel kamery jest najczęściej nieświadomy, że jego domowe urządzenie zabezpieczające stało się węzłem w globalnej sieci przestępczej.

**1.5.2.2 Ryzyka Związane z Oprogramowaniem Firmware** Firmware, czyli oprogramowanie układowe, pełni rolę systemu operacyjnego kamery i stanowi krytyczną,



choć często niewidoczną dla użytkownika, granicę bezpieczeństwa. Połączenie zamkniętego, nieaudytowanego kodu z nieregularnym wsparciem ze strony producenta tworzy trwałą i niebezpieczną powierzchnię ataku.

- **Zamknięty i nieprzejrzyisty kod:** W przeciwieństwie do oprogramowania open-source, firmware większości kamer konsumenckich to "czarna skrzynka". Użytkownicy i niezależni badacze bezpieczeństwa nie mają możliwości łatwego audytu kodu w poszukiwaniu luk, tylnych furtek (backdoorów) czy niebezpiecznych praktyk, takich jak zaszyte na stałe w kodzie hasła (hardcoded credentials).
- **Brak terminowych aktualizacji:** Producenci często z opóźnieniem publikują łatki bezpieczeństwa dla nowo odkrytych podatności (oznaczonych numerami CVE), a wielu użytkowników nie instaluje dostępnych aktualizacji. Stwarza to szerokie "okno możliwości" dla atakujących, którzy mogą wykorzystywać dobrze znane i opisane luki w zabezpieczeniach.
- **Polityka End-of-Life (EOL):** Jest to krytyczne, niemożliwe do obejścia ograniczenie. W momencie, gdy producent ogłasza, że dany model produktu osiągnął status EOL (koniec życia), zaprzestaje wszelkiego wsparcia, w tym wydawania jakichkolwiek aktualizacji bezpieczeństwa. Każda podatność odkryta po tej dacie staje się permanentnym zagrożeniem typu "zero-day", na które nigdy nie powstanie oficjalna łątka. Biorąc pod uwagę długi cykl życia fizycznego kamer, prowadzi to do powstawania w sieci rosnącej populacji przestarzałych urządzeń, które są tykającymi bombami zegarowymi z punktu widzenia bezpieczeństwa.

**1.5.2.3 Implikacje dla Prywatności Użytkownika** Umieszczenie kamer IP w najbardziej prywatnych przestrzeniach – domach, sypialniach, biurach – sprawia, że naruszenie bezpieczeństwa jest jednocześnie głębokim naruszeniem prywatności. Co więcej, model operacyjny oparty na usługach chmurowych wprowadza dodatkowe ryzyka związane z zarządzaniem i ochroną danych.

- **Nieautoryzowana inwigilacja:** Najbardziej bezpośrednim i dotkliwym ryzykiem jest uzyskanie przez atakującego dostępu do transmisji wideo i audio na żywo. Umożliwia to podglądanie i podsłuchiwanie domowników, co prowadziło do udokumentowanych przypadków nękania, szantażu, a nawet szpiegostwa.
- **Bezpieczeństwo danych w chmurze:** W modelu, w którym nagrania wideo są przechowywane na serwerach producenta, użytkownik traci bezpośrednią kontrolę nad swoimi danymi. Musi on w pełni zaufać praktykom bezpieczeństwa stosowanym przez dostawcę usługi w celu ochrony przed włamaniem do infrastruktury chmurowej. Kwestie takie jak polityka prywatności, jurysdykcja przechowywania danych oraz prawa dostępu do nich stają się kluczowe. Udany atak na ser-

wery producenta może skutkować jednoczesnym wyciekiem prywatnych nagrań tysięcy, a nawet milionów użytkowników.

Krajobraz zagrożeń IoT charakteryzuje się głęboką asymetrią ryzyka. Wysiłek wymagany od atakującego do masowego skompromitowania kamer jest niezwykle niski (np. zautomatyzowane skanowanie w poszukiwaniu domyślnych haseł), podczas gdy potencjalne konsekwencje dla ofiary są niezwykle wysokie (naruszenie prywatności, straty finansowe, nieświadomy udział w botnecie). Ten wysoce korzystny dla atakujących stosunek ryzyka do zysku gwarantuje, że tego typu ataki będą kontynuowane i będą rosły w skali. Co więcej, problem EOL nie jest jedynie kwestią techniczną, ale bezpośrednią konsekwencją modelu biznesowego, który priorytetyzuje sprzedaż nowego sprzętu nad wspieraniem istniejących produktów. Tworzy to zjawisko "planowanego starzenia się bezpieczeństwa", w którym fizyczna funkcjonalność urządzenia znacznie przeżywa jego cyfrowe bezpieczeństwo. Decyzja biznesowa o zakończeniu wsparcia dla danego modelu przekłada się bezpośrednio na permanentną, niemożliwą do załatwienia lukę w zabezpieczeniach dla każdego użytkownika, który nie zdecyduje się na wymianę sprzętu.

### 1.5.3 Ograniczenia Modelu Biznesowego i Uzależnienie od Producenta

Ostatnia kategoria ograniczeń nie wynika z samej technologii, lecz ze strategicznych decyzji producentów, mających na celu stworzenie zamkniętych, własnościowych ekosystemów. Działania te, motywowane biznesowo, w sposób fundamentalny ograniczają prawa użytkownika, interoperacyjność i długoterminowe bezpieczeństwo, co stanowi główną motywację dla projektu badawczego opisanego w niniejszej pracy.

**1.5.3.1 Zjawisko "Vendor Lock-in" w Ekosystemach IoT** Producenci celowo projektują swoje produkty w taki sposób, aby stworzyć wysokie koszty zmiany dostawcy, uzależniając klienta od swojego ekosystemu na cały cykl życia produktu.

- **Definicja i mechanizm:** Zjawisko "vendor lock-in" (uzależnienie od dostawcy) ma miejsce, gdy koszt i wysiłek związany ze zmianą produktu na konkurencyjny są tak znaczące, że klient jest w praktyce "uwięziony" u pierwotnego dostawcy. W przypadku kamer IP jest to realizowane poprzez ścisłe powiązanie sprzętu (kamery) z dedykowanym oprogramowaniem (aplikacją mobilną) i usługami backendowymi (platformą chmurową) producenta. Funkcje kluczowe, takie jak pierwsza konfiguracja, zdalny podgląd, powiadomienia o ruchu czy zapis w chmurze, są dostępne wyłącznie za pośrednictwem tego zamkniętego ekosystemu.
- **Konsekwencje dla użytkownika:** Użytkownik nie ma możliwości integracji i zarządzania kamerami różnych marek w jednej, wspólnej aplikacji. Jeśli zdecyduje

się na zmianę platformy (np. z powodu niezadowolenia z usług lub polityki cenowej), często jest zmuszony do wymiany całego posiadanego sprzętu, nawet jeśli jest on w pełni sprawny technicznie.

**1.5.3.2 Konsekwencje Zamkniętych Protokółów i API** Głównym narzędziem technicznym służącym do egzekwowania strategii "vendor lock-in" jest stosowanie zamkniętych, nieudokumentowanych interfejsów programistycznych (API) zamiast otwartych, ustandaryzowanych protokołów.

- **Blokowanie interoperacyjności:** Chociaż istnieją globalne standardy, takie jak ONVIF, stworzone w celu zapewnienia współpracy urządzeń różnych producentów, wielu dostawców sprzętu konsumenckiego celowo ich nie implementuje lub oferuje jedynie częściowe, zawodne wsparcie.
- **Własnościowa kontrola:** Zamiast tego, do sterowania zaawansowanymi funkcjami, takimi jak ruch PTZ, zmiana ustawień czy dostęp do funkcji opartych na AI, wykorzystywane są prywatne, nieudokumentowane API.
- **Konieczność inżynierii wstecznej:** Aby zintegrować taką kamerę z systemem open-source (np. Home Assistant) lub z autorskim rozwiązaniem, takim jak opracowane w ramach niniejszej pracy, programiści są zmuszeni do prowadzenia złożonego i czasochłonnego procesu inżynierii wstecznej w celu rozszyfrowania działania zamkniętych protokołów. Takie rozwiązanie jest z natury niestabilne, gdyż każda aktualizacja oprogramowania firmware przez producenta może zmienić API i zniszczyć działającą integrację.

**1.5.3.3 Ryzyka Związane z Cyklem Życia Usługi** Powiązanie funkcjonalności sprzętu z usługą chmurową przenosi na konsumenta znaczące ryzyko długoterminowe. Dalsze działanie zakupionego urządzenia staje się zależne od ciągłości biznesowej i strategicznych decyzji producenta.

- **"Bricking" przez subskrypcję:** Niektóre modele biznesowe, określane jako "Hostage-as-a-Service", wymagają aktywnej subskrypcji do pełnego funkcjonowania kamery. Jeśli użytkownik przestanie płacić lub producent zmieni warunki, urządzenie może utracić kluczowe funkcje lub stać się całkowicie bezużyteczne – zamienić się w "cegłę" (ang. brick).
- **Zakończenie świadczenia usługi:** Producent może zbankrutować lub podjąć decyzję o wycofaniu danej linii produktów lub zamknięciu powiązanej z nią usługi chmurowej. Jeśli kamera do swojego działania wymaga uwierzytelnienia w tej usłudze, może z dnia na dzień przestać działać, bez możliwości odwołania dla konsumenta, który zakupił sprzęt.

- **Nieprzewidziane koszty i zmiany polityki:** Producent, świadomy wysokich kosztów zmiany platformy przez klienta, może jednostronnie podnosić ceny subskrypcji lub zmieniać zakres oferowanych funkcji, stawiając użytkownika w sytuacji bez wyjścia.

Strategia "vendor lock-in" w sposób bezpośredni potęguje ryzyka związane z bezpieczeństwem. Zamykając użytkownika w swoim ekosystemie, producent zmusza go do polegania na jednym, centralnym punkcie w kwestii aktualizacji bezpieczeństwa. Dostawca, który opieszale publikuje łatki lub stosuje agresywną politykę EOL, naraża całą swoją "uwięzioną" bazę użytkowników. Brak możliwości zmiany oprogramowania na bezpieczniejszą alternatywę open-source lub przejścia na inną platformę zarządzania eliminuje kluczową strategię mitygacji ryzyka, dostępną w bardziej otwartych ekosystemach. W ten sposób uzależnienie od dostawcy odbiera użytkownikowi sprawczość w zarządzaniu własnym bezpieczeństwem, czyniąc go całkowicie zależnym od kompetencji i interesów biznesowych jednej firmy. Co więcej, istnieje fundamentalny konflikt pomiędzy trendem dodawania do kamer "inteligentnych" funkcji opartych na chmurze a dążeniem użytkownika do suwerenności danych i interoperacyjności. Zaawansowane funkcje, takie jak detekcja osób czy rozpoznawanie określonych dźwięków, są często realizowane jako analityka po stronie serwerów producenta i dostępne wyłącznie przez jego własnościową aplikację. Aby z nich korzystać, użytkownik musi zgodzić się na warunki ekosystemu i politykę danych dostawcy. To sprawia, że rezygnacja z platformy producenta na rzecz lokalnego, otwartego rozwiązania wiąże się z utratą tych reklamowanych, "inteligentnych" możliwości. W ten sposób, te same cechy, które czynią produkt atrakcyjnym, stają się jednocześnie łańcuchami, które przywiązują użytkownika do dostawcy. Jest to kluczowe uzasadnienie dla projektu realizowanego w niniejszej pracy, który ma na celu odtworzenie podobnych funkcjonalności w otwartym, lokalnym i niezależnym od producenta środowisku.

## 1.6 Wnioski - Analiza

Łączenie Teorii z Zastosowaniem: Kontekstowe Wnioski z Fundamentalnej Analizy Technologii Kamery IP w Projekcie Integracji Open-Source Wprowadzenie: Dychotomia Nowoczesnego IoT - Zaawansowana Funkcjonalność kontra Zamknięte Ekosystemy Centralnym argumentem niniejszej analizy jest teza, że zaawansowanie technologiczne współczesnych, konsumenckich urządzeń Internetu Rzeczy (IoT) jest paradoksalnie podważane przez modele biznesowe, które wprowadzają je na rynek. Praca inżynierska Marcina Mazura, zatytułowana "Wykorzystanie oprogramowania Open-Source do współpracy z kamerami TP-Link TAPO", wykorzystuje kamerę TP-Link Tapo C200 jako doskonały przykład tego konfliktu. Z jednej strony, urządzenie to oferuje imponujący zestaw funkcji, w tym obraz w rozdzielczości 1080p, zdalne sterowanie obrotem

i pochyleniem (PTZ), tryb nocny, dwukierunkowe audio oraz wbudowane mechanizmy detekcji oparte na sztucznej inteligencji, takie jak wykrywanie osób czy płaczu dziecka. Z drugiej strony, pełna funkcjonalność tych zaawansowanych możliwości jest nierozdzielnie związana z infrastrukturą chmurową producenta i jego dedykowaną aplikacją mobilną, tworząc zamknięty ekosystem, tzw. "ogród otoczony murem" (walled garden). Problem ten nie jest unikalny dla TP-Link; stanowi on powszechne zjawisko w krajo-  
brazie IoT, które hamuje innowacyjność, ogranicza prawa użytkowników i prowadzi do powstawania cyfrowych nierówności.

Analiza ta ujawnia paradoksalną zależność: te same cechy, które definiują urządzenie jako "inteligentne" – takie jak analiza obrazu w chmurze, zdalny dostęp czy powiadomienia push – stają się jednocześnie technicznymi mechanizmami egzekwowania uzależnienia od dostawcy (vendor lock-in). "Inteligentne" funkcje są dostarczane jako usługa, a nie jako cecha produktu, co fundamentalnie zmienia charakter własności urządzenia zakupionego przez użytkownika. Prosta kamera dostarcza jedynie strumień wideo; "inteligentna" kamera oferuje usługi, takie jak inteligentne alerty i zapis w chmurze. Usługi te wymagają infrastruktury backendowej (serwerów) do przetwarzania danych i zarządzania komunikacją, którą w pełni kontroluje producent. Dostęp do tej infrastruktury jest możliwy wyłącznie za pośrednictwem jego autorskiej, zamkniętej aplikacji. W konsekwencji, im więcej "inteligentnych" funkcji użytkownik pragnie wykorzystać, tym głębiej zostaje osadzony w zamkniętym ekosystemie producenta. Reklamowana propozycja wartości staje się łańcuchem, który wiąże użytkownika, co stanowi bezpośrednią motywację dla celu pracy inżynierskiej: oddzielenia sprzętowych możliwości urządzenia od ekosystemu usługowego producenta.

Dekonstrukcja Architektury Kamery IP jako Wstęp do Interwencji Szczegółowej  
Zrozumienie wewnętrznej architektury kamery IP nie jest jedynie ćwiczeniem akademickim; jest to warunek konieczny do zidentyfikowania konkretnych, technicznych punktów kontroli, które producenci wykorzystują do narzucania swoich ekosystemów. Rozdział pierwszy pracy dyplomowej dokonuje dekonstrukcji kamery na jej kluczowe komponenty sprzętowe i programowe, co pozwala na precyzyjne zlokalizowanie tych punktów.

Warstwa Sprzętowa: Fizyczny Fundament  
Fundamentem działania kamery jest jej warstwa sprzętowa, zdominowana przez wysoce zintegrowane komponenty:

System-on-a-Chip (SoC): Praca słusznie identyfikuje SoC jako centralną jednostkę przetwarzającą. W przypadku kamery Tapo C200 jest to najprawdopodobniej procesor z serii Ingenic T31, który integruje w sobie procesor MIPS, dedykowany procesor sygnału obrazu (ISP) oraz sprzętowy koder wideo (H.264/H.265). Taka integracja funkcji na jednym układzie scalonym jest dla producenta efektywna kosztowo, ale jednocześnie centralizuje całą kontrolę nad urządzeniem.

Potok Przetwarzania Obrazu (ISP Pipeline): Praca szczegółowo opisuje proces prze-

tworzenia sygnału "od fotonu do pakietu": od matrycy CMOS, przez filtr Bayera, demosaikowanie, przetwarzanie w ISP, aż po kompresję do formatu H.264. Ten potok jest w całości kontrolowany przez oprogramowanie układowe SoC, które determinuje jakość i format strumienia wideo, zanim ten zostanie w ogóle udostępniony w sieci.

