Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Kanał optyczny transmisji danych dla urządzeń przenośnych maksymalizacja przepustowości

Arkadiusz Lewandowski
Nr indeksu: 208836

Praca inżynierska napisana
pod kierunkiem
Dr. inż. Łukasza Krzywieckiego

Politechnika
Wrocławska
Wrocław 2016

Spis treści

Wstęp **3**

1 Analiza problemu **4**
1.1 Kod QR 4
1.2 Parametry urządzeń 7
1.2.1 Odbieranie 8
1.2.2 Nadawanie 8

2 Projekt systemu **9**
2.1 Diagramy aktywności 9
2.2 Diagramy klas 11
2.3 Diagramy sekwencji 15
2.4 Diagramy stanów 17

3 Implementacja systemu **19**
3.1 Opis technologii 19
3.2 Omówienie kodów źródłowych 20

4 Testy **24**
4.1 Przepustowości nadawania 24
4.2 Badanie przepustowości odbioru 28
4.3 Badanie przepustowości w komunikacji 33
4.4 Dyskusja 38

5 Podsumowanie **39**

Bibliografia **40**

A Zawartość płyty CD **41**

Wstęp

Praca swoim zakresem obejmuje zagadnienia bezprzewodowej wymiany danych pomiędzy urządzeniami przenośnymi bez wykorzystania infrastruktury sieciowej. Poprzez infrastrukturę sieciową rozumie się tutaj infrastrukturę sieci bezprzewodowych realizowanych za pomocą standardu IEEE 802.11*, w ramach których urządzenia wymieniające się danymi muszą być podłączone do tej samej sieci. Brak wykorzystania infrastruktury zakłada, że urządzenia łączą się ze sobą w ramach sieci „ad-hoc" w kanałach elektromagnetycznych (Bluetooth, IrDA, NFC) lub kanałach alternatywnych- dźwiękowych za pomocą modulacji sygnału bądź kanału optycznego. Z racji tego, że współczesne smartfony posiadają kamery i urządzenia rejestrujące, w pracy tej rozważa się kanał optyczny. Celem pracy będzie zbadanie efektywności przesyłania danych za pomocą kanału optycznego, ergonomia nawiązywania połączenia oraz przepustowość transmisji (teoretyczna przepustowość wynikająca z teoretycznych ograniczeń oraz z ograniczeń wynikających z programistycznych implementacji).

Szczegółowymi celami pracy są:

- oszacowanie maksymalnej przepustowości transmisji danych za pomocą kodów QR,
- projekt systemu do przesyłania danych w kanale optycznym z wykorzystaniem technologii QR kodów jako nośnika zakodowywanych bitów po stronie nadawczej i odbiorczej,
- implementacja projektu na platformie iOS z wykorzystaniem bibliotek standardowych lub łatwo dostępnych na wszystkich platformach marki Apple,
- przeprowadzenie testów weryfikujących przepustowość uzyskaną z przepływności, która została wyliczona.

Istnieje szereg aplikacji wykorzystujących kody QR do potwierdzania tożsamości obiektów, przy czym ich przeznaczenie zakłada statyczność kodów oraz brak interakcji z urządzeniem czytającym. Natomiast aplikacji wykorzystujących kody te do wymiany danych lub do szybkiego ich czytania nie ma wielu. Warto zatem przyjrzeć się scenariuszom, w których mogłyby ułatwiać codzienne czynności. Obiekt indentyfikujący się takim kodem może być *statyczny*, w rozumieniu, że nie wymaga on mocy obliczeniowej. Raz wygenerowana sekwencja QR kodów może przedstawiać zbiór danych, co można wykorzystać jako tworzenie fizycznych kopii zapasowych dokumentów, które mogłyby w prosty sposób być czytane za pomocą urządzenia elektronicznego. Takim rozwiązaniem można zapobiec utracie cyfrowych danych przez czynniki zewnętrzne oraz uszkodzenia złośliwym oprogramowaniem, drukując wygenerowane dane i tym samym oddzielić informacje od sieci, wciąż mając do niej łatwy dostęp. Urzędy przechowujące dane o obywatelach mogą wydrukować wszystkie potrzebne informacje o nich i przechowywać bezpiecznie, zachowując przy tym łatwość ich ponownego

wczytania. Zapewniając sobie tym samym bezpieczeństwo danych i brak ingerencji ze strony cyfrowej. Przechowywanie i wczytywanie faktur mogłoby stać się o wiele szybsze i bezpieczniejsze. Użytkownik nie musiałby jej pobierać, a jedynie nakierować urządzenie czytające na monitor lub fizyczny wydruk i miałby dostęp do swoich danych. W tym przypadku naszą uwagę może zwrócić zagrożenie często omawiane przy temacie kodów QR i ich czytania. Problemem może się okazać złośliwy kod wstrzyknięty w naszą sekwencję QR, jednakże nie uruchamiając czytanych symboli, a jedynie wyświetlając je odkodowane, blokujemy jedną z dwóch dróg ataku(zobacz [5]). Druga natomiast, często nazywana przepełnieniem bufora, jest wyeliminowana przez możliwość techniczną urządzenia. Dla kodu QR pojedynczy symbol nie przekracza 3000 bajtów, mieści się zatem w zwykłej zmiennej typu *String* ([4], [2]).

Brak możliwości wymiany danych za pomocą kabli lub komunikacji radiowej. W takim przypadku użytkownik mógłby wygenerować na swoim urządzeniu ciąg danych, które w sposób jednoznaczny i dekodowalny przedstawiałyby plik lub ciąg znaków, do odczytania przez inne urządzenie. Użytkownik minimalizuje ryzyko wycieku danych w sieci oraz uniezależnia się od platformy, którą miałby te dane przesłać. Weźmy przykład, w którym użytkownik nie ma dostępu do sieci oraz jego urządzenia nie są w stanie połączyć się za pomocą takich technologii jak bluetooth, WiFi lub nawet USB oraz platformy nie przewidują komunikacji z innymi urządzeniami spoza swojej marki. W takim przypadku komunikacja za pomocą standardowych QR kodów może zostać alternatywnym kanałem transmisji danych. Sytuacja ta pokazuje także, że użytkownicy tej techniki są niezależni od infrastruktury takich cyfrowych technologii jak telefoniczna sieć komórkowa.

Praca ta ma za zadanie pokazać, że efektywna transmisja za pomocą kodów QR jest możliwa oraz nie wymaga dodatkowej infrastruktury, co umożliwia wykorzystanie w warunkach polowych bez uprzedniego przygotowania.

Dokument pracy składa się z następujących rozdziałów:

- W rozdziale pierwszym zostaje omówiony problem i jego teoretyczne rozwiązanie. Następnie w jego podrozdziałach zostają omówione założenia teoretyczne samej transmisji danych.

- W drugim rozdziale przedstawiony jest projekt prototypu aplikacji, która spełnia wymagania teoretycznego rozwiązania problemu.

- W trzecim rozdziale jest budowa aplikacji od podstaw wraz z oceną możliwości programistycznych, takich jak obsługa wątkowości dla pojedynczego urządzenia wejścia - kamery, a także ograniczenia wynikające ze wspólnego czytania/nadpisywania pamięci. Zostają także omówione biblioteki i wzorce projektowe użyte/odrzucone w aplikacji. Czytelnik zostaje wprowadzony w świat optymalizacji aplikacji mobilnej pod względem zmniejszenia wykorzystania pamięci podręcznej.

- W czwartym zostają przeprowadzone badania na różnych urzadzeniach mobilnych, platformy iOS. Uwzględniając szybkości procesorów i możliwości interakcji z kartą graficzną przez bibliotekę AVFoundation. Zostają także zaprezentowane i omówione jednokierunkowe transfery: osobno z punktu widzenia nadającego i odbierającego urządzenia, a

także perspektywa komunikacji między urządzeniami za pomocą kamer niższej rozdzielczości. W tym rozdziale pojawiają się też wykresy osiągniętych zakresów dla poszczególnych urządzeń i ich specyfikacji, takich jak rozdzielczość, częstotliwość odświeżania ekranu oraz nagrywania.

- Piąty rozdział stanowi podsumowywanie badań i następuje próba oceny możliwości urządzeń w niedalekiej przyszłości oraz przypadki użycia alternatywnego kanału w komunikacji bez interferencji.

1 Analiza problemu

W tym rozdziale przedstawiana jest analiza problemu transmisji danych za pomocą QR kodów. Szczegółowa dokumentację QR kodów można znaleźć w publikacji nr. [6]. Omówiona zostaje konstrukcja kodu QR następnie sposoby i rozważania nad jego użyciem. Przeanalizowanie wymogów jakie musiałaby spełniać aplikacja umożliwiająca zbadanie parametrów maksymalizujących przepustowość kanału. W celu maksymalizacji przepustowości w pracy analizuje się różne czynniki optymalizacji wymiany danych. W przypadku transmisji takim kanałem alternatywnym jak kanał optyczny, szczególnymi czynnikami są przede wszystkim możliwości ustalonego nośnika, algorytmów używanych w jego eksploatacji oraz parametrów technicznych urządzeń, na których jest on wykorzystywany. W tym badaniu wybranym nośnikiem jest kod QR.

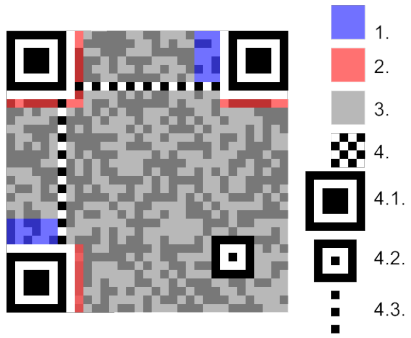
1.1 Kod QR

W czerwcu 2000 roku organizacja ISO włączyła kod QR jako standard ISO/IEC18004. W porównaniu z konwencjonalnym kodem kreskowym, kod QR jest w stanie zawrzeć w sobie od kilkudziesięciu do kilkuset razy więcej informacji. Przez swoją budowę i sposób kodowania - można w nim przechować dane w dowolnej postaci cyfrowej. Główną zaletą jest to, że pojedynczy kod QR jest w stanie reprezentować sobą 7089 cyfr albo 4296 alfranumerycznych symboli, albo 2953 bajtów. Kod QR zawiera odpowiednią nadmiarowość wykorzystywaną w korekcji błędów. Zatem dane zatracone w wyniku częściowego przysłonięcia lub zniszczenia - mogą zostać odzyskane. Maksymalnie każdy pojedynczy QR jest w stanie zakodować 30% danych korekcyjnych ze wszystkich bajtów sobą reprezentowanych w postaci słów kodowych (zobacz [7]).

W pojedynczym kodzie modułami, z których się on składa, nazywamy kwadraty przybierające jeden z dwóch kolorów: biały lub czarny. Zbiory modułów na kodzie tworzą słowa kodowe, które przedstawiają informacje jako poszczególne znaki. Wygląd pojedynczego kodu jest zależny od jego wersji. Odróżniamy je na podstawie poziomu korekcji błędów oraz ilości danych w nich zapisanych. W odczytywaniu kodów QR wykorzystuje się wzór wyszukiwania umożliwiający czytnikowi odnalezienie poszczególnych miejsc w QR kodzie, względem których odczytywane są pozostałe jego części. W pierwszej kolejności wyszukiwane są duże moduły (pole oznaczone liczbą 4.1 na rysunku 1.1), które muszą być oddzielone od krawędzi całego kodu białą ramką szerokości co najmniej jednego modułu. Kolejnym szukanym wzorcem jest (pole oznaczone liczbą 4.3 na rysunku 1.1) wzór synchronizacji, który pozwala na określenie wersji, gęstości oraz współrzędnych poszczególnych danych zapisanych w kodzie. Kolejnym szukanym elementem jest lub są tzw. wyrównania (pole oznaczone liczbą 4.2 na rysunku 1.1). Może ich być wiele w zależności od ilości danych zapisanych w QR. Na podsta-

wie (pole oznaczone liczbą 2. na rysunku 1.1) można ustalić wielkość kodu, a ilość danych w obszarze (pole oznaczone liczbą1. na rysunku 1.1) wskazuje na wersję QR kodu, gdyż jest ona zależna od maskowania i ilości danych. (Pola oznaczone liczbą 3. na rysunku 1.1) prezentują dane i korekcję błędów dla tych danych. Kody QR są w pewnym sensie odporne na zniszczenia i przesłonięcia. Zawdzięczają to korekcji błędów, która duplikuje dane w nich zawarte. Dotyczy to jednak szczególnych miejsc. Słowo kodowe w przypadku kodu QR ma długość 8 bitów, więc jeśli zniszczenie dotknie wszystkich powtórzeń tego słowa- kod nie zostaje odczytany. QR są czytelne pod każdym kątem obrotu w osi prostopadłej do płaszczyzny na której się znajdują. Kolejną zaletą jest możliwość podziału zbyt dużego symbolu na mniejsze, przy jednoczesnym zachowaniu danych. Ta własność w tej pracy jest wykorzystywana jako sposób optymalizacji transmisji w kanale optycznym.

Cechą szczególną kodów QR jest mechanizm maskowania, który powoduje, że średnia kolorów na dowolnych obszarach złożonych z modułów jest sobie jak najbliższa. Przez takie zrównoważenie występowania białych i czarnych modułów wzrasta szybkość odszukiwania i dekodowania obrazów przez skanery.



Rysunek 1.1: Schemat kodu QR

Na rysunku przedstawiono:

1. Informacja o wersji
2. Informacja o formatowaniu
3. Dane i korekcja błedów
4. Wzorce:
   - 4.1. Pozycje
   - 4.2. Wyrównanie
   - 4.3. Chronometraż

Kod QR można wykorzystać jako nośnik danych na kilka różnych sposobów. Pierwszy to przedstawienie wielu kodów QR tej samej wielkości, przy czym każdy z nich będzie zawierał jak najwięcej danych. Implikuje to sporej wielkości kody. Drugim jest generowanie pomniejszych kodów QR tej samej wielkości, które pozwalają na jednoczesną prezentację ich większej ilości na jednym ekranie. Natomiast trzecim generowanie przemiennych kodów QR, losowej wielkości. Pozwoli to stworzyć dla urządzenia czytającego tzw kontrast i zmusi je do ponownego wyszukiwania położenia QR na ekranie.

W kontekście wykorzystania kodu QR, wymaganiami funkcjonalnymi aplikacji w ramach wykonania pracy dyplomowej są:

- Urządzenie nadające tłumaczy plik na ciągi znaków, by następnie wyświetlić je jako kody QR.
- Urządzenie odbierające za pomocą wbudowanej kamery odbiera przesyłane dane, zapisując obraz z wyświetlacza urządzenia nadającego i następnie z pojedynczych klatek transmisji odczytuje kody QR.

Aplikacja w trybie nadawania musi pozwalać na modyfikacje takich cech jak:

- czas między kolejnymi sekwencjami QR,
- wielkość samych QR kodów,
- ilość QR kodów na ekranie i wątków.

Aplikacja w trybie odbierania musi zmaksymalizować względem czytania danych następujące:

- nagrywanie z maksymalną dostępną szybkością,
- nagrywanie z maksymalną dostępną rozdzielczością.

1.2 Parametry urządzeń

Problemem przeprowadzanego badania jest szukanie optymalnych parametrów dla urządzeń w komunikacji, którego znalezienie wymaga wstępnych założeń i zbadania ich poprawności empirycznie. Urządzenia mobilne korzystające natywnie ze swoich komponentów, takich jak karta graficzna lub procesor, posiadają ograniczenia specyfikacji. Rozdzielczość wraz z częstotliwością odświeżania wyświetlacza nie są jedynymi składowymi problemu. Urządzenie nie umożliwia pełnego wykorzystania jego komponentów pojedynczej aplikacji, ale również zależnie od ilości zainstalowanych aplikacji korzystających z procesora w tle - wydajność testowanego oprogramowania spada. Zatem potrzebne są techniki zmian kolejkowania zadań platformy, by zminimalizować użycie podzespołów dla procesów rozpraszających pracę testowanej aplikacji. Kolejnym niezbędnym do rozstrzygnięcia problemem jest ten leżący między urządzeniami komunikującymi się - widoczność sekwencji kodów. Jako, że aplikacja nadająca może przyjąć, że jest w stanie nadawać z maksymalną przepustowością na największej dostępnej jej rozdzielczości, to z kolei odbierająca nie gwarantuje, że w jej kadrze znajdą się wszystkie kraty z kodami, ale również odwrotna sytuacja może mieć miejsce. Kamera urządzenia powinna nagrywać z maksymalną dostępną jej rozdzielczością i maksymalną ilością klatek na sekundę. Wynika to z tego, że urządzenie odbierające nie jest w stanie przewidzieć z jaką częstotliwością będzie nadawało urządzenie nadające. Jeśli byłoby to możliwe od razu, to można by założyć, że dwukrotna przewaga częstości odbierania wystarczy do tego zadania (zobacz [8]).

Problem nie jest trywialny, gdyż po nawiązaniu połączenia, rozumianego tutaj jako odnalezienie punktów skupiających QR kody, należy rozróżnić kiedy transfer będzie większy. Większy w rozumieniu ilości danych przesłanych w określonej jednostce czasu. Dobranie parametrów by zmaksymalizować przepustowość jest jednak trudne do osiągnięcia, gdyż przy niewiele mniejszej rozdzielczości - ilość klatek na sekundę może wzrosnąć prawie dwukrotnie. Mając na względzie powyższe utrudnienia należy przyjąć, że osiągnięte wyniki będą optymalnymi tylko na urządzeniach tutaj wykorzystanych - wciąż zakładając, że ich producent nie wprowadził w nich zmian *w trakcie* lub *po* przeprowadzeniu badań.