Interfejs Sieciowy: Kamera wykorzystuje układ Wi-Fi, taki jak Realtek RTL8188FTV, do zarządzania łącznością sieciową. Układ ten obsługuje niższe warstwy modelu TCP/IP, jednak protokoły warstwy aplikacji są zarządzane przez główny SoC.

Warstwa Programowa: Płaszczyzna Kontroli Nad warstwą sprzętową operuje oprogramowanie, które stanowi faktyczną płaszczyznę kontroli:

Firmware: Praca definiuje firmware jako "pomost" między sprzętem a funkcjonalnością dostępną dla użytkownika. To zamknięte i nieaudytowane oprogramowanie jest ostatecznym punktem kontrolnym, zarządzającym wszystkim, od inicjalizacji sprzętu po implementację usług sieciowych.

Stos Protokółów Sieciowych: Rozdział pierwszy szczegółowo analizuje model TCP/IP w kontekście kamery. Kluczowe protokoły umożliwiające otwartą integrację to:

RTSP (Real-Time Streaming Protocol): Jest to standardowy protokół, który pozwala na dostęp do surowego strumienia wideo w formacie H.264. Kamera Tapo C200 obsługuje ten protokół na porcie 554, udostępniając strumienie o różnej jakości pod określonymi adresami URL. Jest to fundamentalna "szczelina w murze", którą praca inżynierska wykorzystuje. ONVIF (Open Network Video Interface Forum): Tapo C200 deklaruje wsparcie dla standardu ONVIF. Jak zostanie wykazane dalej, wsparcie to jest jednak celowo ograniczone.

Nowoczesna architektura urządzeń IoT oparta na SoC, promując efektywność i niski koszt, nieuchronnie prowadzi do centralizacji kontroli w ramach oprogramowania układowego producenta. Ten wybór architektoniczny jest technicznym warunkiem wstępnym, który umożliwia realizację biznesowego modelu vendor lock-in. Gdyby nie ta nieprzejrzysta, scentralizowana warstwa kontrolna, otwarte protokoły i oprogramowanie modyfikowalne przez użytkownika byłyby domyślnym standardem, a nie wyjątkiem. Producent, kontrolując monolityczny obraz firmware, może swobodnie decydować, które protokoły zaimplementować w pełni (np. standardowy RTSP), a które zachować jako własnościowe (np. sterowanie PTZ). Może również decydować, jakie dane są wysyłane do jego serwerów chmurowych i ma możliwość zdalnego wyłączenia urządzenia poprzez przyszłą aktualizację. Istnieje zatem bezpośredni związek przyczynowy: architektura sprzętowa (scentralizowany SoC) umożliwia architekturę oprogramowania (monolityczny, zamknięty firmware), co z kolei umożliwia model biznesowy (vendor lock-in). Problem inżynierski polega więc nie tylko na napisaniu kodu, ale na świadomym obejściu celowo restrykcyjnej architektury.

Powiązania Technologii, Modeli Biznesowych i Bezpieczeństwa Synteza ustaleń z podrozdziału 1.5 ("Ograniczenia") z techniczną dekonstrukcją kamery dowodzi, że ogra-

niczenia kamer IP nie są przypadkowymi produktami ubocznymi, lecz bezpośrednimi konsekwencjami strategicznych decyzji biznesowych. Decyzje te priorytetyzują kontrolę nad ekosystemem kosztem wolności użytkownika, interoperacyjności i długoterminowego bezpieczeństwa.

Vendor Lock-in jako Celowa Strategia Praca definiuje vendor lock-in jako uzależnienie od autorskiego oprogramowania i infrastruktury chmurowej producenta, co ogranicza możliwości integracji. Jest to realizowane za pomocą mechanizmów technicznych, takich jak własnościowe interfejsy, protokoły i modele danych. Ekosystem Tapo jest tego doskonałym przykładem, wykorzystując niepubliczne API do funkcji sterujących. W rezultacie użytkownicy zostają "uwięzieni", stając w obliczu wysokich kosztów migracji i braku elastyczności. W skrajnych przypadkach, gdy usługa zostanie wycofana (jak w przypadku Google Cloud IoT Core), zakupiony sprzęt może stać się bezużyteczny.

Strategiczne Ograniczanie Otwartych Standardów (ONVIF) Chociaż kamera Tapo C200 obsługuje standard ONVIF, jest to jedynie Profil S. Profil ten obejmuje podstawowe funkcje strumieniowania wideo i audio, ale nie gwarantuje obsługi zaawansowanych funkcji, takich jak dwukierunkowe audio, a wsparcie dla PTZ w urządzeniach konsumenckich jest często zawodne. Fora pomocy technicznej TP-Link zawierają skargi użytkowników oraz oficjalne oświadczenia potwierdzające, że sterowanie PTZ nie jest dostępne przez ONVIF i wymaga użycia aplikacji Tapo. To dowodzi celowego działania: zaimplementować standard w stopniu minimalnym, aby móc deklarować zgodność w celach marketingowych, jednocześnie wstrzymując kluczowe funkcje, aby utrzymać uzależnienie od autorskiej aplikacji.

Bezpieczeństwo i Prywatność jako Koszty Zewnętrzne Model zamkniętego ekosystemu wprowadza poważne zagrożenia bezpieczeństwa. Nieaudytowany firmware może zawierać luki lub tylne furtki, a chmura producenta staje się pojedynczym punktem awarii i atrakcyjnym celem dla atakujących. Model biznesowy tworzy zjawisko "planowanego starzenia się bezpieczeństwa". Gdy urządzenie osiąga status End-of-Life (EOL), producent zaprzestaje wydawania aktualizacji bezpieczeństwa, pozostawiając je na stałe podatne na ataki, mimo że fizycznie jest wciąż sprawne. Prywatność danych jest również zagrożona, ponieważ wrażliwe dane audio i wideo z wnętrza domu są przesyłane i przechowywane na serwerach firm trzecich, gdzie mogą być wykorzystywane do profilowania lub zostać ujawnione w wyniku naruszenia bezpieczeństwa.

Model zamkniętego ekosystemu odwraca tradycyjny paradygmat bezpieczeństwa. Zamiast traktować sieć lokalną jako strefę zaufaną, a publiczny internet jako wroga, architektura producenta traktuje własną sieć lokalną użytkownika jako niezaufaną. Jedyna zaufana ścieżka prowadzi od urządzenia bezpośrednio do chmury producenta, wymuszając przepływ całej kontroli i danych przez monitorowany, monetyzowalny i podatny na ataki kanał zewnętrzny. Domyślny tryb pracy kamery Tapo wymaga stałego połączenia z chmurą TP-Link w celu konfiguracji, otrzymywania alertów i zdalnego do-

stępu. Oznacza to, że "domem" urządzenia jest chmura, a nie sieć LAN użytkownika, która staje się jedynie siecią tranzytową. Zmusza to użytkownika do domyślnego wystawiania swoich prywatnych danych na publiczny internet. Projekt inżynierski, dążący do stworzenia w pełni lokalnego rozwiązania, jest zatem bezpośrednią próbą przywrócenia prawidłowej postawy bezpieczeństwa, w której sieć lokalna jest podstawową i zaufaną domeną operacyjną.

Proponowane Wnioski dla Rozdziału 1: "Wprowadzenie technologiczne Kamer IP" Poniższy tekst stanowi propozycję sformułowania wniosków z pierwszego rozdziału, które syntetyzują przedstawioną analizę i tworzą logiczny pomost do dalszych części pracy inżynierskiej.

Analiza przeprowadzona w niniejszym rozdziale dokonała dekonstrukcji współczesnej kamery IP, ukazując ją jako złożony system wbudowany, który łączy zaawansowaną optykę (matryca CMOS), wyspecjalizowany sprzęt (SoC, ISP) oraz rozbudowany stos sieciowy (TCP/IP, RTSP, H.264). Ta technologiczna podstawa umożliwia szerokie spektrum zastosowań, od prostego monitoringu wizyjnego po zaawansowaną analitykę danych, co czyni kamery IP kluczowym elementem ekosystemów Internetu Rzeczy.

Jednocześnie, dogłębna analiza wykazała istnienie fundamentalnego konfliktu: otwarty, oparty na standardach potencjał technologii IP jest systematycznie ograniczany przez własnościowe modele biznesowe producentów. Ustalono, że o ile protokoły takie jak RTSP oferują bramę do interoperacyjności poprzez dostęp do surowego strumienia wideo, o tyle kluczowe funkcjonalności, takie jak sterowanie urządzeniem (PTZ) czy inteligentna detekcja zdarzeń, są celowo zamykane w ramach nieudokumentowanych, autorskich interfejsów API.

Konsekwencją tej strategii jest krytyczny problem inżynierski i użytkowy, jakim jest "vendor lock-in" – uzależnienie od dostawcy. Zjawisko to nie tylko ogranicza funkcjonalność i swobodę integracji zakupionego sprzętu, ale również wprowadza istotne ryzyka dla bezpieczeństwa i prywatności, wynikające z polegania na nieaudytowanym oprogramowaniu firmware oraz zewnętrznych usługach chmurowych. Analiza dowodzi, że nie są to problemy marginalne, lecz systemowe ograniczenia, nierozzerwalnie związane z dominującym podejściem rynkowym.

W kontekście całej pracy inżynierskiej, niniejszy rozdział pełni rolę niezbędnego fundamentu teoretycznego i uzasadnienia dla podjętych działań projektowych. Precyzyjnie definiuje on "przestrzeń problemową", przekształcając ogólne zjawisko rynkowe w konkretny zestaw wyzwań inżynierskich. Ugruntowuje potrzebę stworzenia niestandardowego, otwartego oprogramowania i wskazuje na specyficzne bariery techniczne – takie jak obejście własnościowych protokołów sterujących i lokalna reimplementacja "inteligentnych" funkcji – które projektowane rozwiązanie musi pokonać, aby osiągnąć swój główny cel: przywrócenie użytkownikowi suwerenności nad posiadanym urządzeniem i generowanymi przez nie danymi.



Od Problemów Fundamentalnych do Imperatywów Inżynierskich Analiza przeprowadzona w rozdziale pierwszym nie jest jedynie teoretycznym tłem; stanowi ona bezpośredni katalog problemów, na które zaimplementowane w dalszej części pracy rozwiązanie programistyczne jest precyzyjną odpowiedzią. Każdy kluczowy komponent opracowanego systemu jest bezpośrednią i konieczną reakcją na ograniczenie celowo narzucone przez producenta. Poniższa tabela przedstawia tę zależność, mapując zidentyfikowane bariery na konkretne, zastosowane technologie open-source.

Tabela 1: Mapowanie Ograniczeń Własnościowych na Rozwiązania Inżynierskie Open-Source

Funkcjonalność Ograniczenie Własnościowe (zidentyfikowane w Rozdziale 1) Zaimplementowane Rozwiązanie Open-Source (Rozdziały 3 i 4) Źródła Dostęp do Strumienia Wideo Dostępny jest standardowy, lecz jednokierunkowy protokół RTSP, który nie oferuje żadnych możliwości sterowania. Bezpośrednie wykorzystanie strumienia RTSP za pomocą FFmpeg do ekstrakcji danych oraz OpenCV do przetwarzania klatek. Sterowanie Kameralą (PTZ) Funkcje sterujące (obrot, pochylenie) są nieobecne w implementacji ONVIF i zablokowane za zamkniętym, nieudokumentowanym, własnościowym API opartym na HTTP. Integracja biblioteki PyTapo, będącej wynikiem społecznościowej inżynierii wstecznej, w celu wysyłania poleceń sterujących bezpośrednio do lokalnego API kamery. Detekcja Zdarzeń (Ruch) Wbudowane funkcje AI (np. detekcja osób) działają jak "czarna skrzynka", a alerty i dane o zdarzeniach są dostępne wyłącznie przez chmurę i aplikację producenta, co uniemożliwia lokalną integrację. Implementacja własnego, działającego po stronie serwera algorytmu detekcji ruchu z wykorzystaniem techniki różnicowania klatek (frame differencing) w OpenCV, dającego użytkownikowi pełną kontrolę nad czułością i akcjami wyzwalanymi przez zdarzenie. Wdrożenie i Skalowalność Systemu Oficjalne rozwiązanie wymaga ręcznej konfiguracji każdego urządzenia za pomocą aplikacji mobilnej i jest powiązane z jednym kontem użytkownika, co uniemożliwia reprodukowalność i skalowalność. Zamknięcie całego stosu aplikacyjnego (serwer Python, zależności) w kontenerze Docker, co umożliwia powtarzalne, jednopoleceniowe wdrożenie na dowolnym hoście, w tym na urządzeniach o niskiej mocy, takich jak Raspberry Pi. Interfejs Użytkownika w Czasie Rzeczywistym Jedynym oficjalnym interfejsem jest autorska aplikacja mobilna, która polega na serwerach chmurowych, wprowadzając opóźnienia i zależność od połączenia z internetem. Stworzenie klienta webowego obsługiwanego przez backend oparty na Flask, wykorzystujący Flask-SocketIO do przesyłania przetworzonych klatek wideo do przeglądarki w czasie zbliżonym do rzeczywistego, z niskim opóźnieniem. Analiza Końcowa Rygorystyczne, oparte na teorii podejście, zaprezentowane w rozdziale pierwszym, jest kluczowe w dziedzinie inżynierii IoT. Rozwiązywanie złożonych, socjotechnicznych problemów, takich jak vendor lock-in, wymaga czegoś więcej niż tylko umiejętności programistycznych; wymaga głębokiego, fundamentalnego

zrozumienia technologii bazowej, strategii biznesowych, które ją kształtują, oraz wynikających z nich implikacji dla bezpieczeństwa. Rozdział pierwszy z powodzeniem dostarcza tego niezbędnego fundamentu. Dokonuje on skrupulatnego mapowania krajobrazu technologicznego, identyfikuje jego systemowe wady i w ten sposób ramuje późniejszą, praktyczną implementację nie jako zwykły "projekt", lecz jako uzasadnioną i konieczną interwencję inżynierską. Wnioski z rozdziału pierwszego stanowią zatem kamień węgielny, na którym zbudowany jest cały wkład intelektualny i praktyczny niniejszej pracy dyplomowej.

## 2 Analiza Kamery TP-Link TAPO C200

### 2.1 Charakterystyka Ogólna i Pozycja Rynkowa

Kamera TP-Link Tapo C200 jest pozycjonowana na rynku jako flagowy przykład konsumenckiego urządzenia **Internetu Rzeczy (IoT)** w kategorii „**Smart Home**”. Jej podstawowym celem rynkowym jest dostarczenie masowemu odbiorcy niedrogiego, łatwego w obsłudze i bogatego w funkcje systemu monitoringu wewnętrznego, który jest w pełni zarządzany za pomocą aplikacji mobilnej. Strategia TP-Link polega na oferowaniu zaawansowanych możliwości sprzętowych w wysoce konkurencyjnej cenie, co ma na celu szybkie zdobycie udziału w rynku i wprowadzenie użytkowników do zamkniętego ekosystemu usług firmy.

Kluczowe funkcje reklamowane w oficjalnej specyfikacji technicznej stanowią fundament jej propozycji wartości:

- **Wysoka jakość obrazu:** Kamera oferuje natywną rozdzielczość 1080p Full HD ( $1920 \times 1080$  pikseli) przy płynnej prędkości 30 klatek na sekundę. Stanowi to standard rynkowy dla nowoczesnych systemów monitoringu, pozwalający na wyraźną identyfikację szczegółów.
- **Mechanizm Pan/Tilt (PTZ):** Urządzenie jest wyposażone w zmotoryzowaną głowicę, umożliwiającą zdalny obrót w poziomie (Pan) w zakresie  $360^\circ$  oraz pochylenie w pionie (Tilt). Ta funkcja eliminuje martwe strefy i pozwala na monitorowanie całego pomieszczenia za pomocą jednego urządzenia.
- **Tryb nocny (Noktowizja):** Zintegrowane diody LED podczerwieni (IR) o długości fali 850 nm zapewniają widoczność w całkowitej ciemności na deklarowany dystans do 40 stóp (około 12 metrów).
- **Dwukierunkowe audio:** Wbudowany mikrofon i głośnik umożliwiają komunikację w czasie rzeczywistym, co przekształca kamerę z pasywnego sensora w interaktywny interkom.
- **Zaawansowana detekcja:** Poza standardową detekcją ruchu, Tapo C200 reklamuje funkcje oparte na sztucznej inteligencji (AI), w tym „**Detekcję Osób**” (**Person Detection**) oraz „**Detekcję Płaczu Dziecka**” (**Baby Crying Detection**).