Urządzenia wybrane na docelową serię badań zostały dobrane ze względu na trzy aspekty. Pierwszym jest ich wyświetlacz retina. Jego zaletą w badaniu jest wysoka gęstość piksli, dzięki której możliwe jest zbadanie przypadku dla dużej ilości małych QR kodów oraz małej ilości dużych QR kodów. Drugim aspektem jest zróżnicowanie procesora i jego zintegrowanej karty graficznej. Pozwala to na ustalenie czy odkodowanie kodu po jego znalezieniu wymaga dużego nakładu pracy. Trzecim- rodzaj kamery, tzn. ilość klatek na sekundę oraz jej rozdzielczość. Dobór odpowiedniego urządzenia czytającego pozwala w naturalny sposób zwiększyć ilość danych w transmisji. Wybrane zostały urządzenia mobilne, by wskazać, że dziś każdy użytkownik ogólnodostępnej technologii jest w stanie wymieniać się danymi z innymi użytkownikami bez względu na platformę czy typ urządzenia. Przedstawione tutaj dane dotyczą parametrów nagrywania i wyświetlania są dostępne w specyfikacji urządzeń marki Apple, z dnia 2 lipca 2016 roku (zobacz [1]).

Rysunek 1.2: Tabela przedstawiająca rozdzielczości kamery tylnej oraz przedniej w poszczególnych urządzeniach wraz z ich minimalnymi i maksymalnymi prędkościami klatek na sekundę.

model	ppx_{max}(przód)	ppx_{max}(tył)	fps_{min}	fps_{max}
iPhone 5c	720	1080	30	60
iPhone 5s	720	1080	30	120
iPhone 6	720	1080	30	240
iPhone 6s	720	4K	30	240
iPhone SE	720	4K	30	240

1.2.1 Odbieranie

Przepustowością odbierania danych dla wybranych urządzeń mobilnych jest ich maksymalna ilość klatek na sekundę czytanych przez kamerę (tylną), na co nałożony jest czas potrzebny na odczytanie pojedynczego symbolu QR z klatek. Przyjmując, że algorytm używany przez Apple w AVFoundation framework jest algorytmem optymalnym dla testowanej platformy. Biorąc pod uwagę, iż pojedyncze klatki obrazu są chwytane i dopiero na nich odszukiwany jest symbol QR oraz następuje jego dekodowanie, można zauważyć, że wciąż istnieje problem po stronie nadającego. Jest nim jego sposób odświeżania ekranu. Może się okazać, że kod QR w momencie chwytania klatki kamerą natrafia już na obraz w jakiejś części złożony z dwóch symboli, zamiast jednego.

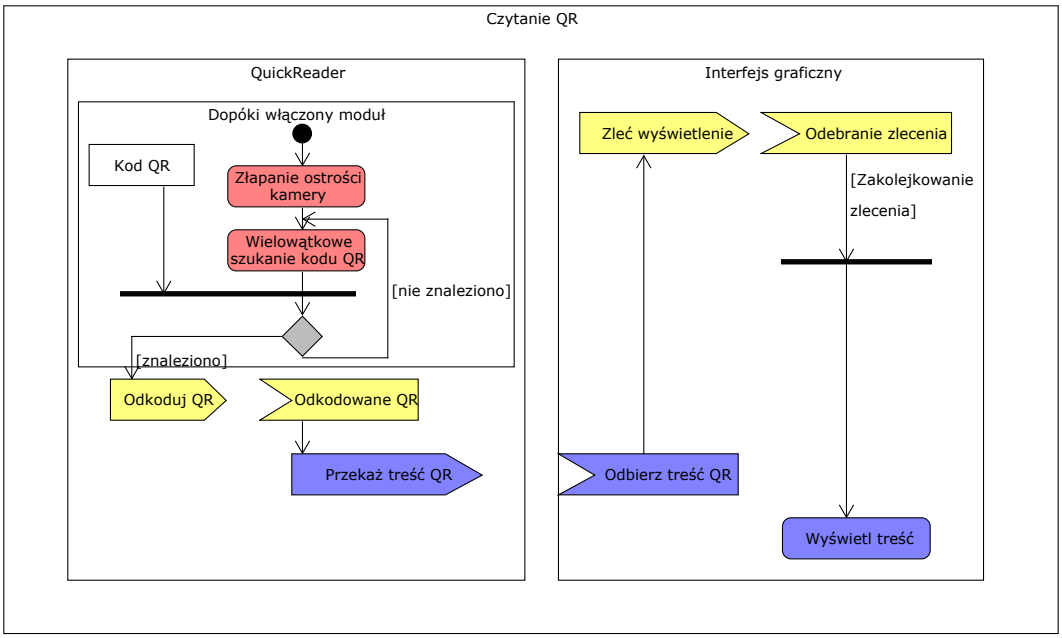
1.2.2 Nadawanie

Wyświetlanie kodów QR jest procedurą wymagającą zgrania z wyświetlaniem. Platforma *iOS* nie pozwala na prezentację grafiki nie będącej w całości załadowanej do pamięci podręcznej, zatem można przyjąć pierwszy problem za rozstrzygnięty. Kolejną kwestią do rozważenia jest sposób postrzegania kodów QR przez urządzenie odbierające. Jako wysyłający nie możemy spowodować, że urządzenie odbierające przestanie zauważać kody QR lub nie będzie w stanie ich odczytać przez zbyt małą dokładkość ich prezentacji. Zatem kody QR muszą mieć rozsądną wielkość, czytelną z takiej odległości, która zapewniałaby najlepszy transfer.

2 Projekt systemu

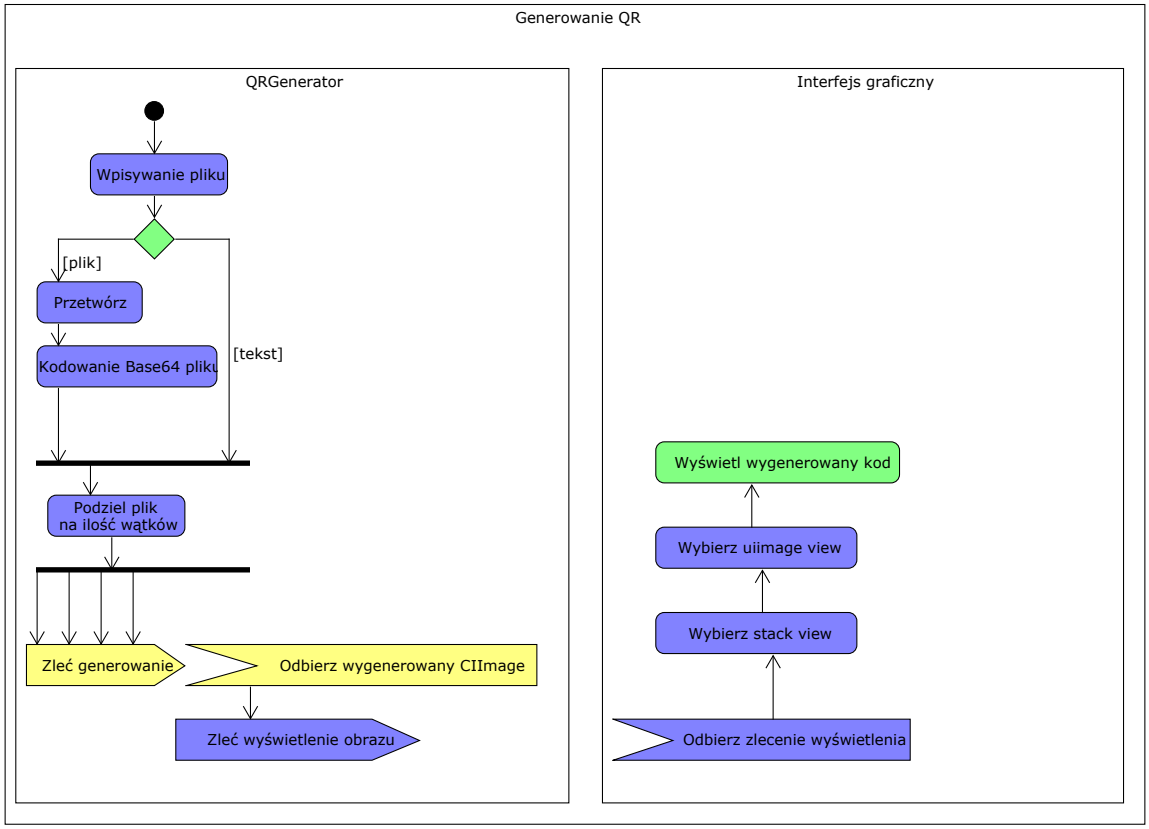
Ten rozdział przedstawia szczegółowy projekt aplikacji w notacji UML, uwzględnia on założenia funkcjonalne opisane w rozdziale 1.2. Opis relacji między składowymi systemu wyrażono diagramami.

2.1 Diagramy aktywności



Rysunek 2.1: Diagram przedstawia aktywności wykonywane w trakcie czytania kodów QR.