Należy jednak podkreślić, że zamierzony przez producenta model operacyjny (**Intended Operational Model**) jest fundamentalnie oparty na koncepcji „**zamkniętego ogrodu**” (**walled garden**). Pełna funkcjonalność, począwszy od krytycznego procesu pierwszej konfiguracji (provisioningu), aż po dostęp do zaawansowanych funkcji detekcji i zdalnego podglądu, jest nierozzerwalnie związana z autorską aplikacją mobilną Tapo oraz infrastrukturą chmurową TP-Link. Ten model stanowi centralny problem badawczy niniejszej pracy. Tytułowe „**Wykorzystanie oprogramowania Open-Source do**

**współpracy z kamerami TP-Link TAPO**” jest bezpośrednią odpowiedzią inżynierską na wyzwanie, jakim jest obejście tych sztucznych ograniczeń. Niniejszy rozdział dokonuje systematycznej dekonstrukcji kamery Tapo C200, aby precyzyjnie zidentyfikować, które jej komponenty są otwarte i możliwe do integracji, a które zostały celowo zamknięte przez producenta w ramach strategii „**vendor lock-in**”. Analiza ta stanowi techniczne uzasadnienie dla zaprojektowania i implementacji niestandardowego oprogramowania opisanego w kolejnych rozdziałach pracy.

## 2.2 Architektura Sprzętowa

Analiza architektury sprzętowej jest kluczowa dla zrozumienia zarówno potencjału, jak i ograniczeń kamery. Komponenty fizyczne definiują surowe możliwości urządzenia, które oprogramowanie układowe (**firmware**) następnie eksponuje – lub ukrywa – użytkownikowi.

Sercem każdej kamery IP jest jej przetwornik obrazu. Tapo C200 wykorzystuje sensor 1/2.8" **Progressive Scan CMOS**. Jest to kluczowa informacja, ponieważ rozmiar sensora i typ technologii CMOS determinują bazową jakość obrazu, czułość na światło (kluczową dla noktowizji) oraz zakres dynamiczny. Jest to fundament, na którym opiera się cały potok przetwarzania wideo.

W zakresie systemów peryferyjnych, kamera wyposażona jest w zintegrowany mikrofon i głośnik, co stanowi techniczną podstawę dla funkcji dwukierunkowego audio. Interfejs sieciowy jest ograniczony wyłącznie do komunikacji bezprzewodowej w paśmie 2.4 GHz, obsługując standardy *IEEE802.11b/g/n*. Brak portu Ethernet oraz nieobsługiwanie pasma 5 GHz jednoznacznie pozycjonują C200 jako urządzenie klasy konsumenckiej, gdzie priorytetem jest łatwość instalacji bezprzewodowej, a nie maksymalna stabilność i przepustowość połączenia, jakiej wymagałyby zastosowania profesjonalne.

Centralną jednostką obliczeniową urządzenia jest wysoce zintegrowany układ **System-on-a-Chip (SoC)**. Choć oficjalna specyfikacja nie wymienia konkretnego modelu, analiza typowych architektur dla tego segmentu urządzeń wskazuje na użycie procesora integrującego wiele funkcji w jednym układzie (np. z serii Ingenic T31). Taki SoC łączy w sobie główny procesor (CPU), dedykowany procesor sygnału obrazu (ISP) odpowiedzialny za operacje takie jak demozaikowanie i redukcja szumów, oraz – co najważniejsze – sprzętowy koder wideo H.264.

Wybór takiej architektury SoC jest kluczową decyzją inżynierską i biznesową. Z jednej strony, wysoka integracja drastycznie obniża koszty produkcji (**Bill of Materials - BOM**), co pozwala na oferowanie kamery w atrakcyjnej cenie. Z drugiej strony, taka monolityczna architektura ma głębokie implikacje dla otwartości systemu. Oznacza to, że każda pojedyncza funkcja urządzenia – od ruchu silnikami PTZ, przez odczyt z

sensora CMOS, aż po kompresję H.264 i zarządzanie interfejsem sieciowym – jest kontrolowana przez jeden, monolityczny obraz oprogramowania układowego dostarczany i podpisywany cyfrowo przez TP-Link. Ten wybór sprzętowy jest technicznym fundamentem, który umożliwia skuteczną implementację biznesowego modelu „**vendor lock-in**”, który zostanie szczegółowo omówiony w sekcji 2.5.

Poniższa tabela syntetyzuje kluczowe specyfikacje sprzętowe, które stanowią bazę dla dalszej analizy oprogramowania i funkcjonalności.

Tabela 2.1: Kluczowe Specyfikacje Techniczne TP-Link Tapo C200

Kategoria	Specyfikacja
Przetwornik Obrazu	1/2.8" Progressive Scan CMOS
Obiektyw	Ogniskowa: 4 mm, Przyśłona: F2.0
Noktowizja	Dioda IR LED 850 nm (zasięg do 40 stóp / 12 m)
Rozdzielczość	1080P HD (1920 × 1080 px)
Szybkość Klatek	30 fps
Kompresja Wideo	H.264
System Audio	Wbudowany mikrofon i głośnik
Standard Wi-Fi	IEEE802.11b/g/n, 2.4 GHz
Zapis Lokalny	Gniazdo microSD (do 512 GB)

## 2.3 Architektura Oprogramowania i Protokoły Komunikacyjne

Warstwa oprogramowania jest miejscem, w którym realizowana jest strategia producenta. To tutaj potencjał sprzętowy jest albo udostępniany poprzez otwarte standardy, albo celowo ograniczany przez zamknięte protokoły. Analiza Tapo C200 ujawnia świadome i celowe rozdzielenie tych dwóch podejść.

### Oprogramowanie Układowe (Firmware)

Urządzenie działa pod kontrolą zamkniętego (**closed-source**) oprogramowania układowego, bazującego najprawdopodobniej na zmodyfikowanej dystrybucji Linuksa, co

jest powszechną praktyką w urządzeniach IoT. Ten firmware stanowi „czarną skrzynkę”, która zarządza całym sprzętem i udostępnia wszystkie usługi sieciowe. Jak wykazano w sekcji 2.6, ten monolityczny i nieaudytowalny charakter firmware’u jest sam w sobie znaczącym wektorem ataku.

## **Standardowy Stos Sieciowy i „Iluzja Otwartości”**

Na poziomie sieciowym, kamera implementuje standardowy stos TCP/IP, aby móc funkcjonować w typowej sieci domowej. Obejmuje to podstawowe usługi, takie jak DHCP do automatycznej konfiguracji adresu IP, DNS do rozwiązywania nazw oraz NTP do synchronizacji czasu. Ponadto, kamera wykorzystuje HTTPS, co wskazuje na szyfrowaną komunikację, jednak ta komunikacja jest przeznaczona niemal wyłącznie dla serwerów chmurowych TP-Link.

Prawdziwa analiza pod kątem integracji open-source zaczyna się od protokołów warstwy aplikacji, gdzie obserwujemy strategiczną dychotomię:

**RTSP (Real-Time Streaming Protocol):** Specyfikacja techniczna potwierdza wsparcie dla RTSP. Jest to absolutnie kluczowy, otwarty i ustandaryzowany protokół, który pozwala na dostęp do surowego, skompresowanego strumienia wideo (H.264) i audio. Dostępność strumienia RTSP jest fundamentalnym umożliwicielem (enabler) dla całego projektu niniejszej pracy. To właśnie ten protokół pozwala narzędziom takim jak FFmpeg i OpenCV na przechwycenie obrazu i jego dalszą analizę w sposób całkowicie niezależny od ekosystemu producenta.

**ONVIF (Open Network Video Interface Forum):** Specyfikacja również deklaruje zgodność z ONVIF. Jest to jednak przykład strategicznego „open-washingu” – marketingowego wykorzystania otwartego standardu w sposób, który sugeruje interoperacyjność, jednocześnie jej nie dostarczając. Jak potwierdzają badania oraz liczne raporty społeczności open-source, implementacja ONVIF w Tapo C200 jest celowo okrojona. Ogranicza się ona w najlepszym razie do minimalnego zestawu funkcji (np. Profile S, co oznacza jedynie możliwość udostępniania strumienia wideo, co i tak jest już realizowane przez RTSP). Co najważniejsze, implementacja ta nie udostępnia kluczowej funkcjonalności sterowania PTZ.

**Rzeczywisty Mechanizm Sterowania: Własnościowe API** Skoro ONVIF nie pozwala na sterowanie kamerą, powstaje pytanie, w jaki sposób realizuje to oficjalna aplikacja Tapo. Odpowiedź leży w istnieniu nieudokumentowanego, własnościowego (**proprietary**) protokołu sterowania.

Badania społecznościowe wykazały, że aplikacja mobilna Tapo komunikuje się z kamerą w sieci lokalnej za pomocą niestandardowego, opartego na HTTP (lub HTTPS)

API. Wysyła ona zaszyfrowane lub zakodowane żądania w celu wykonania operacji takich jak ruch Pan/Tilt, włączenie trybu nocnego, czy zmiana ustawień detekcji.

Ten zamknięty protokół jest technicznym narzędziem egzekwowania „**vendor lock-in**”. Ponieważ jest nieudokumentowany i może ulec zmianie przy każdej aktualizacji firmware’u, uniemożliwia on standardowym, otwartym platformom (jak Home Assistant, ZoneMinder czy openHAB) natywną kontrolę nad urządzeniem.

To właśnie ta bariera zrodziła potrzebę inżynierii wstecznej (**reverse-engineering**) po stronie społeczności. Biblioteka PyTapo, która jest jednym z kluczowych narzędzi wykorzystywanych w niniejszej pracy, jest bezpośrednim rezultatem tego procesu. PyTapo implementuje logikę tego nieudokumentowanego protokołu, hermetyzując jego złożoność i pozwalając na programistyczne sterowanie kamerą z poziomu Pythona. Zależność niniejszej pracy od PyTapo jest sama w sobie dowodem na istnienie i celowość bariery w postaci zamkniętego API.

Poniższa tabela podsumowuje krytyczną analizę protokołów komunikacyjnych kamery.

Tabela 2.2: Analiza Protokołów Komunikacyjnych Tapo C200 pod kątem Integracji Open-Source

Protokół	Cel	Status	Dostępność	Użyteczność dla Projektu
RTSP	Dostęp do strumienia A/V	Otwarty Standard	Tak	Kluczowa. Stanowi podstawę do przechwytywania i analizy wideo ( <i>FFmpeg/OpenCV</i> ).
ONVIF	Interoperacyjność ( <i>Stream</i> + Sterowanie)	Otwarty Standard	Pozornie Tak	Znikoma. Implementacja jest okrojona i nie udostępnia sterowania PTZ. Nazywana „open-washingiem”.
Proprietary API	Pełne sterowanie urządzeniem ( <i>PTZ</i> , ustawienia)	Zamknięty / Własnościowy	Brak informacji	Kluczowa (Pośrednio). Wymaga <b>inżynierii wstecznej</b> . Projekt wykorzystuje <i>PyTapo</i> do obsługi tego API.
Protokół Chmurowy ( <i>HTTPS</i> )	Zdalny dostęp, alerty, <i>provisioning</i>	Zamknięty / Własnościowy	Brak informacji	Brak. Jest to mechanizm, który projekt ma na celu ominąć, aby <b>uniezależnić się od producenta</b> .

## 2.4 Analiza Możliwości Funkcjonalnych

Sekcja ta dokonuje ponownej oceny funkcji reklamowanych w sekcji 2.1, tym razem przez pryzmat inżynierski, oceniając ich rzeczywistą dostępność dla dewelopera open-source, w przeciwieństwie do ich teoretycznej obecności w urządzeniu.

### Przetwarzanie i Strumieniowanie Wideo

Ta funkcja jest w pełni dostępna. Kamera niezawodnie dostarcza wysokiej jakości strumień H.264 (1080p przy 30 fps) poprzez otwarty protokół RTSP. Z punktu widzenia projektu, jest to solidny i wystarczający fundament. Pozwala na pobranie „surowca” (danych wideo), który następnie może być przetwarzany lokalnie przez autorskie algorytmy. Dostępność ta jest warunkiem koniecznym dla powodzenia całego projektu.



## Funkcjonalność PTZ (Pan/Tilt/Zoom)

W tym przypadku obserwujemy fundamentalne rozłączenie między możliwością sprzętową a dostępnością programową. Mechanizmy (silniki) do obrotu i pochylenia są fizycznie obecne w urządzeniu. Jednak, jak ustalono w sekcji 2.3, są one niedostępne przez jakikolwiek otwarty standard, taki jak ONVIF. Dostęp do nich jest strzeżony przez własnościowe API.

W konsekwencji, z perspektywy dewelopera open-source, kamera Tapo C200 bez dodatkowej inżynierii wstecznej jest funkcjonalnie kamerą statyczną. Dopiero zastosowanie biblioteki PyTapo „odblokowuje” tę natywną funkcję sprzętową, co jest jednym z głównych celów implementacyjnych niniejszej pracy.

## Wbudowane Funkcje AI: Problem „Czarnej Skrzynki”

Najbardziej złożona sytuacja dotyczy wbudowanych funkcji „AI Detection”, takich jak wykrywanie osób i płaczu dziecka. Stanowią one istotę „Wyzwania Open Source” (tytuł sekcji 2.3 w).

Problem nie polega na tym, że te funkcje nie działają. Można założyć, że algorytmy uczenia maszynowego (prawdopodobnie uruchamiane na wyspecjalizowanym koprocesorze w ramach SoC) skutecznie analizują obraz i generują zdarzenia. Problem polega na niedostępności wyjścia tych algorytmów.

Model operacyjny TP-Link dla tych zdarzeń jest następujący:

1. Wbudowany algorytm AI na kamerze wykrywa zdarzenie (np. „osoba”).
2. Kamera nie emituje tego zdarzenia w sieci lokalnej (LAN) w formie otwartego komunikatu (np. przez MQTT, ONVIF Events, czy nawet prosty webhook).
3. Zamiast tego, kamera wysyła zaszyfrowany komunikat o zdarzeniu wyłącznie do serwerów chmurowych TP-Link.
4. Serwery TP-Link przetwarzają ten komunikat i wysyłają powiadomienie push do aplikacji mobilnej użytkownika.

Ten model, w którym metadane zdarzeń są „brane jako zakładnik” („**data hostage**”) przez infrastrukturę chmurową, czyni całą zaawansowaną, wbudowaną analitykę AI całkowicie bezużyteczną dla lokalnych systemów automatyki. Niemożliwe jest stworzenie w prosty sposób automatyzacji w systemie Home Assistant typu: „JEŻELI kamera Tapo wykryje osobę, TO włącz światło w korytarzu”.

Ta celowa blokada dostępu do danych o zdarzeniach ma kluczową implikację dla niniejszej pracy: zmusza ona do **reimplementacji** funkcjonalności, która już istnieje w urządzeniu. Skoro nie można odczytać zdarzenia „detekcja ruchu” z kamery, projekt musi sam pobrać surowy strumień wideo (przez RTSP) i przeprowadzić własną,

serwerową analizę detekcji ruchu (np. za pomocą OpenCV). Jest to kluczowe uzasadnienie dla jednego z głównych celów szczegółowych pracy – implementacji własnego algorytmu detekcji.

## 2.5 Ograniczenia i Zjawisko „Vendor Lock-in”

Synteza analizy sprzętu, oprogramowania i funkcjonalności prowadzi do jednoznacznego wniosku: ograniczenia kamery Tapo C200 nie są wynikiem braków technicznych, lecz świadomą strategią biznesową znaną jako „**vendor lock-in**” (uzależnienie od dostawcy).

Krytyka tego modelu biznesowego wskazuje, że kamera jest traktowana jako niskomarżowy „**koń trojański**”. Rzeczywistym celem nie jest jednorazowa sprzedaż sprzętu, ale „**uwięzienie**” użytkownika w zamkniętym ekosystemie Tapo, co otwiera drogę do generowania przychodów cyklicznych, np. poprzez sprzedaż subskrypcji na przechowywanie nagrań w chmurze (Tapo Care).

Z technicznego punktu widzenia, strategia „vendor lock-in” w przypadku Tapo C200 opiera się na trzech filarach:

1. **Zamknięte API Sterowania (Proprietary Control API):** Jak omówiono w sekcji 2.3, brak otwartego standardu sterowania PTZ zmusza użytkowników do korzystania wyłącznie z oficjalnej aplikacji lub polegania na niestabilnych, reverse-engineeryjnych rozwiązaniach, takich jak PyTapo.
2. **Uchwycenie Metadanych AI (AI Metadata Capture):** Jak omówiono w sekcji 2.4, przesyłanie zdarzeń detekcji wyłącznie do chmury uniemożliwia lokalną automatyzację i wymusza na użytkowniku poleganie na infrastrukturze producenta w zakresie otrzymywania alertów.
3. **Szyfrowany i Chmurowy Provisioning:** Jest to pierwszy i najbardziej fundamentalny zamek. Proces inicjalizacji kamery i jej podłączenia do sieci Wi-Fi (provisioning) jest nieudokumentowany, szyfrowany i wymaga obowiązkowej weryfikacji po stronie chmury TP-Link. Oznacza to, że kamery nie można nawet uruchomić w sieci lokalnej bez użycia oficjalnej aplikacji mobilnej i aktywnego połączenia z internetem. Jest to tak złożona bariera, że niniejsza praca musi ją zaakceptować jako ograniczenie: w założeniach projektu stwierdza się, że „praca zakłada, że kamera została jednorazowo skonfigurowana w sieci Wi-Fi przy użyciu oficjalnej aplikacji mobilnej”.

Wniosek z tej analizy jest jasny: „Wyzwanie Open Source” nie jest przypadkowym niedopatrzeniem inżynierów TP-Link. Jest to precyzyjnie zaprojektowany zestaw barier

technicznych, których celem jest ochrona modelu biznesowego firmy. Praktyczna implementacja opisana w Rozdziale 3 niniejszej pracy jest zatem w swojej istocie aktem inżynierii obchodzenia (**bypass engineering**) tych celowo narzuconych ograniczeń.

## 2.6 Aspekty Bezpieczeństwa i Prywatności

Ostatnia warstwa analizy dotyczy bezpieczeństwa i prywatności. Jest to najważniejszy argument przemawiający za koniecznością stworzenia otwartego, lokalnego rozwiązania. Model „vendor lock-in” nie tylko ogranicza funkcjonalność, ale także generuje poważne i udokumentowane zagrożenia dla użytkowników.

### Ryzyka dla Prywatności

Model operacyjny oparty na chmurze zmusza użytkownika do fundamentalnego kompromisu w zakresie prywatności. Wymaga on przesyłania wrażliwych danych – strumieni audio i wideo z wnętrza prywatnego domu – na serwery firmy trzeciej. Taka architektura generuje trzy główne ryzyka:

- **Ryzyko wycieku danych:** Pomyślny atak na infrastrukturę chmurową TP-Link mógłby skutkować masowym wyciekiem prywatnych nagrań tysięcy użytkowników.
- **Ryzyko nadużycia:** Użytkownik traci suwerenność nad swoimi danymi i musi ufać, że pracownicy dostawcy lub jego podwykonawcy nie uzyskają nieautoryzowanego dostępu do jego strumieni.
- **Ryzyko prawne:** Dane przechowywane w chmurze podlegają jurysdykcji prawnej kraju, w którym znajdują się serwery, i mogą być przedmiotem żądań organów ścigania bez wiedzy użytkownika.

Lokalne rozwiązanie, do którego dąży niniejsza praca, całkowicie eliminuje te ryzyka, ponieważ dane nigdy nie opuszczają sieci lokalnej użytkownika.

### Zidentyfikowane Luki w Zabezpieczeniach

Zamknięty, nieaudytowalny firmware kamery Tapo C200 okazał się być podatny na krytyczne luki bezpieczeństwa. Nie jest to już teoretyczne ryzyko; jest to udokumentowany fakt.

#### 1. CVE-2021-4045: Krytyczna Luka RCE

Najpoważniejszą znaną luką jest CVE-2021-4045, której przyznano ocenę 9.8 (KRYTYCZNA) w skali CVSS.

- **Problem:** Luka typu „unauthenticated Remote Code Execution” (nieuwierzytelnione zdalne wykonanie kodu).
- **Wektor:** Luka znajduje się w binarnym pliku uhttpd – tym samym wbudowanym serwerze WWW, który jest używany do obsługi... własnościowego API sterującego.
- **Wpływ:** Serwer uhttpd działa z uprawnieniami użytkownika root (najwyższymi możliwymi). Oznacza to, że nieuwierzytelniony atakujący w tej samej sieci (np. gość korzystający z Wi-Fi) mógł zdalnie przejąć całkowitą kontrolę nad kamerą. Mógł ją wyłączyć, podsłuchiwać, podglądać, a także – co być może najgroźniejsze – wykorzystać ją jako „przyczółek” (beachhead) do ataku na inne urządzenia w sieci lokalnej użytkownika (np. komputer lub dysk NAS).
- **Zasięg:** Luka dotyczyła oprogramowania w wersji 1.1.15 i starszych.

## 2. Inne Wyniki Testów Penetracyjnych

Niezależne badania bezpieczeństwa potwierdziły istnienie wielu innych słabości:

- Badanie Ariefianto / Biondi et al., w ramach którego opracowano metodykę PETIoT, wykorzystało Tapo C200 jako studium przypadku i zidentyfikowało trzy nieznane wcześniej (zero-day) luki: Denial of Service (DoS), podsłuchiwanie strumienia wideo (video eavesdropping) oraz nowy typ ataku nazwany „Motion Oracle”.
- Inna praca dyplomowa (KTH) przeprowadzająca testy penetracyjne C200, zidentyfikowała podatności na ataki typu brute force, RCE, Man-in-the-Middle (MITM) oraz replay attack, wskazując na fundamentalne problemy z szyfrowaniem firmware’u i protokołami komunikacyjnymi.
- Ogólnym zagrożeniem dla wszystkich słabo zabezpieczonych urządzeń IoT, w tym kamer, jest ryzyko rekrutacji do botnetu (np. Mirai), który wykorzystuje ich moc obliczeniową do przeprowadzania zmasowanych ataków DDoS.

## Wniosek Końcowy: Związek „Vendor Lock-in” z Lukami w Zabezpieczeniach

Niniejsza analiza wykazuje istnienie bezpośredniego związku przyczynowego między modelem biznesowym „vendor lock-in” a katastrofalnymi lukami bezpieczeństwa.

Logika jest następująca:

1. Aby zrealizować strategię „vendor lock-in”, TP-Link musiał zrezygnować z otwartego standardu ONVIF do sterowania.
2. Wymusiło to stworzenie własnościowego, zamkniętego API.

3. Aby to API było dostępne, kamera musi uruchamiać niestandardowy, wbudowany serwer WWW (uhttpd).
4. Aby ten serwer mógł kontrolować sprzęt (silniki PTZ, diody IR), musiał otrzymać najwyższe uprawnienia systemowe (root).
5. W ten sposób stworzono idealny wektor ataku: skomplikowaną, nieaudytowaną, autorską usługę sieciową działającą z maksymalnymi uprawnieniami.
6. Dokładnie w tym miejscu – w serwerze uhttpd – odkryto krytyczną lukę RCE (CVE-2021-4045).

Wniosek: To nie przypadek. To decyzja biznesowa o zamknięciu ekosystemu bezpośrednio doprowadziła do stworzenia architektury oprogramowania, która była fundamentalnie niebezpieczna.

## **Ostateczne Uzasadnienie dla Projektu**

Powyższa analiza bezpieczeństwa i prywatności stanowi ostateczne i najsilniejsze uzasadnienie dla celu niniejszej pracy. Projektowane rozwiązanie open-source nie jest jedynie ćwiczeniem z inżynierii wstecznej w celu odblokowania funkcji PTZ. Jest to fundamentalna interwencja w zakresie bezpieczeństwa.

Tworząc w pełni funkcjonalny, lokalny serwer sterujący, rozwiązanie to daje użytkownikowi możliwość wykonania kluczowego kroku hardeningu: całkowitego zablokowania kamery dostępu do Internetu na poziomie routera (firewalla).

Taka konfiguracja, niemożliwa przy korzystaniu z oficjalnej aplikacji, natychmiast:

- Rozwiązuje problem prywatności: Dane audio/wideo nigdy nie opuszczają sieci lokalnej.
- Neutralizuje ryzyko botnetu: Kamera nie może komunikować się z serwerami C&C (Command and Control).
- Omija podatny na ataki serwer: Użytkownik komunikuje się z bezpiecznym, audytowanym serwerem Python (rozwiązaniem z pracy), zamiast z dziurawym, działającym jako root uhttpd.

Rozdział ten udowodnił, że TP-Link Tapo C200 jest idealnym studium przypadku konfliktu IoT. Stanowi on techniczne uzasadnienie, dlaczego proponowana w niniejszej pracy architektura – lokalna, oparta na otwartym oprogramowaniu i przywracająca użytkownikowi kontrolę – jest nie tylko pożądana z punktu widzenia funkcjonalności, ale wręcz konieczna z punktu widzenia prywatności i cyberbezpieczeństwa.

### 3 Metodologia i implementacja rozwiązania

Poprzednie rozdziały dokonały teoretycznej dekonstrukcji technologii kamer IP (Rozdział 1) oraz przeprowadziły szczegółową analizę studium przypadku — kamery TP-Link Tapo C200 (Rozdział 2). Analiza ta zidentyfikowała kluczowy problem badawczy: fundamentalny konflikt między potencjałem sprzętowym urządzenia a ograniczeniami narzuconymi przez zamknięty ekosystem producenta (tzw. „**vendor lock-in**”).

Niniejszy rozdział przechodzi od teorii do praktyki. Stanowi on techniczną odpowiedź na zdefiniowane wyzwania. Opisany zostanie kompletny proces projektowy i wdrożeniowy – od wybranej metodyki badawczej, przez architekturę systemu, aż po szczegóły implementacji poszczególnych komponentów. Celem jest budowa autorskiego, otwartego rozwiązania programistycznego, które uwalnia pełen potencjał kamery i realizuje cele postawione w niniejszej pracy.

#### 3.1 Metodyka Projektowa

##### 3.1.1 Double Diamond

Model Double Diamond (Podwójny Diament) jest ustrukturyzowaną metodyką procesową, pierwotnie sformalizowaną przez British Design Council w 2005 roku. Stanowi ona mapę procesu projektowego, którego celem jest efektywne nawigowanie od wstępnej idei do wdrożonego rozwiązania, przy jednoczesnym zarządzaniu złożonością i niepewnością.

Metodyka ta jest fundamentalna dla współczesnego projektowania (w tym inżynierii oprogramowania, projektowania produktów i usług) i bazuje na koncepcji myślenia projektowego (Design Thinking).

Nazwa modelu pochodzi od jego wizualnej reprezentacji jako dwóch sąsiadujących rombów („diamentów”). Każdy diament reprezentuje sekwencję dwóch typów myślenia:

- **Myślenie Rozbieżne:** Faza otwierania „diamentu”. Polega na eksploracji, generowaniu dużej liczby pomysłów, zbieraniu szerokiego spektrum danych i powstrzymywaniu się od oceny. Celem jest poszerzenie perspektywy i zrozumienie kontekstu.
- **Myślenie Zbieżne:** Faza zamykania „diamentu”. Polega na syntezie, analizie, krytycznej ocenie i podejmowaniu decyzji. Celem jest zawężenie opcji i wyłonienie konkretnego kierunku działania.

Model zakłada, że aby opracować właściwe rozwiązanie, należy najpierw dogłębnie zrozumieć i zdefiniować właściwy problem.

## **Diament 1: Przestrzeń Problemu**

Celem tego etapu jest zidentyfikowanie i precyzyjne zdefiniowanie kluczowego problemu, który ma zostać rozwiązany.

1. **Faza Odkrywania (Discover) – Dywergencja** Jest to faza intensywnych badań (research) i empatii. Zespół projektowy wychodzi poza własne założenia, aby zrozumieć rzeczywisty kontekst użytkownika i zidentyfikować jego niezaspokojone potrzeby.
2. **Faza Definiowania (Define) – Konwergencja** W tej fazie następuje synteza danych zebranych podczas Odkrywania. Zespół filtruje i analizuje informacje, szukając wzorców i kluczowych wyzwań. Celem jest przekształcenie rozproszonych obserwacji w klarowną i mierzalną definicję problemu.

## **Diament 2: Przestrzeń Rozwiązania**

3. **Faza Rozwijania (Develop) – Dywergencja** Mając jasno zdefiniowany problem, zespół ponownie przechodzi w tryb dywergencyjny, aby wygenerować jak najszerszy wachlarz potencjalnych rozwiązań. Kładzie się nacisk na ilość, a nie jakość, oraz na kreatywność i multidyscyplinarność.
4. **Faza Dostarczania (Deliver) – Konwergencja** Jest to ostatnia faza, skupiona na testowaniu, walidacji i iteracyjnym udoskonalaniu wybranych koncepcji. Rozwiązania są poddawane rygorystycznym testom z udziałem użytkowników, aby zidentyfikować błędy i obszary do poprawy, a następnie zawęzić wybór do jednego, optymalnego rozwiązania gotowego do wdrożenia.

## **Zastosowanie modelu w niniejszej pracy**

1. **Odkrywanie (Discover):** Tę fazę reprezentuje research przeprowadzony w Rozdziałach 1 i 2. Zbadano ogólne działanie kamer IP, a następnie przeanalizowano specyfikę Tapo C200, identyfikując jej otwarte porty (RTSP) oraz zamknięte, własnościowe API do sterowania PTZ.
2. **Definiowanie (Define):** Zebrane informacje skumulowano do konkretnego problemu: **vendor lock-in** uniemożliwia lokalną kontrolę. Celem pracy stało się więc stworzenie lokalnego systemu dającego pełną kontrolę.
3. **Rozwój (Develop):** W tej fazie nastąpił brainstorming nad architekturą rozwiązania. Rozważano różne technologie i narzędzia (np. gotowe platformy vs. własna aplikacja). Zdecydowano się na elastyczny stos technologiczny, który umożliwi realizację wszystkich celów.

4. **Dostarczanie (Deliver):** Wybrano konkretne, optymalne rozwiązanie: aplikacja webowa oparta na Pythonie, Flasku i WebSockets, wykorzystująca bibliotekę Py-Tapo (do sterowania) oraz OpenCV i FFmpeg (do analizy wideo), całość hermetyzowana w Dockerze. Implementacja tego rozwiązania stanowi dalszą część niniejszego rozdziału.

## 3.2 Architektura rozwiązania

System został zaprojektowany jako **Real-Time IoT Gateway**, stanowiąca pomost między klientami internetowymi o wysokim opóźnieniu a niskopoziomowymi protokołami sprzętowymi. U swej podstawy aplikacja opiera się na **Architekturze Trójwarstwowej** (*Three-Tier Multitier Architecture*), ściśle oddzielając Warstwę Prezentacji (Klient), Logikę Aplikacji (Middleware) oraz Warstwę Danych/Sprzętową (Źródło). Taka separacja zapewnia, że złożoność własnościowych protokołów kamery zostaje całkowicie abstrahowana i ukryta przed interfejsem użytkownika końcowego.

Kluczową decyzją projektową była implementacja aplikacji w języku Python jako inteligentnej **warstwy pośredniej** (*Middleware*). Ponieważ nowoczesne przeglądarki internetowe nie są w stanie natywnie obsługiwać surowych strumieni wideo RTSP ani komunikować się bezpośrednio za pomocą protokołu ONVIF, warstwa pośrednia działa jako dwukierunkowy translator protokołów. Pobiera ona synchroniczne strumienie sprzętowe i przekształca je w asynchroniczne zdarzenia WebSocket. Pozwala to na uzyskanie responsywnego doświadczenia użytkownika bez konieczności ujawniania danych uwierzytelniających sprzętu czy jego adresu IP w sieci publicznej.

W celu sprostania specyficznym wymaganiom przetwarzania audiowizualnego, system wykorzystuje hybrydę 2 wzorców architektonicznych. Pierwszym z nich jest wzorzec **Architektury Potokowej** (ang. *Pipe and Filter*). Zastosowany w warstwie przetwarzania do sekwencyjnej obsługi klatek wideo i fragmentów audio (Akwizycja → Przetwarzanie → Kodowanie → Emisja).

Kolejnym wzorcem architektonicznym jest **Architektura Sterowana Zdarzeniami** (*Event-Driven Architecture*), która umożliwia komunikację w czasie rzeczywistym między klientem a serwerem. Akcje użytkownika (takie jak sterowanie PTZ) oraz zmiany stanu systemu (np. wykrycie ruchu) są propagowane natychmiastowo poprzez magistralę zdarzeń (WebSockets), zamiast polegać na cyklicznym odpytywaniu (*polling*).