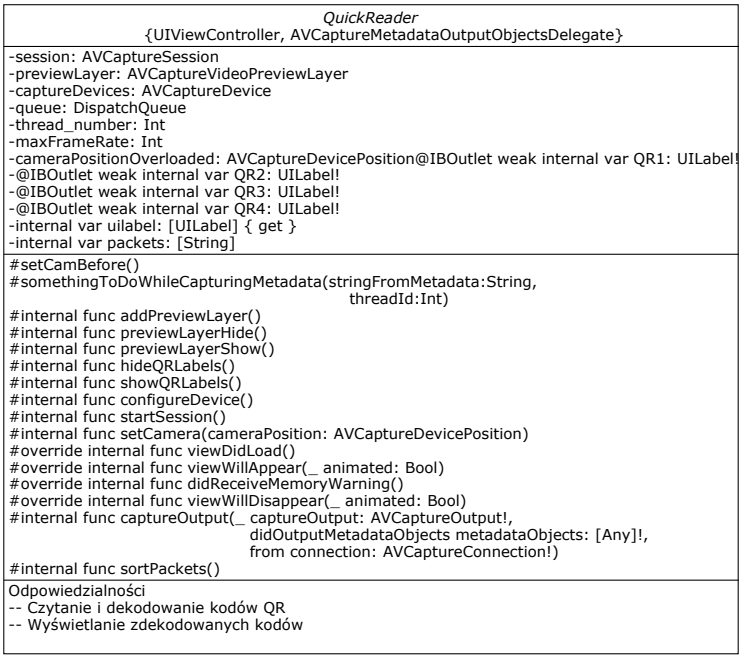
Klasa `QuickReader` po złapaniu ostrości w urządzeniu nagrywającym, chwyta kolejne klatki sesji wideo. Na kolejnych ujęciach cztery wątki szukają na tej samej klatce osobnych kodów QR. Po ich znalezieniu, przekazywana jest dana klatka do odkodowania. Klatki nie zawierające kodu są odrzucane. Wartość numeryczna kodu jest czytana i przekazywana do modelu klasy `QuickReader`, a następnie przekazuje on swoim kontrolerem tą wartość do interfejsu graficznego. Gdzie wyświetlanie również jest zlecane osobnym wątkom, aby nie spowalniać procesu nagrywania w wątku głównym.



Rysunek 2.2: Diagram przedstawia aktywności wykonywane w trakcie generowania kodów QR z pliku.

Klasa `QRCodeGeneratorViewController` jako `QRGenerator` po wybraniu pliku lub tekstu przez użytkownika, zależnie od wybranej opcji, zleca jego przetworzenie lub bezpośrednio zleca generowanie kodu QR. Generacja kodu w `QRGenerator` jest zależna od długości danych jakie należy przekonwertować. Podzielone na mniejsze części dane są zlecane jako osobne kody QR do wygenerowania. Stworzony QR w postaci `CIImage` zlecany jest do wyświetlenia w wątku głównym.

2.2 Diagramy klas

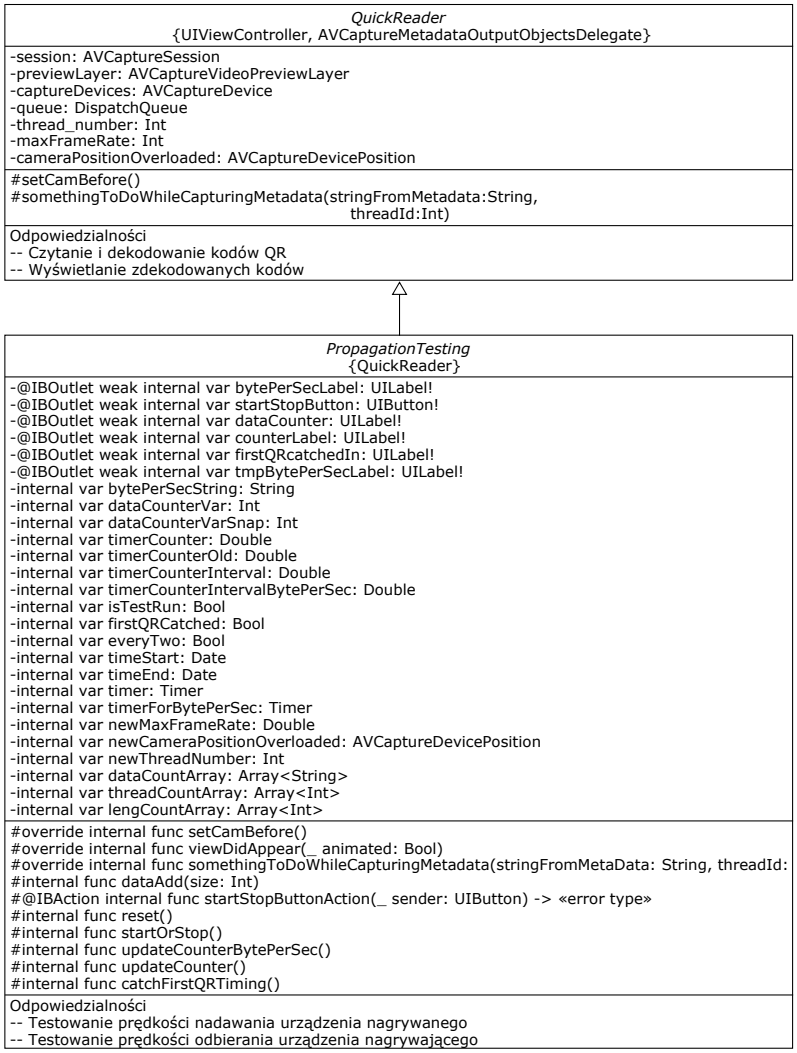


Rysunek 2.3: Diagram przedstawia moduł klasy `QuickReader`, służącej do szybkiego czytania kodów QR. Klasa wykorzystuje rozproszenie pracy na cztery wirtualne wątki.

Klasa `QuickReader` w prototypie aplikacji pełni funkcję widoku oraz kontrolera. Najważniejszą funkcjonalnością klasy jest natywne ustawianie kamery urządzenia z niestandardowymi właściwościami. Jako delegat dla protokołu `AVCaptureMetadataOutputObjects`, otrzymuje ona dane odczytane z obrazów przekazywanych za pomocą `AVCaptureSession` i `AVCaptureVideoPreviewLayer`. Pozwala to na szybkie dekodowanie kodów QR z wykorzystaniem metod zoptymalizowanych pod platformę iOS. Ważną właściwością klasy jest specjalna kolejka `DispatchQueue` oraz funkcja `somethingToDoWhileCapturingMetadata(...)`, dzięki którym aplikacja nie korzysta z wątku głównego i nie blokuje interfejsu graficznego oraz pozwala na podglądanie, który wątek obsługiwał czytanie kodu QR, oraz wykonanie określonej czynności zleconej przez klasę dziedziczącą po klasie `QuickReader`.

Rysunek 2.4: Diagram przedstawia moduł klasy `QRCodeGeneratorViewController`, służącą do generowania sekwencji kodów QR na wyświetlaczu urządzenia.

Klasa `QRCodeGeneratorViewController` jest kontrolerem oraz widokiem zaimplementowanym wyłącznie do generowania kodów QR. Korzysta ona z `CIFilter` do tworzenia obrazów `CIImage` natywnie. `CIImage` jest przekazywany przez referencję, co pozwala zaoszczędzić pamięć w przekazywaniu do klas rozszerzających obrazów z QR kodami. Kolejną zaletą użycia w tej klasie `CIFilter` jest jego własność `thread-safe`, klasa ta pozwala tworzyć osobne instancje obiektu, czyli każdy wątek pracuje na własnej kopii filtra obrazu. Dzięki temu, programista może rozszerzać klasę o generowanie QR kodów o różnej korekcji błędów.



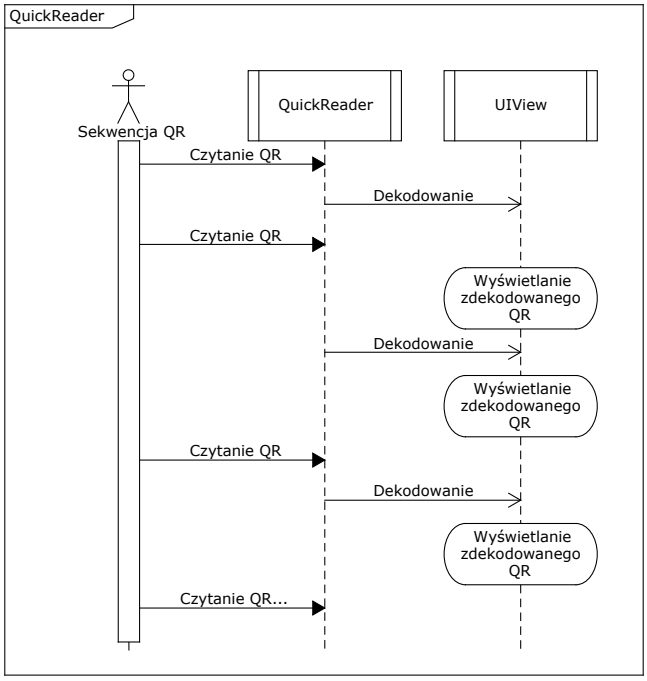
Rysunek 2.5: Diagram przedstawia moduł klasy `PropagationTesting`. Klasa ta testuje prędkości nadawania urządzenia zewnętrznego, które jest czytane przez aplikację uruchomioną na aparacie testującym.