Ze względu na ciągły charakter strumieniowania wideo, architektura systemu w znacznym stopniu polega na **wielowątkowości** (*threaded concurrency*). Aplikacja utrzymuje współdzielony stan w pamięci operacyjnej (*shared in-memory state*), co pozwala na odseparowanie szybkich pętli wejściowych (odczyt ze sprzętu) od obsługi żądań wyjściowych (obsługa klientów webowych). Gwarantuje to, że obciążające procesor zadania, takie jak detekcja ruchu czy muksowanie wideo, nie blokują interfejsu użyt-



kownika.

W kolejnych sekcjach szczegółowo omówiono odpowiedzialności poszczególnych warstw (Prezentacji, Logiki i Danych), przeanalizowano wewnętrzny przepływ danych w potokach medialnych oraz przedstawiono strukturę klas wykorzystaną do implementacji powyższej architektury.

Rysunek 3.1: Schemat architektury rozwiązania (Klient - Middleware - Sprzęt)

### 3.2.1 Architektura Wielowarstwowa

Współczesna inżynieria systemów **Internetu Rzeczy** (*IoT*), a w szczególności projektowanie bram sieciowych (*IoT Gateways*) obsługujących strumieniowanie multimediów w czasie rzeczywistym, wymaga rygorystycznego podejścia do strukturalizacji kodu oraz zarządzania przepływem danych. W ramach niniejszej pracy inżynierskiej, jako fundament logiczny i fizyczny rozwiązania, przyjęto **Architekturę Trójwarstwową**.

Architektura warstwowa jest powszechnie uznawana w literaturze przedmiotu za *de facto* standard w projektowaniu aplikacji korporacyjnych i systemów rozproszonych, umożliwiając dekompozycję złożonego problemu na separowalne, zarządzalne poziomy abstrakcji. W kontekście systemów IoT, model ten ewoluuje w kierunku struktur typu **Edge-Fog-Cloud**, gdzie brama (*Gateway*) pełni rolę kluczowego węzła pośredniczącego. Zastosowany w projekcie model trójwarstwowy dokonuje ścisłej separacji odpowiedzialności (*Separation of Concerns* - *SoC*) pomiędzy interakcją z użytkownikiem, logiką biznesową przetwarzania sygnału oraz fizycznym dostępem do urządzenia.

Poniższa tabela (Tabela 3.1) przedstawia szczegółowy podział odpowiedzialności oraz stos technologiczny wykorzystany w poszczególnych warstwach systemu.

W dalszej części rozdziału przeprowadzona zostanie szczegółowa analiza każdej z warstw, z naciskiem na uzasadnienie doboru technologii oraz analizę wydajnościową przyjętych rozwiązań.

**3.2.1.1 Warstwa Prezentacji** Warstwa Prezentacji stanowi najwyższy poziom abstrakcji w systemie, będąc jedynym punktem styku użytkownika z infrastrukturą monitoringu. Zaproponowane rozwiązanie opiera się na modelu **cienkiego klienta** (*thin client*). Podejście to zakłada, że przeglądarka internetowa odpowiada wyłącznie za renderowanie obrazu i przesyłanie zdarzeń sterujących, podczas gdy ciężar obliczeniowy związany z dekodowaniem, analizą wizyjną i zarządzaniem bezpieczeństwem spoczywa na serwerze.

Wybór architektury *thin client* dla warstwy prezentacji jest podyktowany specyfiką środowiska IoT oraz dążeniem do uniwersalności. Klienci „cienicy” charakteryzują się mniejszymi wymaganiami sprzętowymi po stronie użytkownika, co umożliwia dostęp

Tabela 3.1: Podział warstw architektury systemu IoT

Poziom Archi- tektury (Tier)	Rola w Systemie IoT	Implementacja (Stack Technolo- giczny)	Odpowiedzialność Funkcjonalna
<b>Tier 1:</b> Warstwa Prezen- tacji ( <i>Presenta- tion Layer</i> )	Interfejs Użytkownika (GUI), wizualizacja da- nych, obsługa zdarzeń wejściowych.	<b>Klient Webowy (SPA):</b> HTML5, Ja- vaScript, Socket.IO Client, HTML5 Ca- nvas.	Renderowanie stru- mienia wideo (MJPE- G/Canvas), panel ste- rowania PTZ, wyświe- tlanie alertów detekcji ruchu.
<b>Tier 2:</b> Warstwa Logiki ( <i>Business Logic / Middleware</i> )	Przetwarzanie reguł, koordynacja procesów, analiza danych, trans- lacja protokołów.	<b>Serwer Aplikacyjny:</b> Python 3.13, Flask, Flask-SocketIO, OpenCV ( <i>Computer Vision</i> ), Multiproces- sing.	Detekcja ruchu ( <i>Back- ground Subtraction</i> ), obsługa sesji WebSoc- ket, buforowanie klat- tek, orkiestracja wątków.
<b>Tier 3:</b> Warstwa Danych i Sprzętu ( <i>Data- /Hardware Layer</i> )	Fizyczny dostęp do danych, abstrakcja sprzętowa, trwała pa- mięć masowa.	<b>HAL &amp; DAO:</b> PyTapo (Driver), FFmpeg (Vi- deo Capture), System Plików (Storage).	Komunikacja z API Tapo, obsługa stru- mienia RTSP, zapis nagrań na dysk, zarzą- dzanie poświadcze- niami.

do systemu monitoringu z szerokiego spektrum urządzeń – od wydajnych stacji roboczych, po budżetowe smartfony i tablety, bez konieczności instalacji dedykowanego oprogramowania. Co więcej, centralizacja logiki na serwerze ułatwia zarządzanie bezpieczeństwem – wrażliwe algorytmy detekcji oraz bezpośrednie poświadczenia do kamery nigdy nie opuszczają bezpiecznej strefy serwera, co drastycznie redukuje wektor ataku.

W omawianym rozwiązaniu Warstwa Prezentacji została zaimplementowana jako responsywna aplikacja internetowa, wykorzystująca standardowe technologie webowe (HTML5, CSS3, JavaScript), co eliminuje konieczność instalacji dodatkowych wtyczek po stronie klienta. Architektura interfejsu została podzielona na dwa funkcjonalne moduły, różniące się sposobem obsługi mediów:

### 1. Moduł czasu rzeczywistego (Live Dashboard)

W przypadku podglądu na żywo zrezygnowano z tradycyjnego podejścia opartego na znaczniku wideo HTML5 (<video>), który ze względu na wewnętrzne mechanizmy buforowania przeglądarki wprowadza opóźnienia nieakceptowalne przy sterowaniu kamerą. Zamiast tego zastosowano renderowanie imperatywne na elemencie **Canvas**. Mechanizm ten działa w ścisłej symbiozie z warstwą lo-

giczną serwera:

**Wizualizacja:** Przeglądarka traktowana jest jako dynamiczne płótno, na którym rysowane są poszczególne ramki obrazu otrzymywane asynchronicznie poprzez kanał **WebSocket**. Pozwala to na zachowanie płynności i minimalizację opóźnień (*low-latency*), gdyż pomijany jest narzut związany z konteneryzacją strumienia wideo.

**Interakcja:** Sterowanie mechaniką kamery (PTZ) oraz procesem nagrywania zrealizowano w oparciu o model zdarzeniowy. Akcje użytkownika nie powodują przeładowania strony, lecz generują natychmiastowe sygnały sterujące wysyłane do serwera.

**Audio:** Warstwa Prezentacji przejmuje również odpowiedzialność za rekonstrukcję dźwięku. Surowe dane audio są buforowane i odtwarzane przy użyciu **Web Audio API**, co pozwala na synchronizację ścieżki dźwiękowej z obrazem bez obciążania łącza przesyłaniem ciężkich plików multimedialnych.

## 2. Moduł archiwizacji

Dla funkcjonalności przeglądania nagrań historycznych zastosowano odmienną strategię. Ponieważ w tym przypadku priorytetem jest stabilność odtwarzania i możliwość przewijania materiału, a nie czas reakcji, warstwa prezentacji wykorzystuje natywne możliwości przeglądarki do odtwarzania plików wideo (MP4) serwowanych standardowym protokołem HTTP.

Całość interfejsu została zaprojektowana zgodnie z zasadami **Responsive Web Design** (RWD), co zapewnia spójne doświadczenie użytkownika niezależnie od tego, czy system obsługiwany jest na monitorze stacji roboczej, czy na ekranie dotykowym urządzenia mobilnego. Rolę medium transportowego dla danych dynamicznych pełni tutaj biblioteka klienta WebSocket, która zarządza stabilnością połączenia z warstwą pośrednią (*Middleware*).

**3.2.1.2 Warstwa Logiki** Warstwa Logiki, umiejscowiona centralnie w architekturze trójwarstwowej, pełni rolę „mózgu” systemu, orkiestrując przepływ danych pomiędzy użytkownikiem a sprzętem. W literaturze dotyczącej IoT warstwa ta jest często definiowana jako *Middleware*, którego zadaniem jest ukrycie heterogeniczności urządzeń końcowych i udostępnienie ujednoliconych usług dla warstwy aplikacji. W niniejszym projekcie warstwa ta została zaimplementowana w języku **Python 3.13** z wykorzystaniem frameworka **Flask**, działając jako inteligentna brama (*Smart Gateway*) realizująca nie tylko routing, ale i zaawansowane przetwarzanie danych na brzegu sieci (*Edge Computing*).

**Middleware jako Translator Protokołów i Agregator** Głównym zadaniem Warstwy Logiki jest integracja różnorodnych standardów komunikacyjnych. Kamera Tapo C200 operuje na dwóch niezależnych kanałach: standardowym strumieniu wideo RTSP (port 554) oraz zamkniętym, szyfrowanym protokole sterowania HTTP (*proprietary API*). Warstwa Logiki działa tutaj jako **Translator Protokołów**, konwertując te nie-spójne interfejsy na jednolity strumień zdarzeń WebSocket zrozumiałych dla klienta webowego. *Middleware* realizuje następujące funkcje kluczowe:

**Ingestia i Transkodowanie:** Pobiera strumień H.264 z kamery, dekoduje go i transkoduje do formatu MJPEG/JPEG w czasie rzeczywistym, co jest niezbędne, gdyż przeglądarki nie obsługują natywnie surowego RTSP.

**Abstrakcja Sterowania:** Tłumaczy wysokopoziomowe polecenia (np. `move_camera('left')`) na specyficzne, zaszyfrowane payloady wymagane przez API Tapo, wykorzystując biblioteki warstwy niższej.

**Zarządzanie Sesjami:** Monitoruje stan połączeń klientów. Dzięki architekturze stanowej (*Stateful*), system może wstrzymać pobieranie strumienia z kamery, gdy żaden użytkownik nie jest podłączony, co znacząco oszczędza zasoby sieciowe i obliczeniowe bramy.

**Wyzwanie Współbieżności: Python GIL i Model Przetwarzania** Krytycznym aspektem projektowym w Warstwie Logiki był wybór modelu współbieżności. Język Python, mimo swojej elastyczności i bogatego ekosystemu bibliotek (OpenCV, NumPy), posiada istotne ograniczenie w postaci **Global Interpreter Lock** (GIL). GIL to mechanizm w referencyjnej implementacji CPython, który wymusza, aby w danym momencie tylko jeden wątek wykonywał kod bajtowy Pythona. W systemie przetwarzania wideo, który musi jednocześnie odbierać dane z sieci (I/O), analizować obraz (CPU) i wysyłać go do klientów (I/O), GIL staje się poważnym wąskim gardłem, uniemożliwiającym pełne wykorzystanie procesorów wielordzeniowych w modelu wielowątkowym.

Aby rozwiązać ten problem i zapewnić wysoką wydajność (wysoki FPS, niskie opóźnienia), w projekcie zastosowano hybrydowy model współbieżności, oparty na analizie charakterystyki zadań:

**Wielowątkowość (threading) dla zadań I/O-bound:** Obsługa serwera WWW, gniazd WebSocket oraz komunikacji sieciowej z kamerą jest realizowana za pomocą wątków (przy użyciu biblioteki `eventlet` lub `gevent` zintegrowanej z Flask-SocketIO). Operacje wejścia/wyjścia w Pythonie zwalniają blokadę GIL, co pozwala na efektywną obsługę wielu równoległych połączeń sieciowych w jednym procesie.

**Wieloprocusowość (multiprocessing) dla zadań CPU-bound:** Kluczowe zadania obliczeniowe, takie jak dekodowanie klatek wideo i algorytmy detekcji ruchu (*Computer Vision*), zostały wydzielone do osobnych procesów systemowych. Każdy proces posiada własną instancję interpretera Python i własny GIL, co pozwala na rzeczywiste równoległe wykonywanie kodu na dostępnych rdzeniach procesora (np. 4 rdzenie w Raspberry Pi). Badania wskazują, że dla zadań przetwarzania obrazu w OpenCV, model wieloprocusowy oferuje liniowy wzrost wydajności, podczas gdy model wątkowy jest ograniczony przez GIL i nie skaluje się.

**Implementacja Wzorca Producent-Konsument** Przepływ danych wideo wewnątrz Warstwy Logiki został zorganizowany zgodnie z asynchronicznym wzorcem projektowym **Producent-Konsument** (*Producer-Consumer Pattern*), co zapewnia separację procesu akwizycji danych od ich dystrybucji.

**Proces Producenta:** Dedykowany proces, który w pętli nieskończonej pobiera klatki ze strumienia RTSP (używając `cv2.VideoCapture`). Jego wyłącznym zadaniem jest utrzymanie stabilnego połączenia z kamerą i umieszczanie najnowszej klatki w pamięci współdzielonej (kolejce).

**Bufor (Kolejka):** Zastosowano kolejkę o bardzo małym rozmiarze (np. `maxsize=1`). Jest to celowy zabieg inżynierski wynikający ze specyfiki *live streamingu*: w systemie monitoringu ważniejsza jest minimalizacja opóźnień (*latency*) niż zachowanie każdej klatki. Jeśli proces konsumenta (serwer wysyłający) jest zbyt wolny, kolejka automatycznie odrzuca stare klatki, zapewniając, że klient zawsze otrzymuje najbardziej aktualny obraz („gubienie klatek” zamiast „buforowania opóźnień”).

**Proces Konsumenta:** Wątek serwera Flask, który pobiera klatkę z kolejki, aplikuje na nią logikę biznesową (np. detekcję ruchu) i emituje do klientów WebSocket.

Taka architektura zapewnia izolację błędów – awaria lub spowolnienie po stronie klienta webowego nie wpływa na stabilność połączenia z kamerą, a chwilowe problemy z siecią kamery nie blokują interfejsu użytkownika.

**3.2.1.3 Warstwa Danych** Najniższy poziom architektury stanowi fundament integrujący system cyfrowy ze światem fizycznym. W klasycznej inżynierii oprogramowania warstwa ta (*Data Access Layer* - DAL) odpowiada za komunikację z bazą danych. W systemach IoT pojęcie to ulega rozszerzeniu o **Warstwę Abstrakcji Sprzętu** (*Hardware Abstraction Layer* - HAL). W projekcie przyjęto założenie, że kamera IP jest specyficznym rodzajem „bazy danych”, która dostarcza strumień informacji (wideo, audio, telemetria) i przyjmuje polecenia modyfikacji stanu (PTZ, konfiguracja).

**Warstwa Abstrakcji Sprzętu (HAL) jako Izolator** Podstawowym celem implementacji HAL jest uniezależnienie wyższych warstw systemu od konkretnego modelu sprzętowego. Warstwa Logiki nie powinna operować na niskopoziomowych szczegółach, takich jak adresy URL strumieni RTSP, algorytmy szyfrowania haseł czy specyficzne kody błędów HTTP zwracane przez kamerę. Zamiast tego, HAL udostępnia ujednolicony interfejs programistyczny (API wewnętrzne), np. metodę `camera.move_left()`, która „pod spodem” wykonuje całą komunikacyjną „brudną robotę”.

W projekcie HAL realizowany jest poprzez wzorzec **Adapter**, który „opakowuje” zewnętrzną bibliotekę PyTapo. PyTapo jest efektem inżynierii wstecznej społeczności *Open Source* i służy do komunikacji z zamkniętym API Tapo. Bezpośrednie użycie PyTapo w kontrolerach frameworka Flask naruszyłoby zasadę separacji odpowiedzialności. Stworzenie własnej klasy typu *wrapper* (TapoCameraDriver) w warstwie HAL pozwala na:

**Łatwą wymianę sterownika:** Jeśli w przyszłości biblioteka PyTapo przestanie być rozwijana, wystarczy podmienić implementację wewnątrz klasy TapoCameraDriver na inną, bez konieczności przepisywania setek linii kodu w Warstwie Logiki.

**Centralizację obsługi błędów:** HAL tłumaczy specyficzne wyjątki sieciowe (np. `ConnectionRefused` czy kody błędów z serwera `uhttpd` kamery) na zrozumiałe wyjątki domenowe (np. `CameraOfflineException`), upraszczając logikę obsługi błędów w wyższych warstwach.

**Bezpieczeństwo:** HAL odpowiada za bezpieczne przechowywanie i wstrzykiwanie poświadczeń (login/hasło) do żądań. Dzięki temu dane uwierzytelniające nigdy nie „wyciekają” do warstwy prezentacji. Jest to kluczowe w kontekście znanych podatności kamer Tapo, takich jak CVE-2021-4045 (luka RCE w serwerze `uhttpd`), która wymusza traktowanie urządzenia jako potencjalnie niebezpiecznego i izolowanie interakcji z nim.

### 3.2.2 Wzorzec Architektury Potokowej

Uzupełnieniem struktury warstwowej w warstwie logiki biznesowej jest zastosowanie **Architektury Potokowej** (ang. *Pipe and Filter*). Wzorzec ten jest standardem w systemach przetwarzających strumienie danych multimedialnych, gdzie kluczowe jest zachowanie ciągłości i niskiego opóźnienia przetwarzania.

**3.2.2.1 Zasada działania** Istotą tego wzorca jest dekompozycja złożonego procesu przetwarzania danych na serię niezależnych, sekwencyjnych kroków (filtrów), połączonych kanałami komunikacyjnymi (potokami). Dane wejściowe wchodzą do systemu,

przechodzą przez szereg transformacji, a następnie są przekazywane do wyjścia (ujścia). Kluczową cechą tej architektury jest izolacja poszczególnych filtrów – każdy z nich wykonuje tylko jedną, specyficzną operację (np. skalowanie obrazu, detekcja ruchu) i przekazuje zmodyfikowane dane do następnego elementu w łańcuchu. Takie podejście sprzyja modularności, ułatwia testowanie oraz pozwala na łatwą wymianę algorytmów bez naruszania struktury całego systemu.

**3.2.2.2 Zastosowanie w projekcie** W zrealizowanym systemie wzorzec ten stanowi fundament działania klas `VideoStreamer` oraz `AudioStreamer`, które operują w nieskończonych pętlach wątków tła. Ponieważ dane z kamery (protokół RTSP) napływają w sposób ciągły, każda jednostka danych (klatka wideo lub pakiet audio) musi zostać przetworzona w czasie rzeczywistym, zanim zostanie nadpisana przez kolejną. Architektura potokowa zapewnia tutaj deterministyczny przepływ danych od momentu ich akwizycji ze sprzętu aż do momentu wysłania do klienta webowego lub zapisu na dysku.

Wzorzec ten został zaimplementowany w następujących obszarach systemu:

- **Potok Wideo:** Przekształcanie surowych macierzy pikseli w obrazy JPEG wyświetlane w przeglądarce.
- **Potok Audio:** Dekodowanie, resampling i miksowanie kanałów dźwiękowych.
- **Potok Rejestracji:** Buforowanie ramek w pamięci RAM i ich finalna kompozycja do pliku MP4 (realizowana przez bibliotekę `MoviePy`).

**3.2.2.3 Przykład implementacji: Potok przetwarzania wideo** Najbardziej reprezentatywnym przykładem wykorzystania tego wzorca w projekcie jest pętla przetwarzania obrazu zaimplementowana w klasie `VideoStreamer`. Proces ten można przedstawić jako sekwencję pięciu filtrów:

1. **Źródło (*Source*):** Metoda `camera.read_frame()` dokonuje akwizycji surowej klatki obrazu bezpośrednio ze sterownika sprzętowego.
2. **Filtr Optymalizacyjny:** Surowy obraz, często o wysokiej rozdzielczości natywnej, jest poddawany operacji skalowania (`cv2.resize`). Zmniejszenie rozdzielczości na tym etapie jest krytyczne dla wydajności kolejnych kroków analizy i transmisji.
3. **Filtr Analityczny (*Motion Detection*):** Przeskalowana klatka trafia do modułu `MotionDetector`. Jest ona porównywana z modelem tła (średnią kroczącą z poprzednich klatek). Wynikiem tego filtra nie jest modyfikacja obrazu, lecz wygenerowanie metadanych (flaga `is_motion`), które sterują logiką powiadomień.

4. **Filtr Kodujący:** Obraz będący macierzą pikseli (format BGR) jest kompresowany do formatu JPEG (`cv2.imencode`). Jest to niezbędny krok transformacji danych do formatu zrozumiałego dla przeglądarek internetowych.
5. **Ujście (*Sink*):** Zakodowany obraz, wraz z metadanymi o detekcji ruchu, jest przekazywany do warstwy transportowej (`socketio.emit`), która dystrybuje go do wszystkich podłączonych klientów.

Dzięki zastosowaniu architektury potokowej, dodanie nowej funkcjonalności – np. rozpoznawania twarzy – sprowadzałoby się jedynie do wpięcia nowego „filtra” pomiędzy etap skalowania a kodowania, bez konieczności modyfikacji logiki pobierania obrazu czy komunikacji sieciowej.

### 3.2.3 Wzorzec Architektury Opartej na Zdarzeniach

Trzecim filarem architektonicznym omawianego systemu, odpowiedzialnym za interaktywność i komunikację między warstwami, jest **Architektura Oparta na Zdarzeniach** (ang. *Event-Driven Architecture* – EDA). W przeciwieństwie do klasycznego modelu żądanie-odpowiedź (*Request-Response*), typowego dla statycznych stron WWW, model ten zakłada, że przepływ sterowania w systemie jest determinowany przez wystąpienie określonych zdarzeń (akcji użytkownika, zmian stanu czujników), a nie przez sekwencyjny kod proceduralny.

**3.2.3.1 Zasada działania** Istotą EDA jest odwrócenie zależności komunikacyjnych. Komponenty systemu nie odpytują się wzajemnie o zmianę stanu (co generowałoby zbędny ruch sieciowy i opóźnienia), lecz oczekują na nadejście sygnału. Wzorzec ten składa się z trzech głównych elementów:

**Producent Zdarzenia (*Event Producer*):** Komponent, który wykrywa zmianę (np. naćśnięcie przycisku, wykrycie ruchu) i emituje komunikat. Producent nie musi wiedzieć, kto i w jaki sposób obsłuży to zdarzenie.

**Kanał Zdarzeń (*Event Channel*):** Medium transportowe, które przekazuje zdarzenie od producenta do konsumenta. W projekcie rolę tę pełni biblioteka `Flask-SocketIO` działająca na protokole **WebSocket**.

**Konsument Zdarzenia (*Event Consumer*):** Komponent, który nasłuchuje na określony typ zdarzenia i w reakcji na nie uruchamia odpowiednią logikę biznesową.

**3.2.3.2 Zastosowanie w projekcie** W zrealizowanym systemie bramy IoT, architektura sterowana zdarzeniami została wykorzystana jako główny mechanizm komunikacji dwukierunkowej (*Full-Duplex*) między Warstwą Prezentacji (przeglądarką) a Warstwą



Logiki (serwerem Python). Zastosowanie tego wzorca było niezbędne do osiągnięcia niskiej latencji (opóźnienia) wymaganej przy zdalnym sterowaniu mechanicznym oraz do natychmiastowego powiadamiania użytkownika o zagrożeniach. Wzorzec ten obsługuje trzy kluczowe obszary funkcjonalne:

- **Sterowanie PTZ (*Uplink*):** Zdarzenia płynące od użytkownika do serwera, sterujące silnikami kamery.
- **Powiadomienia o Alarmach (*Downlink*):** Zdarzenia płynące z serwera do użytkownika, informujące o wykryciu ruchu przez algorytm analizy obrazu.
- **Zarządzanie Stanem Nagrywania:** Synchronizacja interfejsu użytkownika (np. zmiana koloru diody nagrywania) ze stanem procesu rejestracji wideo na serwerze.

**3.2.3.3 Przykład implementacji: Sterowanie pozycją kamery** Najbardziej obrazowym przykładem zastosowania EDA w projekcie jest mechanizm sterowania ruchem kamery (*Pan/Tilt*). W tradycyjnym modelu HTTP, naciśnięcie przycisku musiałoby wysłać żądanie POST, a serwer musiałby odesłać odpowiedź, co przy szybkim klikaniu powodowałoby „zatykanie” się kolejki żądań. W modelu zdarzeniowym proces ten przebiega asynchronicznie:

1. **Produkcja (Klient):** Użytkownik naciska przycisk „W Lewo” w interfejsie przeglądarki. Warstwa Prezentacji (JavaScript) przechwytuje zdarzenie `mousedown` i natychmiast emituje zdarzenie WebSocket o nazwie `'move_camera'` z ładunkiem danych `{'direction': 'left', 'step': 2}`. Interfejs nie czeka na potwierdzenie, pozostając responsywnym.
2. **Transport:** Zdarzenie jest przesyłane otwartym kanałem TCP do serwera, bez narzutu nagłówków HTTP.
3. **Konsumpcja (Serwer):** Funkcja `handle_move_camera` w pliku `socket_handlers.py`, która nasłuchuje na ten konkretny typ zdarzenia (`@socketio.on`), zostaje wybudzona. Przekazuje ona komendę do sterownika sprzętowego `camera.py`, który wykonuje fizyczny obrót urządzenia.
4. **Sprzężenie zwrotne (Opcjonalne):** Jeśli kamera osiągnie fizyczny limit obrotu, warstwa logiki staje się nowym producentem zdarzenia. Emituje ona zdarzenie `'ptz_limit'`, na które nasłuchuje przeglądarka, aby zablokować odpowiedni przycisk w interfejsie graficznym.

Dzięki luźnemu powiązaniu komponentów (*loose coupling*), serwer może obsłużyć setki takich zdarzeń na sekundę, zapewniając płynne sterowanie „oko-ręka”, niemożliwe do osiągnięcia w architekturze synchronicznej.

### 3.3 Diagramy

#### Diagram komponentów

#### Diagram klas

#### Diagram komunikacji

### 3.4 Zastosowane narzędzia i technologie

Niniejszy rozdział stanowi dogłębną analizę techniczną **stosu technologicznego** (*technology stack*) dobrane do realizacji projektu inżynierskiego, którego celem jest stworzenie otwartego systemu obsługi kamer IoT, na przykładzie modelu TP-Link Tapo C200. Wybór poszczególnych komponentów nie był procesem arbitralnym, lecz wynikiem wieloaspektowej analizy wymagań funkcjonalnych i нефunkcjonalnych, ze szczególnym uwzględnieniem ograniczeń zasobowych urządzeń brzegowych (*Edge Computing*), konieczności minimalizacji opóźnień (*low-latency*) w przetwarzaniu strumienia audiowizualnego oraz imperatywu przełamania barier interoperacyjności narzuconych przez producenta (*vendor lock-in*).

W poniższych podrozdziałach dokonano dekonstrukcji architektury systemu na poziomie narzędziowym, omawiając zarówno warstwę językową, środowiskową, jak i biblioteki specyficzne dla domeny przetwarzania sygnałów. Każda decyzja projektowa została osadzona w kontekście aktualnego stanu wiedzy (*State of the Art*), odnosząc się do literatury przedmiotu w zakresie strumieniowania wideo, inżynierii wstecznej oraz optymalizacji systemów wbudowanych.

#### 3.4.1 Język Programowania

Wybór języka **Python** w wersji **3.13** jako fundamentu warstwy logicznej projektu stanowił decyzję strategiczną, wynikającą z analizy wymagań stawianych współczesnym systemom IoT oraz aplikacjom przetwarzającym multimedia. W kontekście inżynierii oprogramowania, dobór technologii musi uwzględniać wypadkową dostępnych narzędzi oraz skalowalności rozwiązania. Python, dzięki swojemu dojrzałemu ekosystemowi, pełni w projektowanym systemie rolę **warstwy orkiestracji** (ang. *glue code*).

#### Bogactwo ekosystemu bibliotecznego i interoperacyjność

Kluczowym argumentem przemawiającym za wyborem tego środowiska jest dostępność i stabilność zaawansowanych bibliotek dedykowanych przetwarzaniu sygnałów. W ekosystemie Pythona możliwe jest wykorzystanie gotowych, wysoce zoptymalizowanych *wrapperów* na biblioteki natywne. Moduły takie jak `threading` oraz `multiprocessing` pozwalają na efektywne zarządzanie operacjami wejścia/wyjścia (*I/O bound*), co jest krytyczne dla zachowania płynności strumieniowania w czasie rzeczywistym.

## Szybkie prototypowanie i paradygmat Rapid Application Development (RAD)

Specyfika pracy inżynierskiej wymaga narzędzi umożliwiających szybką iterację i weryfikację hipotez. Python, jako język dynamicznie typowany o wysokiej ekspresywności składni, drastycznie skraca cykl wytwórczy oprogramowania. W kontekście integracji z urządzeniami IoT, pozwala to na elastyczne dostosowywanie protokołów komunikacyjnych i logiki sterowania bez konieczności długotrwałej rekompilacji całego projektu.

### 3.4.2 Zarządzanie Zależnościami

W inżynierii oprogramowania systemów wbudowanych, **stabilność i powtarzalność środowiska** są kluczowe. Tradycyjne narzędzia zarządzania pakietami w Pythonie, takie jak `pip`, często zawodzą w złożonych scenariuszach CI/CD ze względu na wolny proces rozwiązywania zależności (*dependency resolution*) i brak determinizmu.

#### Nowoczesne Narzędzie: `uv`

W projekcie zastosowano `uv` – nowoczesny menedżer pakietów napisany w języku **Rust**. Narzędzie zostało wybrane ze względu na swoją bezkompromisową **wydajność**. Benchmarki wskazują, że `uv` potrafi instalować pakiety i rozwiązywać drzewa zależności od 10 do 100 razy szybciej niż standardowy `pip`. Skrócenie tego czasu znacząco przyspiesza cykl deweloperski (*feedback loop*) w kontekście budowania obrazów **Docker**.

#### Determinizm i Pliki Blokady

Kluczowym aspektem dla pracy inżynierskiej jest gwarancja, że system wdrożony na urządzeniu produkcyjnym będzie posiadał identyczne wersje bibliotek co środowisko deweloperskie. `uv` wprowadza obsługę uniwersalnych **plików blokady** (`uv.lock`), które precyzyjnie definiują całe drzewo zależności wraz z **sumami kontrolnymi** (*hashes*), gwarantując kryptograficzną spójność środowiska. Jest to mechanizm podnoszący standard inżynierski projektu, analogiczny do `Cargo.lock` w Rust.

#### Zaawansowana integracja z Dockerem

W projekcie wykorzystano specyficzne techniki optymalizacji współpracy `uv` z systemem plików Docker (*OverlayFS*). Zastosowanie mechanizmu montowania *cache'u* (*BuildKit cache mounts*) pozwala na **persystencję pobranych artefaktów** pomiędzy kolejnymi budowaniami kontenera. Dodatkowo, strategia **Bytecode Compilation** wspierana natywnie przez `uv` skraca czas startu aplikacji (*cold start*), co jest istotne w przypadku restartu usługi na urządzeniu monitoringu.

### 3.4.3 Ekosystem Konteneryzacji

Wdrożenie oprogramowania na **urządzeniach brzegowych** (*Edge Devices*) wiąże się z wyzwaniami heterogeniczności sprzętowej i konfliktów bibliotecznych. Zastosowanie technologii **Docker** w niniejszym projekcie nie jest jedynie wygodą, lecz koniecznością architektoniczną zapewniającą **izolację, przenośność i bezpieczeństwo**.

#### Izolacja Procesów i Bezpieczeństwo

Kamery IoT, w tym modele **Tapo**, operują w strefie podwyższonego ryzyka cybernetycznego (patrz: analiza podatności *CVE* w Rozdziale 2). Uruchomienie autorskiego serwera sterującego bezpośrednio na systemie operacyjnym hosta (*bare-metal*) niosłoby ryzyko, że ewentualne przejęcie kontroli nad aplikacją dałoby atakującemu dostęp do całego systemu. Docker zapewnia silną **izolację procesów** wykorzystując mechanizmy jądra **Linux** (*cgroups, namespaces*).