Klasa `PropagationTesting` jest klasą dziedziczącą po klasie `QuickReader`. Korzystając z jej metody `somethingToDoWhileCapturingMetadata(...)` pozwala prowadzić statystyki o odbieraniu kodów QR. Każdy nowy odczytany kod QR jest zwracany wraz z numerem wątku i wartością kodu QR. Na podstawie zwracanych danych można określać czas łapania ostrości obrazu przez urządzenie. Prędkość nadawania danych urządzenia zewnętrznego i inne zaimplementowane moduły w aplikacji.

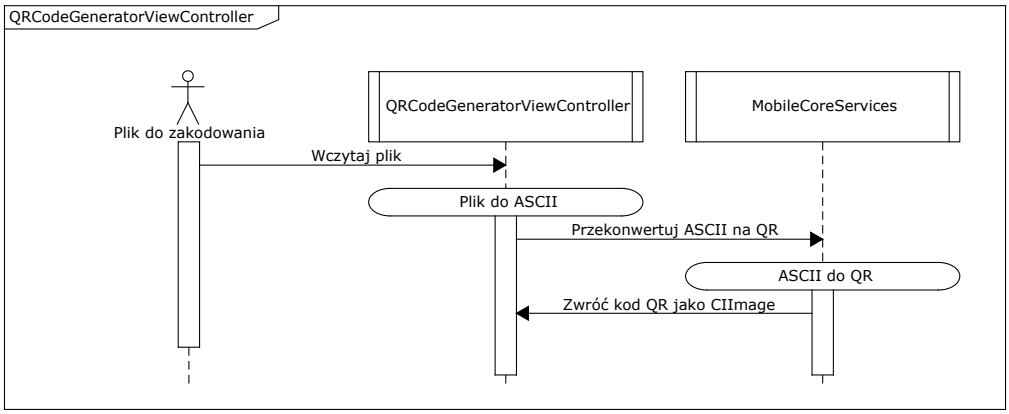
Rysunek 2.6: Diagram przedstawia moduł klasy `KeyExchanger`, której zadaniem jest wymiana klucza między dwoma urządzeniami. Moduł ten wykorzystuje wszystkie generyczne klasy aplikacji.

Klasa `KeyExchanger` dziedziczy po `QuickReader` oraz `QRCodeGeneratorViewController`. Jej zadaniem jest badanie wymiany klucza między dwoma urządzeniami. Szczególnymi cechami działania klasy jest sposób generowania kodów QR po każdym odczytaniu kodu z innego urządzenia. Jeśli w klasie `QRCodeGeneratorViewController` jest zaimplementowany specjalny protokół nadawania, np. enkapsulacja danych w ramki, to można na podstawie całej wymiany danych wnioskować w jakiej kolejności pakiety potwierdzały swoją trasę między urządzeniami.

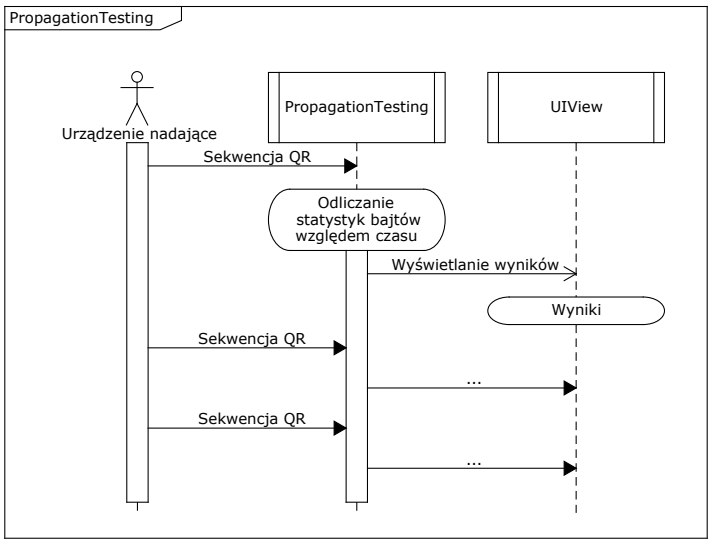
2.3 Diagramy sekwencji



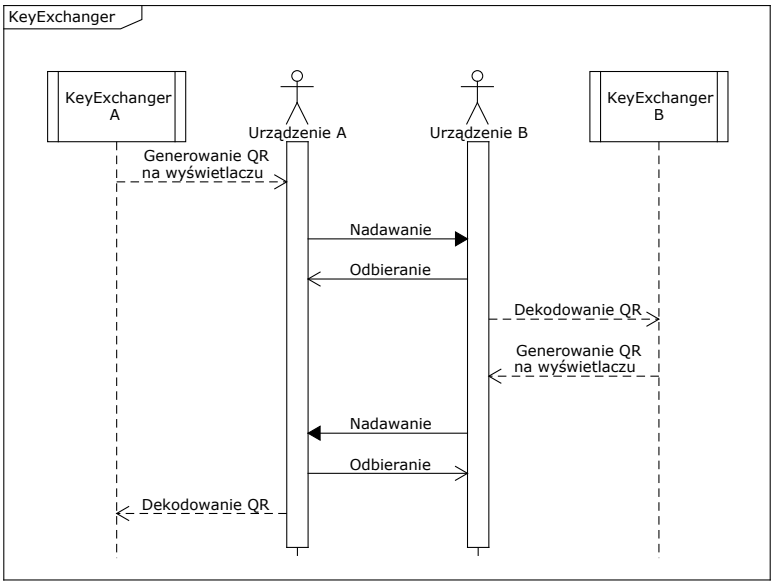
Rysunek 2.7: Diagram przedstawia sekwencję akcji wykonywanych przez klasę `QuickReader`, której zadaniem jest odczytanie kodu QR i zdekodowanie go do postaci tekstowej oraz wyświetlenie wartości na wyświetlaczu.



Rysunek 2.8: Diagram przedstawia zbiór sekwencji klasy `QRCodeGeneratorViewController` powtarzanych przez osobne wątki w aplikacji. Użytkownik wybiera plik lub tekst do zakodowania w postaci kodu QR lub sekwencji kodów QR. Następnie plik ten jest odczytywany w formie bajtkodu, jego pojedyncze bajty są traktowane jako kody ASCII i zostaje zlecone do `MobileCoreServices` jego zakodowanie.

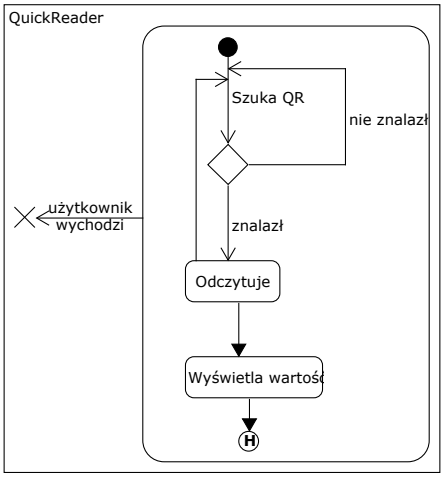


Rysunek 2.9: Diagram sekwencji przedstawia schemat działania klasy `PropagationTesting`, której zadaniem jest zbadanie prędkości nadawania urządzenia zewnętrznego. Klasa ta umożliwia wyświetlenie szczegółowych statystyk o prędkościach odbierania sekwencji kodów QR.

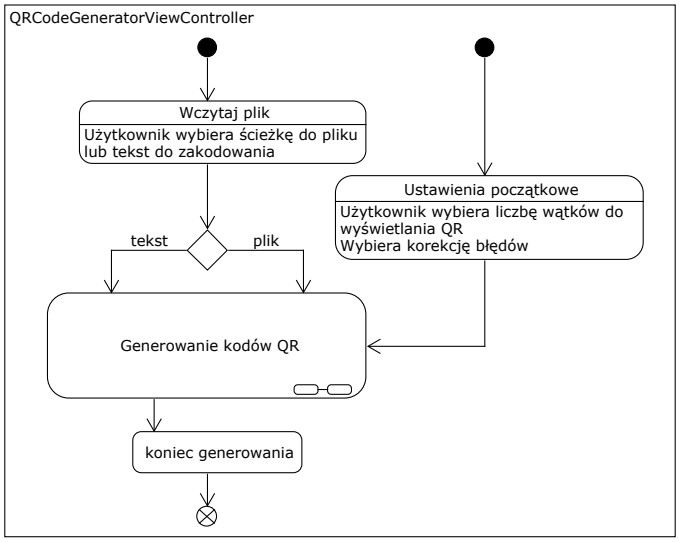


Rysunek 2.10: Diagram sekwencji przedstawia klasę `KeyExchanger`, która sprawdza wymianę klucza między dwoma urządzeniami. Urządzenie A rozpoczyna konwersację przez uruchomienie modułu, następnie moduł ten uruchamia urządzenie B. W momencie, w którym urządzenie B odczyta jakikolwiek kod z wyświetlacza urządzenia A, zaczyna na jego podstawie generować odpowiedź na swoim wyświetlaczu. Następnie urządzenie A, odczytując QR z B, reaguje i również odpowiada. Czynność jest wykonywana do momentu ujednolicenia kodów QR na ekranach urządzeń.