W projekcie zastosowano dodatkowo praktykę „non-root user” wewnątrz kontenera oraz **minimalizację uprawnień** (*capabilities drop*), co drastycznie redukuje powierzchnię ataku i chroni system hosta.

#### Multi-stage Builds i Optymalizacja Rozmiaru

Urządzenia klasy *embedded* często dysponują ograniczoną przestrzenią dyskową. Aby pogodzić wymagania posiadania ciężkich narzędzi kompilacji (np. GCC, *numpy*) z koniecznością lekkiego obrazu końcowego, zastosowano technikę **budowania wieloetapowego** (*Multi-stage Builds*).

1. **Stage 1 (Builder):** Obraz zawierający pełny *toolchain* (kompilatory GCC, nagłówki systemowe, *uv, git*).
2. **Stage 2 (Runtime):** Obraz typu „slim” (np. *python:3.13-slim-bookworm*), do którego kopiowane są jedynie wynikowe artefakty z etapu pierwszego.

Dzięki temu podejściu, finalny obraz kontenera jest pozbawiony zbędnych plików tymczasowych, *cache’u* i narzędzi deweloperskich, osiągając rozmiar rzędu **200-300 MB** zamiast ponad 1 GB, co przyspiesza jego dystrybucję i aktualizację.

### 3.4.4 Interfejs Webowy i Protokół Komunikacji

Efektywna interakcja użytkownika z systemem IoT wymaga **warstwy prezentacji**, która jest w stanie obsłużyć dynamiczny charakter **danych strumieniowych**. W tradycyjnym modelu webowym, opartym na **bezstanowym protokole HTTP** (*Request-Response*), realizacja płynnego sterowania w czasie rzeczywistym jest nieefektywna ze względu na narzut sieciowy (*overhead*).

## Serwer Aplikacyjny: Flask

Wybrano **Flask** – **lekki mikro-framework** w Pythonie (zgodny ze standardem *WSGI*). W przeciwieństwie do rozwiązań typu „full-stack”, Flask nie narzuca sztywnej struktury. Posiada minimalny narzut pamięciowy i pełni rolę „cienkiego klienta” serwerowego, odpowiedzialnego za:

- **Orkiestrację wątków:** Integracja asynchronicznych bibliotek sterujących kamerą.
- **Routing:** Obsługa statycznych plików interfejsu oraz końcówek API (*endpoints*).

## Protokół Transportowy: WebSockets

Zastosowano protokół **WebSocket** (*RFC 6455*) przy użyciu biblioteki `Flask-SocketIO`. Zapewnia on zestawienie **trwałego, dwukierunkowego kanału komunikacji** (pełny duplex) między przeglądarką klienta a serwerem, eliminując opóźnienia wynikające z cyklicznego odpytywania (*polling*).

Umożliwia to realizację dwóch celów:

- **Transmisja Wideo (*Low-Latency Streaming*):** Klatki wideo są przesyłane jako **binarne ładunki** (*binary payloads*) przez otwarty socket, co pozwala na redukcję opóźnień transmisji (*latency*).
- **Sterowanie Czasu Rzeczywistego (*Real-Time Control*):** Komendy sterujące **PTZ** (Pan/Tilt/Zoom) są przesyłane jako lekkie obiekty **JSON**. Czas reakcji kamery jest zminimalizowany ( $< 100ms$ ).

## Warstwa Klienta (Frontend): Vanilla HTML/JS

W warstwie interfejsu użytkownika podjęto świadomą decyzję o rezygnacji z rozbudowanych frameworków JavaScript (np. React, Vue) na rzecz **natywnych technologii webowych: Vanilla JavaScript**, HTML5 oraz CSS3. Zastosowanie **czystego JavaScriptu** pozwoliło na:

- **Maksymalną wydajność renderowania:** Bezpośrednia manipulacja **drzewem DOM** (*Document Object Model*) jest szybsza niż mechanizmy wirtualnego DOM.
- **Redukcję długu technologicznego:** Kod klienta nie wymaga procesu kompilacji (*transpilacji/bundlingu*).
- **Intuicyjną obsługę:** Natywne `EventListeners` służą do przechwytywania zdarzeń klawiatury (sterowanie kamerą za pomocą strzałek).

### 3.4.5 Biblioteki Przetwarzania Multimediiów

W projektowanym systemie nadzoru wizyjnego, kluczową rolę technologiczną odgrywa biblioteka **OpenCV** (*Open Source Computer Vision Library*). Stanowi ona rdzeń analityczny, odpowiadając za **akwizycję strumienia wideo** z kamer TP-Link Tapo oraz jego zaawansowaną **analizę w czasie rzeczywistym**. Biblioteka **PyAV** została wprowadzona jako rozwiązanie komplementarne, dedykowane wyłącznie do obsługi **ścieżki dźwiękowej**.

#### OpenCV jako główny silnik wideo i analityczny

Decyzja o uczynieniu OpenCV główną biblioteką projektu podyktowana była jej pozycją jako **standardu przemysłowego** oraz kompleksowością oferowanych rozwiązań. W ramach opracowanego oprogramowania, OpenCV realizuje pełen cykl życia danych wizyjnych:

- **Akwizycja Obrazu (*Video Acquisition*):** Wykorzystanie interfejsu `cv2.VideoCapture` pozwala na **stabilne nawiązanie połączenia** ze strumieniem **RTSP** kamery.
- **Przetwarzanie Macierzowe i Analityka:** Po pobraniu klatki, OpenCV wykonuje na niej operacje „inteligentne” (*Smart Features*). Zaimplementowany **algorytm detekcji ruchu** (oparty na odejmowaniu tła i filtracji Gaussa) oraz nanoszenie metadanych (*OSD*) są realizowane bezpośrednio na obiektach tej biblioteki.
- **Optymalizacja:** Dzięki **backendowi napisanemu w C++**, OpenCV zapewnia wysoką wydajność operacji na macierzach, co jest kluczowe przy przetwarzaniu obrazu o wysokiej rozdzielczości na urządzeniach o ograniczonej mocy obliczeniowej.

#### PyAV: Uzupełnienie luki funkcjonalnej (Audio)

Mimo wszechstronności w dziedzinie wideo, **OpenCV** posiada ograniczenia w zakresie **obsługi dźwięku** – biblioteka ta całkowicie ignoruje pakiety audio przesyłane w kontenerze RTSP.

W celu rozwiązania tego problemu inżynierskiego zastosowano bibliotekę **PyAV** (*binding dla FFmpeg*). Jej rola w projekcie jest ściśle zdefiniowana i ograniczona do: **Równoległego nawiązania połączenia, Ekstrakcji, dekodowania i transkodowania strumienia audio** (z formatów PCM/AAC), przy jednoczesnym ignorowaniu pakietów wideo w celu oszczędności zasobów CPU.

Taka architektura pozwala na wykorzystanie pełnej mocy OpenCV do analizy obrazu, delegując jedynie niezbędne minimum (obsługę mikrofonu) do wyspecjalizowanej biblioteki PyAV.

Tabela 3.2: Podział kompetencji w warstwie multimedialnej

Biblioteka	Status w projekcie	Odpowiedzialność
OpenCV	Główna ( <i>Core</i> )	Pobieranie wideo (RTSP), dekodowanie obrazu, detekcja ruchu, nanoszenie OSD, przygotowanie klatek do streamingu.
PyAV	Pomocnicza ( <i>Auxiliary</i> )	Przechwytywanie wyłącz- nie ścieżki audio, transko- dowanie dźwięku.

### 3.4.6 Kontrola Kamery i Inżynieria Wsteczna

Realizacja nadrzędnego celu pracy – **pełnego uniezależnienia systemu monitoringu od infrastruktury chmurowej producenta** – wymagała rozwiązania problemu **zamkniętej architektury** urządzenia. Kamera TP-Link Tapo C200 nie udostępnia publicznego **API** dla sieci lokalnej (*LAN*), co jest klasycznym przykładem strategii **Vendor Lock-in**.

Aby przełamać to ograniczenie, w warstwie sterowania wykorzystano bibliotekę **PyTapo**. Jest to rozwiązanie typu **Open Source**, stanowiące implementację klienta własnościowego protokołu komunikacyjnego TP-Link, powstałe w wyniku procesów **inżynierii wstecznej** (*Reverse Engineering*).

#### Mechanizm działania i emulacja klienta

Działanie biblioteki opiera się na **symulacji zachowania oficjalnej aplikacji mobilnej**. Analiza ruchu sieciowego wykazała, że kamera wykorzystuje zmodyfikowany protokół **HTTP** do przesyłania sterujących ładunków danych (*payloads*) w formacie **JSON**. Komunikacja ta jest zabezpieczona na kilku poziomach, które **PyTapo** skutecznie emuluje:

- **Negocjacja sesji (*Handshake*):** Biblioteka implementuje złożony proces **uwierzytelniania**, wymagający wymiany **kluczy sesyjnych** oraz **tokenów** (*stok*), generowanych w oparciu o algorytmy skrótu (MD5/SHA) i liczby losowe (*nonce*).
- **Szyfrowanie Payloadu:** W przeciwieństwie do otwartych standardów, parametry sterujące (np. koordynaty silnika PTZ) nie są przesyłane jawnym tekstem. PyTapo implementuje algorytmy **szyfrowania symetrycznego** (warianty AES), co pozwala na konstruowanie poprawnych, zaszyfrowanych zapytań.

## Przewaga nad standardem ONVIF

Wybór PyTapo był podyktowany ograniczeniami implementacyjnymi standardu **ONVIF** (*Open Network Video Interface Forum*), który w tanich kamerach Tapo ogranicza się często tylko do strumieniowania wideo (*RTSP*).

Zastosowanie PyTapo umożliwiło dostęp do „ukrytych” **funkcji administracyjnych**, niedostępnych przez generyczne sterowniki:

- **Pełna kontrola PTZ** (*Pan-Tilt-Zoom*): Precyzyjne sterowanie silnikami krokowymi kamery.
- **Zarządzanie sensorem**: Programowe przełączanie trybu nocnego (**kontrola filtra IR-Cut**) oraz regulacja czułości detekcji ruchu.
- **Funkcje prywatności**: Możliwość zdalnego **wygaszenia obiektywu** (*Privacy Mode*) lub wyłączenia diody statusu LED.
- **Formatowanie nośników**: Zdalne zarządzanie kartą SD.

### 3.4.7 Narzędzie do Kompozycji i Zapisu Danych

Ostatnim ogniwem w łańcuchu przetwarzania danych multimedialnych jest moduł odpowiedzialny za **trwały zapis** (*persystencję*) materiału dowodowego. Ze względu na przyjętą **architekturę hybrydową**, w której obraz i dźwięk przetwarzane są przez niezależne biblioteki (**OpenCV** i **PyAV**), zaistniała konieczność zastosowania narzędzia efektywnie integrującego te dwa rozłączne strumienie. Do realizacji tego zadania wybrano bibliotekę **MoviePy**.

#### Rola integratora strumieni (Multipleksing)

MoviePy pełni w projekcie funkcję **orkiestratora procesu zapisu**. Jego zadaniem jest przeprowadzenie **multipleksowania** (*muxing*), czyli scalenia danych wizyjnych (**macierzy NumPy** z OpenCV) i danych fonicznych (**próbek audio** z PyAV) zgromadzonych w buforach pamięci. Wynikiem jest **enkapsulacja** do standardowego **kontenera multimedialnego** (MP4) z kodekami **H.264** i **AAC**. Wybór dedykowanej biblioteki gwarantuje zachowanie **spójności struktury pliku wynikowego**.

#### Abstrakcja nad FFmpeg i Synchronizacja A/V

MoviePy to wysokopoziomowa nakładka (*wrapper*) na oprogramowanie **FFmpeg**. Zastosowanie jej eliminuje złożoność bezpośredniego wywoływania komend FFmpeg i zapewnia automatyczną **synchronizację A/V** (*lip-sync*), zarządzając osią czasu i dopasowując długość ścieżki audio do sekwencji wideo. To kluczowe w przypadku **de-**



**tekcji ruchu**, gdzie nagrania mają zmienną długość. Proces kodowania odbywa się w sposób **wsadowy** (*batch processing*) w momencie zakończenia nagrania.

## Rozszerzalność (Extensibility)

Biblioteka ta oferuje bogaty zestaw funkcji do **nieliniowego montażu wideo** (*NLE*) z poziomym kodu, co ułatwia przyszły rozwój oprogramowania i implementację dodatkowych funkcjonalności, takich jak:

- **Dynamiczne dodawanie znaków wodnych** (*Watermarking*).
- **Łączenie (konkatenacja)** wielu klipów zdarzeń w jeden raport wideo.
- **Nakładanie napisów końcowych** z parametrami zdarzenia (data, typ wykrytego obiektu).

## 3.5 Proces implementacji rozwiązania

Niniejszy podrozdział stanowi techniczną dokumentację procesu transformacji koncepcji architektonicznej, zdefiniowanej w sekcji 3.4, w w pełni funkcjonalny **artefakt programistyczny**. Celem poniższego opisu jest przedstawienie ewolucyjnego **cyklu wytwarzania oprogramowania** (*SDLC*), który doprowadził do powstania systemu integrującego zamknięty ekosystem kamer TP-Link Tapo z otwartym środowiskiem *Open Source*.

Proces implementacji został podzielony na etapy odzwierciedlające **warstwową strukturę** projektowanego systemu, począwszy od najniższej warstwy sprzętowej (*Hardware Abstraction Layer*), poprzez **logikę biznesową** i warstwę komunikacyjną (*Middleware*), aż po **warstwę prezentacji** (*Frontend*). Takie podejście pozwoliło na empiryczną weryfikację założeń o trójwarstwowej i potokowej architekturze rozwiązania, zapewniając jednocześnie izolację poszczególnych modułów i łatwość ich późniejszego testowania.

W kolejnych punktach przedstawiono szczegółowo sposób rozwiązania kluczowych **wyzwań inżynierskich**, takich jak:

- nawiązanie stabilnego połączenia z urządzeniem **IoT** operującym na **własnościowych protokołach**,
- implementacja **asynchronicznego przetwarzania** strumieni multimedialnych (audio/wideo) w czasie rzeczywistym przy użyciu bibliotek **OpenCV** i **FFmpeg**,
- opracowanie **algorytmów detekcji ruchu** działających na brzegu sieci (*edge processing*),
- **konteneryzacja aplikacji** z wykorzystaniem środowiska **Docker**, zapewniająca jej przenośność i powtarzalność wdrożenia.

Opisany poniżej proces stanowi syntezę doboru odpowiednich narzędzi (język Python, framework Flask, protokół WebSocket) oraz metodologii inżynierskiej, prowadzącą do uzyskania gotowego narzędzia nadzoru wizyjnego, niezależnego od chmury producenta.

### 3.5.1 Provisioning i pierwotna konfiguracja środowiska kamery

Proces implementacji rozwiązania rozpoczęto od fizycznego uruchomienia urządzenia oraz przeprowadzenia procedury **provisioningu** (*wstępnej konfiguracji*), mającej na celu włączenie kamery do lokalnej infrastruktury sieciowej i odblokowanie interfejsów komunikacyjnych niezbędnych dla tworzonego oprogramowania.

#### Stan faktyczny i ograniczenia fabryczne

Analiza fabrycznie nowego urządzenia TP-Link Tapo C200 wykazała, że funkcjonuje ono w modelu tzw. „zamkniętego ogrodu” (*walled garden*). Domyślna konfiguracja *firmware'u* jest nastawiona wyłącznie na komunikację z chmurą producenta. W stanie „po wyjęciu z pudełka” (*OBE – Out-of-box experience*) kamera charakteryzuje się następującymi **ograniczeniami**:

- Brak otwartego dostępu do strumieniowania protokołem **RTSP** (*Real Time Streaming Protocol*).
- Zablokowane porty odpowiedzialne za standard **ONVIF**.
- Brak zdefiniowanego lokalnego użytkownika administracyjnego, co uniemożliwia autoryzację zewnętrznych skryptów sterujących.