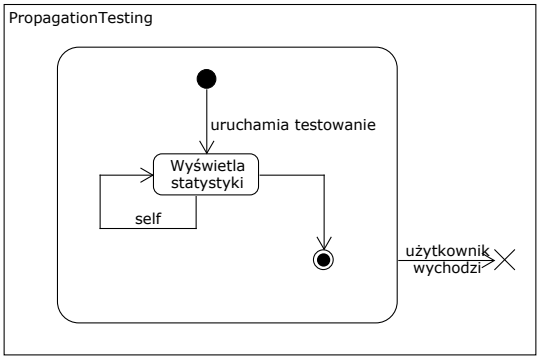
2.4 Diagramy stanów



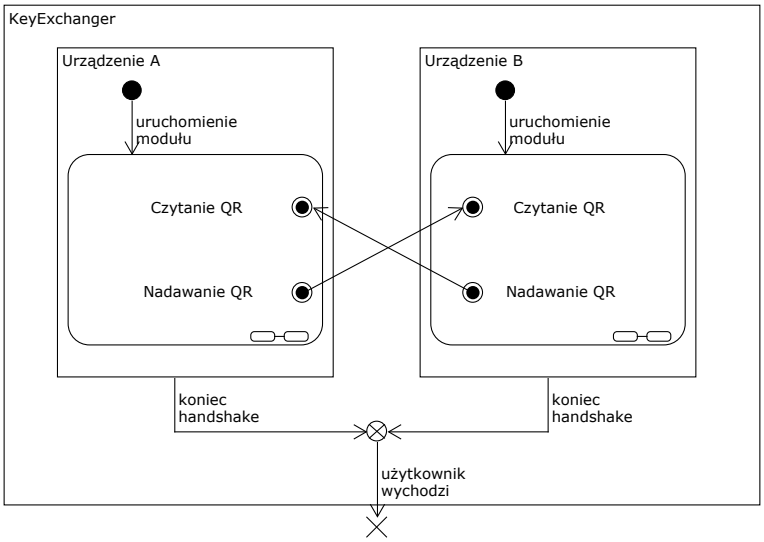
Rysunek 2.11: Diagram stanów dla modułu `QuickReader`.



Rysunek 2.12: Diagram stanów dla modułu `QRCodeGeneratorViewController`.



Rysunek 2.13: Diagram stanów dla modułu `PropagationTesting`.



Rysunek 2.14: Diagram stanów dla modułu `KeyExchanger`.

3 Implementacja systemu

3.1 Opis technologii

Podczas implementacji systemu wykorzystano następujące technologie:

- Język programowania: Swift®, wersje kolejno 2, 2.1, 2.2, 2.3, 3.0, 3.0.1,
- Apple® LLVM version 8.0.0 (clang-800.0.42.1),
- Target: x86_64-apple-darwin15.6.0,
- Thread model: posix.

Wykorzystano środowisko programisty Xcode, wersja 8.1 (8B62). W tworzeniu prototypu duży wpływ na końcową wersję miały często zmiany w platformie iOS oraz Swift. Włączając w to duże zmiany w platformach zarówno iOS jak i OS X. Pisząc pracę wymagane były dwie migracje, które pomimo wielu zmian pozwoliły na zbadanie natywnych bibliotek i ich sposobów czytania kodów QR. Ostateczną wersją, która była użyta w pracy jest - 8.1 (8B62).

Dodatkowo podczas implementacji wykorzystano następujące biblioteki (zobacz [3]):

- `UIKit` framework - konstrukcja i zarządzanie interfejsem iOS. Reakcje na interakcje użytkownika i wydarzenia systemowe oraz dostęp do natywnych cech urzadzenia, takich jak widoki, przejścia i animacje.
- `AVFoundation` framework - nagrywanie, przychwytywanie sesji kamery. Framework ten zapewnia interfejs dla metod napisanych w języku Objective-C, dzięki któremu możliwe jest używanie audio-wizualnych właściwości platformy.
- MobileCoreServices framework - operacje graficzne z użyciem natywnych metod.

Wzorcami wykorzystanymi w pracy są: `Model View Controller` - podczas tworzenia oprogramowania Swift® przez swą budowę wymaga użytkowania. `Fasada` - Ważny w prototypie aplikacji, dzięki któremu dostępny jest zbiór strategii dla transmisji kodów QR.

Powodem wykorzystania wyżej wymienionych frameworków jest pokazanie, że każde urządzenie mobilne marki Apple, które posiada kamerę oraz wyświetlacz retina, jest w stanie za pomocą tego samego kodu źródłowego dokonać transmisji kanałem optycznym.

3.2 Omówienie kodów źródłowych

Kod źródłowy 69 przedstawia klasę tworzącą generyczny `view controller`, który jest wykorzystywany w implementacjach poszczególnych modułów aplikacji. Interfejs ten pozwala na uruchomienie kamery urządzenia z dowolnymi parametrami oraz umożliwia czytanie i dekodowanie kodów QR. Kompletne kody źródłowe znajdują się na płycie CD dołączonej do niniejszej pracy w katalogu `Kody` (patrz Dodatek A).

```
1 import UIKit
2 import AVFoundation
3 internal class QuickReader : UIViewController,
4                              AVCaptureMetadataOutputObjectsDelegate {
5     /// Obiekt czytający dane z urządzeń
6     internal let session: AVCaptureSession
7     /// Powłoka przechwytująca oraz wyświetlająca podgląd z sesji
      urządzenia
8     internal var previewLayer: AVCaptureVideoPreviewLayer?
9     /// Interfejs przedstawiający wszystkie wejścia urządzenia
10    internal var captureDevices: AVCaptureDevice?
11    /// Kolejka do oddelegowania zadania w systemie
12    internal let queue: DispatchQueue
13    /// Kolejka do oddelegowania zadań
14    /// które niezależnie od czasu ukończenia musza zostać wykonane
15    internal let backgroundQueue: DispatchQueue
16    /// Liczba wątków do dekodowania QR kodów
17    internal var thread_number: Int
18    /// Maksymalna ilość klatek na sekundę definiowana w poprzednim widoku
19    internal var maxFrameRate: Double
20    /// Pozycja kamery: Pozioma | Pionowa
21    internal var cameraPositionOverloaded: AVCaptureDevicePosition
22    /// Etykieta do wyświetlania zawartości zdekodowanego QR kodu
23    @IBOutlet weak internal var QR1: UILabel!
24    @IBOutlet weak internal var QR2: UILabel!
25    @IBOutlet weak internal var QR3: UILabel!
26    @IBOutlet weak internal var QR4: UILabel!
27    /// Tablica wewnętrznych Etykiet QR
28    internal var uilabel: [UILabel] { get }
29    /// Tablica zawierająca wszystkie zdekodowane QR
30    internal var packets: [String]
31    /// Dodawanie podglądu kamery do rozszerzanego view controllera
32    internal func addPreviewLayer()
33    /// Ukrywanie podglądu
34    internal func previewLayerHide()
35    /// Włączenie podglądu
36    internal func previewLayerShow()
37    /// Ukrywanie Etykiet QR
38    internal func hideQRLabels()
39    /// Włączenie Etykiet QR
40    internal func showQRLabels()
41    /// Konfiguracja urządzenia, jednokrotne na instancję
42    internal func configureDevice()
43    /// Uruchomienie sesji kamery
44    internal func startSession()
```

```
45     /// Ustawienie kamery ze zmienną pozycją
46     internal func setCamera(cameraPosition: AVCaptureDevicePosition)
47     /// Funkcja do ustawienia kamery przed pierwszym użyciem
48     internal func setCamBefore()
49     /// Standardowa funkcja po załadowaniu widoku
50     override internal func viewDidLoad()
51     /// Standardowa funkcja po ukazaniu widoku
52     override internal func viewWillAppear(_ animated: Bool)
53     /// Standardowa funkcja po otrzymaniu ostrzeżenia o braku pamięci
54     override internal func didReceiveMemoryWarning()
55     /// Standardowa funkcja po zniknięciu widoku
56     override internal func viewWillDisappear(_ animated: Bool)
57     /// Funkcja protokołu do zaimplementowania,
58     /// co instancja QuickReader ma wykonywać
59     /// podczas przeczytania przez wątek QR kodu
60     internal func somethingToDoWhileCapturingMetadata(stringFromMetaData:
       String,
61                                                       threadId: Int)
62     /// Przechwytywanie meta danych z połączenia z kamerą za pomocą
       delegacji
63     internal func captureOutput(_ captureOutput: AVCaptureOutput!,
64                                 didOutputMetadataObjects metadataObjects:
       [Any]!,
65                                 from connection: AVCaptureConnection!)
66     /// Sortowanie pakietów wedle zaimplementowanego protokołu transmisji
       danych
67     internal func sortPackets()
68 }
```

Kod źródłowy 3.1: Szybkie czytanie kodów QR: `QuickReader`.

Klasa `QuickReader` służy jako generyczny model widoku, który posiada podgląd obrazu z kamery. Klasa ta jest delegatem `AVCaptureMetadataOutputObjects`. Oznacza to, że wszystkie metadane przechwycone przez skonfigurowaną pod konkretny rodzaj, tutaj kod QR, trafiają do klasy `QuickReader`, do funkcji `captureOutput(...)`. Konfigurację kamery pominięto w objaśnieniach z powodu możliwych zmian w implementacji przy kolejnych aktualizacjach całej platformy iOS. `QuickReader` zbiera teksty kodów QR w formie tablicy pakietów, które następnie wedle określonego protokołu w funkcji `sortPackets()` mogą zostać scalone w plik wynikowy.

Kod źródłowy 65 przedstawia opisy poszczególnych metod interfejsu: `QRCodeGeneratorViewController`. Kompletne kody źródłowe znajdują się na płycie CD dołączonej do niniejszej pracy w katalogu `Kody` (patrz Dodatek A).

```
1 import UIKit
2 import MobileCoreServices
3
4 internal class QRCodeGeneratorViewController : UIViewController,
5                                                UIDocumentPickerDelegate {
6     /// Inicjalizator generowanych QR kodów
7     internal var qrcodeImage: CIImage!
8     /// Zmienna przyjmująca tekst do przedstawienia jako QR kod
9     internal var target: String
```

```
10    /// Aktualna ilość wątków w generowaniu QR
11    internal var thread_number: Int
12    /// Co ile symboli ma być generowany QR kod
13    internal var text_spread: Int
14    /// Krok czasu do rozpoczęcia generacji kolejnego QR kodu
15    internal var stepDelay: Double
16    /// Licznik ile objektów QR zostało wygenerowanych
17    internal var dataObjects: Int
18    /// True - Low | False - High : Poziom korekcji błędów w QR
19    internal var swBool: Bool
20    /// --------------------
21    internal var files: [AnyObject]
22    /**
23        Poniżej specjalne podziały widoku na podwidoki,
24        służące do poprawnego wyświetlania QR kodów
25    */
26    @IBOutlet weak internal var stackViewAll: UIStackView!
27    @IBOutlet weak internal var stackViewSubUp: UIStackView!
28    @IBOutlet weak internal var stackViewSubMid: UIStackView!
29    @IBOutlet weak internal var stackViewSubDown: UIStackView!
30    /// Widok typu ImageView, do wyświetlania QR jako obrazu UIImage
31    @IBOutlet weak internal var imgQRCode: UIImageView!
32    @IBOutlet weak internal var imgQRCode2: UIImageView!
33    @IBOutlet weak internal var imgQRCode3: UIImageView!
34    @IBOutlet weak internal var imgQRCode4: UIImageView!
35    @IBOutlet weak internal var imgQRCode5: UIImageView!
36    @IBOutlet weak internal var imgQRCode6: UIImageView!
37    /// Ukryte pole do przechowywania nazw z poprzednich widoków
38    @IBOutlet weak internal var textField1: UITextField!
39    /// Tablica wszystkich widoków imgQRCode*
40    internal var t_imgQRCodes: [UIImageView] { get }
41    /// Tablica wszystkich pól do przechowywania nazw z poprzednich
      widoków
42    internal var tField: [UITextField] { get }
43    /// ----------------------
44    internal func documentPicker(_ controller:
      UIDocumentPickerViewController,
45                         didPickDocumentAt url: @autoclosure URL) ->
      <<error type>>
46    /// ----------------------
47    internal func documentPickerWasCancelled(_ controller:
      UIDocumentPickerViewController) -> <<error type>>
48    /// Przeciążenie standardowej funkcji, ukrycie paska stanu
49    override internal var prefersStatusBarHidden: Bool { get }
50    /// Standardowa funkcja, inicjalizacja po załadowaniu widoku
51    override internal func viewDidLoad()
52    /// Standardowa funkcja, inicjalizacja po ukazaniu widoku
53    override internal func viewDidAppear(_ animated: Bool)
54    /// Ukrycie wszystkich podwidoków na widoku głównym
55    internal func hideStackViews()
56    /// Pokazanie wszystkich podwidoków na widoku głównym
57    internal func showStackViews()
58    /// Funkcja otrzymująca oddelegowane sygnały z przycisku na widoku
      głównym
```

```
59      @IBAction internal func performButtonAction(_ sender: AnyObject)
60      /// Funkcja generujaca QR kody z plików testowych
61      internal func generateTestGivenInString(str: String)
62      /// Funkcja generująca QR kody na podstawie niezajętości UIImageViews
63      internal func displayQRCodeImage(_ imageIter: Int)
64  }
```

Kod źródłowy 3.2: Generowanie QR kodów na wątkach `QRCodeGenerator`.

Klasa `QRCodeGeneratorViewController` służy jako model widoku kontrollera, obsługującego generowanie kodów QR na maksymalnie sześciu `UIImageView`.

Platforma iOS posiada wiele typów zarówno wysokiego jak i niskiego poziomu abstrakcji do obsługi wątkowości na wszystkich swoich urządzeniach. Do zmaksymalizowania przepustowości w kanale optycznym użyte zostały `DispatchWorkItems`, `DispatchQueues`:(`Async` oraz `Sync`). Kolejkowanie z użyciem `Grand Central Dispatch` umożliwia stworzenie struktur, wymaganych do wygenerowania kodów QR w odpowiedniej sekwencji, a następnie w głównym wątku wyświetlenie ich na wyświetlaczu urządzenia. Każda aplikacja posiada własną specjalną kolejkę, która jako jedyna może zarządzać widzialnym interfejsem. Pozostałe pozwalają nadawać priorytety, na bazie których możliwe jest utrzymanie aplikacji przy życiu i wykonywanie złożonych obliczeń. `DispatchWorkItem` jest blokiem kodu do wykonania w określonym momencie. Służyć to może do tworzenia zadań atomowych albo w przypadku tej pracy jako sekwencjonowanie przygotowania pliku zgodnie z protokołami transmisji, generowania kodów QR jako obrazów gotowych do wyświetlenia oraz samego wyświetlenia ich na ekranie. Rozbijając te czynności na pomniejsze oraz nie przeplatając ich, zapewnia się urządzeniu brak czytania/pisania do wspólnej pamięci- optymalizując przy okazji czas wykonania zadań i możliwość parametryzacji wyświetlania kodów QR. Z tablicy gotowych kodów, można dodawać do wątku głównego kolejno wszystkie sekwencje i decydować w jakim odstępie czasu mają się wyświetlać nowe. W przypadku generowania i wyświetlania kodów jeden po drugim, czynność taka byłaby niemożliwa, gdyż czas generowania mógłby być różny dla każdego kodu QR. Aplikacje pisane w języku Swift mogą korzystać z `Automatic Reference Counter`, który to liczy ilość nowych objektów używanych przez aplikację od poszczególnych użyć innych objektów. W momencie odliczenia do zera, zwalnia pamięć objektu. Pozwala to na oszczędność w przejściach między kolejnymi widokami aplikacji, ale również na przechowanie dużej ilości danych, a w przypadku, gdy stają się one niepotrzebne na zwolnienie ich. Platforma, umożliwia oszczędność cykli procesorów przy odliczaniu czasu, dzięki specjalnej strukturze funkcji. Struktura ta pozwala zaznaczyć jaka funkcja ma się wykonać w określonych interwałach czasowych, które to zostają odliczane wspólnie w jednym wątku. Wątek ten odlicza czas jaki upłynął w nanosekundach od ostatniego uruchomienia systemu.

4 Testy

Testowanie aplikacji odbywało się w trybie `Debug`. Zatem prędkości wykonania mogą różnić się od tych, które użytkownicy mogą uzyskać w trybie `Release`.

4.1 Przepustowości nadawania

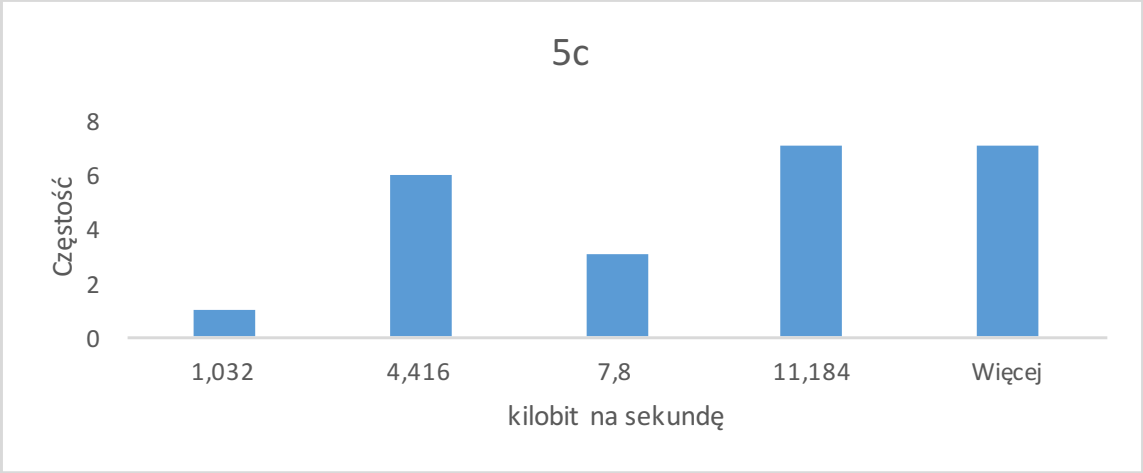
Nadawanie jest niezależne względem czynników zewnętrznych. Ograniczeniami urządzenia są jego własne możliwości podane w specyfikacji oraz jego procesy uruchomione w tle. Zakładając, że urządzenie nie będzie przechodziło w trakcie testów w stan inny niż aktywny i pierwszoplanowy, przyjęty zostaje brak przerw w nadawaniu.
Mamy następujący wzór dla przesyłania danych.