#### Procedura konfiguracji i odblokowania dostępu

W celu przystosowania urządzenia do współpracy z autorskim rozwiązaniem *Open Source*, przeprowadzono następującą **sekwencję działań konfiguracyjnych**:

1. **Inicjalizacja sieciowa:** Wykorzystując oficjalną aplikację mobilną Tapo, nawiązano tymczasowe połączenie i przekazano poświadczenia docelowej sieci Wi-Fi.
2. **Utworzenie „Konta Kamery” (*Camera Account*):** W ustawieniach zaawansowanych aplikacji mobilnej zdefiniowano dedykowane konto lokalne. Jest to krok **krytyczny**, ponieważ utworzone w ten sposób login i hasło (TAPO\_USERNAME i TAPO\_PASSWORD) są przechowywane w pamięci urządzenia i służą do późniejszej autoryzacji zapytań **RTSP** oraz **HTTP** wysyłanych przez aplikację.
3. **Aktywacja interfejsów otwartych:** Wymuszono tryb zgodności z oprogramowaniem zewnętrznym poprzez włączenie opcji obsługi „Third-party software” /

**ONVIF**. Operacja ta skutkowała otwarciem portu **554** (dla strumienia wideo RTSP) oraz portu **2020** (dla protokołu ONVIF).

> **Uwaga inżynierska:** Aby zapewnić stabilność połączenia dla serwera Python, na routerze sieci lokalnej wykonano **rezerwację adresu IP** (*Static DHCP Lease*) dla adresu MAC kamery.

Przeprowadzenie powyższych kroków pozwoliło na transformację urządzenia z pasywnego klienta chmury w **aktywny węzeł sieciowy**, gotowy do przyjmowania poleceń i udostępniania mediów poprzez standardowe protokoły sieciowe.

### 3.5.2 Konfiguracja Środowiska Programistycznego

Przygotowanie środowiska deweloperskiego rozpoczęto od inicjalizacji projektu przy użyciu narzędzia `uv`. Proces ten przebiegał w dwóch głównych fazach: utworzenia podstawowej struktury (*scaffolding*) oraz instalacji i zablokowania zależności bibliotecznych.

#### Inicjalizacja i zarządzanie zależnościami

W pierwszej kolejności, w pustym katalogu roboczym, wykonano polecenie `uv init`. Operacja ta wygenerowała podstawowe pliki konfiguracyjne, w tym plik `.python-version`, w którym sztywno zdefiniowano wersję interpretera na **Python 3.13**.

Następnie przystąpiono do instalacji wymaganych bibliotek zewnętrznych za pomocą polecenia `uv add`:

- `flask` oraz `flask-socketio` – obsługa serwera HTTP i komunikacji **WebSocket**.
- `opencv-python-headless` – przetwarzanie obrazu (wersja zoptymalizowana dla serwerów).
- `pytapo` – biblioteka kliencka do komunikacji z kamerą.
- `moviepy` oraz `numpy` – operacje na plikach wideo i macierzach danych.

Efektem tego procesu było wygenerowanie plików `pyproject.toml` (deklaracja zależności) oraz `uv.lock` (drzewo zależności ze **skrótami kryptograficznymi**), co zamknęło etap konfiguracji środowiska uruchomieniowego.

#### Struktura projektu

Na bazie zainicjowanego środowiska utworzono docelową strukturę katalogów i plików, która dzieli aplikację na **logiczne moduły funkcjonalne**. Architektura plików w projekcie prezentuje się następująco:

- **Katalog główny (*Root*):**

- `run.py`: Główny punkt wejścia (*entry point*) uruchamiający serwer.
- `config.json`: Zewnętrzny plik konfiguracyjny (adres IP kamery, poświadczenia).
- `Dockerfile` oraz `uv.lock/pyproject.toml`.

- **Moduł aplikacji (`app/`):**

- `init.py`: Fabryka aplikacji **Flask**, inicjująca instancję serwera.
- `settings.py` oraz `shared.py`: Moduły do ładowania konfiguracji i **współdzielenia obiektów** (np. instancji kamery) między wątkami.
- `camera/`: **Warstwa abstrakcji sprzętowej** (*HAL*) obsługująca bezpośrednie połączenie z urządzeniem.
- `video/`, `audio/`, `detection/`, `recording/`: Logika biznesowa i przetwarzanie mediów/analiza obrazu.
- `web/`: **Warstwa prezentacji** zawierająca trasy (*routes*), obsługę zdarzeń WebSocket oraz szablony HTML (*templates*).

Tak przygotowana struktura zapewniła separację **logiki biznesowej** od warstwy sprzętowej i interfejsu użytkownika, co było niezbędne do dalszej implementacji poszczególnych funkcjonalności

### 3.5.3 Implementacja serwera HTTP

Centralnym punktem logicznym systemu, spajającym warstwę prezentacji z logiką backendową, jest serwer HTTP zrealizowany w oparciu o mikroframework **Flask**. Wybór tego rozwiązania podyktowany był koniecznością zachowania **niskiego narzutu pamięciowego** (*low footprint*) przy jednoczesnej elastyczności w obsłudze protokołów asynchronicznych.

#### Rozwiązanie problemu „Double Execution” i limity sprzętowe

Krytycznym wyzwaniem na etapie implementacji serwera było dostosowanie jego cyklu życia do **ograniczeń sprzętowych** kamery TP-Link Tapo C200, która limituje liczbę jednoczesnych sesji **RTSP** (zazwyczaj do dwóch).

Standardowy tryb deweloperski frameworka Flask wykorzystuje mechanizm *reloader*, który monitoruje zmiany w kodzie i powoduje uruchomienie **dwóch procesów systemowych**: procesu monitorującego oraz procesu roboczego. W kontekście aplikacji IoT skutkowało to zjawiskiem „podwójnego uruchomienia” (*double execution*) wątków strumieniujących. Efektem było natychmiastowe **wyczerpanie puli dostępnych połączeń** kamery (*Resource Exhaustion*) i odrzucanie prób autoryzacji.

W celu wyeliminowania tego błędu, w głównym punkcie wejścia aplikacji (`run.py`) wymuszono konfigurację serwera z flagą:

```
1 if __name__ == '__main__':  
2     socketio.run(app, debug=True, use_reloader=False)
```

Listing 3.1: Wymuszenie jednokrotnego uruchomienia serwera Flask

Ustawienie parametru `use_reloader=False` zapewniło **deterministyczne, jednokrotne uruchomienie** wątków `VideoStreamer` i `AudioStreamer`, gwarantując stabilność połączenia z kamerą przy zachowaniu pełnej kontroli nad zasobami sieciowymi.

### 3.5.4 Implementacja Warstwy Dostępu do Sprzętu (HAL)

**Warstwa Dostępu do Sprzętu** (ang. *Hardware Abstraction Layer* – *HAL*) stanowi fundamentalny element architektury systemu, izolujący wysokopoziomą logikę biznesową od specyfiki protokołów komunikacyjnych urządzenia końcowego. W projekcie funkcję tę pełni moduł `app/camera`, którego centralnym komponentem jest klasa `TapoCamera`.

Klasa ta realizuje **wzorzec fasady**, ukrywając złożoność obsługi strumienia **RTSP** oraz administracyjnego **API HTTP/ONVIF**. Dzięki takiemu podejściu, pozostałe moduły systemu (np. detektor ruchu czy interfejs webowy) operują na jednolitym interfejsie obiektowym, nie wymagając znajomości niskopoziomowych detali implementacyjnych.

### Inicjalizacja połączenia i akwizycja wideo

Nawiązanie komunikacji z kamerą odbywa się dwutorowo. Metoda `connect()` (Listing 3.1) inicjuje niezależne sesje dla **podsystemu wideo** oraz **podsystemu sterowania**.

Do akwizycji obrazu wykorzystano bibliotekę **OpenCV** (klasa `cv2.VideoCapture`). Adres strumienia budowany jest dynamicznie w oparciu o poświadczenia, zgodnie ze schematem `rtsp://user:password@ip/stream1`. Równolegle inicjowana jest instancja obiektu `Tapo` z biblioteki pomocniczej **PyTapo**, która autoryzuje się w panelu administracyjnym kamery, umożliwiając wydawanie poleceń sterujących (*PTZ*).

**Listing 3.1. Implementacja inicjalizacji połączenia hybrydowego (RTSP + HTTP) w klasie `TapoCamera`.**

```
1 class TapoCamera:  
2     def __init__(self, ip, user, password, cloudPassword):  
3         self.ip = ip  
4         self.user = user  
5         self.password = password  
6         self.cloudPassword = cloudPassword  
7         self.cap = None
```

```

8         self.admin = None
9
10    def connect(self):
11        # 1. Setup Video (RTSP)
12        url = f"rtsp://{self.user}:{self.password}@{self.ip}/stream1"
13        self.cap = cv2.VideoCapture(url)
14        video_success = self.cap.isOpened()
15
16        # 2. Setup Controls (Pytapo)
17        try:
18            self.admin = Tapo(self.ip, self.user, self.cloudPassword)
19            # Weryfikacja połączenia poprzez proste zapytanie
20            self.admin.getBasicInfo()
21            control_success = True
22        except Exception as e:
23            print(f"Control Error: {e}")
24            control_success = False
25
26        if video_success and control_success:
27            print(f"Fully Connected to {self.ip}")
28            return True
29        else:
30            return False

```

Źródło: Opracowanie własne.

## Abstrakcja sterowania mechaniką (PTZ)

Istotnym wyzwaniem inżynierskim była implementacja sterowania silnikami *Pan-Tilt-Zoom* (PTZ) w sposób bezpieczny dla mechaniki urządzenia. Bezpośrednie wywoływanie metod biblioteki PyTapo obarczone jest ryzykiem przekroczenia **fizycznego zakresu obrotu** głowicy.

W celu rozwiązania tego problemu zaimplementowano metodę `move()`, która pełni rolę tłumacza (*wrapper*). Konwertuje ona semantyczne polecenia kierunkowe (np. „up”, „left”) na wektory przesunięcia silników. Metoda ta zawiera również **mechanizm obsługi wyjątków** (*Exception Handling*), który przechwytuje informacje o osiągnięciu limitu obrotu („limit\_reached”) i zwraca je w ustrukturyzowanej formie do klienta API, zamiast przerywać działanie aplikacji.

### Listing 3.2. Implementacja metody sterującej z obsługą błędów granicznych.

```
1  def move(self, direction, step=5):
2      """Wykonuje ruch kamera. Zwraca słownik ze statusem
3          operacji."""
4      if not self.admin:
5          return {"success": False, "error": "not_connected"}
6
7      step = int(step)
8
9      try:
10         if direction == "up":
11             self.admin.moveMotor(0, step)
12         elif direction == "down":
13             self.admin.moveMotor(0, -step)
14         elif direction == "left":
15             self.admin.moveMotor(-step, 0)
16         elif direction == "right":
17             self.admin.moveMotor(step, 0)
18
19         return {"success": True}
20
21     except Exception as e:
22         error_msg = str(e).lower()
23         # Sprawdzenie słów kluczowych oznaczających
24         koniec zakresu ruchu
25         if "range" in error_msg or "limit" in error_msg:
26             print(f" Limit Reached: {direction}")
27             return {"success": False, "error": "limit_reached"}
28
29         try:
30             self.admin = Tapo(self.ip, self.user, self.password)
31         except:
32             pass
33         return {"success": False, "error": "unknown"}
```

Źródło: Opracowanie własne.

## Obsługa Audio i Separacja Logiczna

Warto zauważyć, że w przedstawionej klasie `TapoCamera` **celowo pominięto obsługę strumienia audio**. Zgodnie z przyjętą strategią niezawodności (*Fault Tolerance*), obsługa dźwięku została wydzielona do **niezależnego obiektu** korzystającego z biblioteki **PyAV**. Taka separacja gwarantuje, że ewentualne opóźnienia w dekodowaniu dźwięku lub błędy formatu audio nie wpływają na płynność strumienia wideo i responsywność sterowania PTZ.

### 3.6 Podsumowanie



## 4 Testowanie i Analiza wyników

### 4.1 Zakres Testów

Opis metodologii testowania funkcjonalności PTZ, streamingu, detekcji ruchu i nagrywania.

### 4.2 Środowisko Testowe

Specyfikacja sprzętu (np. Raspberry Pi lub PC hostujący Docker), wersji oprogramowania i konfiguracji sieciowej.

### 4.3 Wyniki testów i Analiza

Prezentacja kluczowych metryk, w tym:

- **Opóźnienie (Latency):** Porównanie opóźnień strumienia RTSP przetworzonego przez Flask/WebSockets z oficjalną aplikacją.
- **Skuteczność Detekcji Ruchu:** Wyniki testów algorytmu OpenCV (np. metryki True Positive Rate, False Positive Rate).
- **Wydajność Systemu:** Obciążenie procesora hosta w warunkach ciągłego streamingu i detekcji.

### 4.4 Podsumowanie

## Wnioski Końcowe

Niniejsza praca inżynierska zrealizowała postawiony cel główny, tworząc kompletne i modułowe rozwiązanie Open Source dla kamer TP-Link Tapo C200. Projekt udowodnił, że bariera **vendor lock-in** może być skutecznie przełamana za pomocą inżynierii wstecznej protokołów własnościowych i integracji sprawdzonych narzędzi (Docker, FFmpeg, OpenCV).

Osiągnięte wyniki potwierdzają:

1. Pełną i stabilną kontrolę PTZ z poziomu autorskiego interfejsu.
2. Możliwość lokalnego streamingu wideo z minimalnymi opóźnieniami.
3. Skuteczną implementację autorskiej detekcji ruchu, oferującej większą konfigurowalność niż wbudowane funkcje kamery.

## Kierunki dalszego rozwoju

Dalsze prace mogą koncentrować się na:

- Integracji z protokołami Smart Home (np. MQTT) w celu komunikacji z systemami takimi jak Home Assistant.
- Zastosowaniu WebRTC (Web Real-Time Communication) w celu dalszej redukcji opóźnień streamingu.
- Wdrożeniu lekkich modeli uczenia maszynowego (np. YOLOv5 Nano) dla zaawansowanej detekcji obiektów, wykorzystujących akcelerację sprzętową GPU hosta.

## Podsumowanie pracy

Opracowany prototyp stanowi w pełni funkcjonalną i otwartą alternatywę dla zamkniętego ekosystemu producenta, oferując użytkownikowi pełną suwerenność nad gromadzonymi danymi i możliwością rozbudowy systemu.

## Bibliografia

- Abdalla, P. (2020). *Testing IoT Security: The Case Study of an IP Camera*. ResearchGate PDF. URL: [https://www.researchgate.net/publication/342184780\\_Testing\\_IoT\\_Security\\_The\\_Case\\_Study\\_of\\_an\\_IP\\_Camera](https://www.researchgate.net/publication/342184780_Testing_IoT_Security_The_Case_Study_of_an_IP_Camera).
- Alaba, F.A. i in. (2017). „Internet of Things security: A survey”. W: *Future Generation Computer Systems* 79, s. 273–283. DOI: 10.1016/j.future.2017.03.005. URL: <https://pmc.ncbi.nlm.nih.gov/articles/PMC7506579/>.
- Bou-Harb, Elias, José Guevara, Waleed Alasmary i in. (2024). „Unmasking vulnerabilities by a pioneering approach to Smart IoT cameras”. W: *Alexandria Engineering Journal* 69, s. 335–349. DOI: 10.1016/j.aej.2024.02.004. URL: <https://www.sciencedirect.com/science/article/pii/S1110866524000768>.
- Fossum, Eric R. (1997). „CMOS image sensors: electronic camera-on-a-chip”. W: *IEEE Transactions on Electron Devices* 44.10, s. 1689–1698. DOI: 10.1109/16.628824.
- Howe, Dennis i H. S. El-Ghoroury (2014). *Digital Image Processing: An Algorithmic Introduction using Java*. Springer. ISBN: 978-1447166832.
- Minerva, Roberto, Abyi Biru i Domenico Rotondi (2021). „A survey on IoT platforms: Communication, security, and more”. W: *Computer Networks* 193, s. 108040. DOI: 10.1016/j.comnet.2021.108040. URL: <https://www.sciencedirect.com/science/article/pii/S1389128621001444>.

;

## **Spis rysunków**

3.1	Schemat architektury rozwiązania (Klient - Middleware - Sprzęt) . . . . .	62
-----	---	----

## Spis tabel

1.1	Główne obszary zastosowań kamer IP (na podstawie raportu Hanwha Vision, 2025) . . . . .	7
1.2	Porównanie protokołów TCP i UDP w kontekście kamery IP. . . . .	17
1.3	Etapy przetwarzania w potoku ISP. . . . .	22
1.4	Podstawowe komendy protokołu RTSP. . . . .	26
2.1	Kluczowe Specyfikacje Techniczne TP-Link Tapo C200 . . . . .	50
2.2	Analiza Protokołów Komunikacyjnych Tapo C200 pod kątem Integracji Open-Source . . . . .	53
3.1	Podział warstw architektury systemu IoT . . . . .	63
3.2	Podział kompetencji w warstwie multimedialnej . . . . .	76