- Zatem $V_1 = (b/s)$,

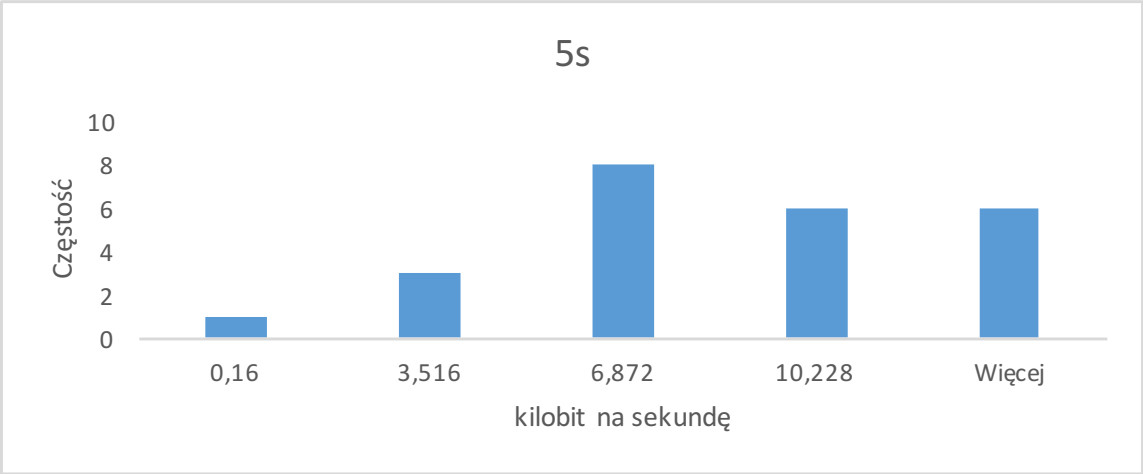
gdzie b to ilość bajtów w jednym QR kodzie, a s jest jednostką czasu. Następnie biorąc pod uwagę, że można przesyłać jeden duży QR kod lub wiele mniejszych, wzór można przekształcić następująco:

- Niech q oznacza ilość QR kodów, jakie są wyświetlane w tej samej jednostce czasu.
- Zatem $V_2 = ((b * q)/s)$

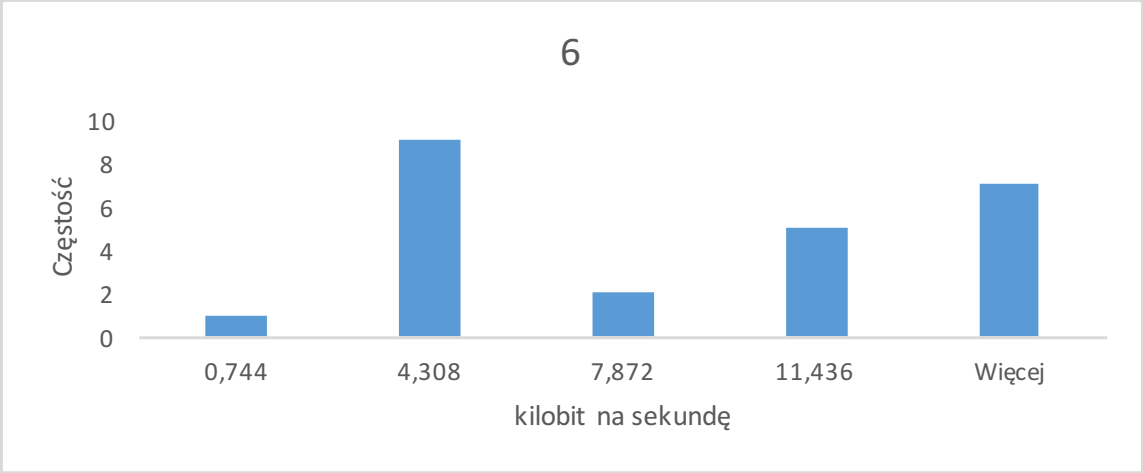
Należy jednak wziąć pod uwage, że kod QR posiada korekcję błędów na czterech różnych poziomach. Zatem część obszaru przez niego pokrywanego nie będzie nośnikiem danych, a powtórzeniem już istniejących, co także uwzględniamy. Wobec czego powyższy wzór należy przedstawić następująco: $V_3 = ((b * q)/s) - V_2 * C$,
gdzie C jest jedną z czterech wartości: { 7%, 15%, 25%, 30% }.



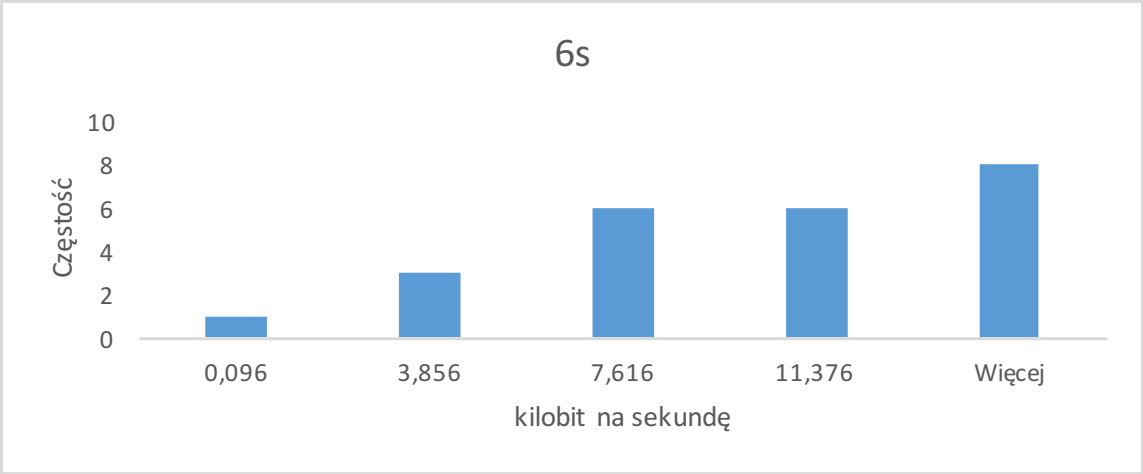
Rysunek 4.1: Prędkość nadawania QR kodów na iPhone 5c. Dolna oś przedstawia górną granicę przedziału dla transferu danych.



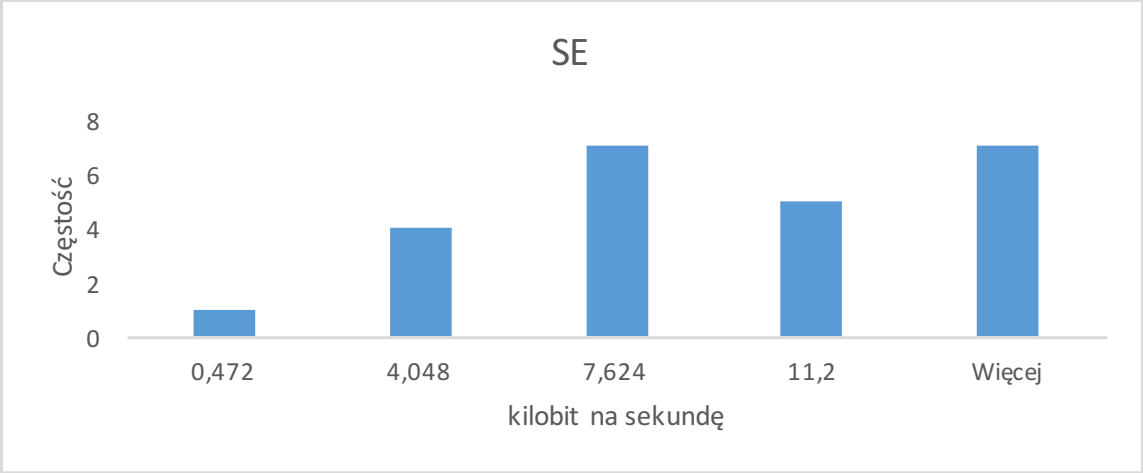
Rysunek 4.2: Prędkość nadawania QR kodów na iPhone 5s. Dolna oś przedstawia górną granicę przedziału dla transferu danych.



Rysunek 4.3: Prędkość nadawania QR kodów na iPhone 6. Dolna oś przedstawia górną granicę przedziału dla transferu danych.



Rysunek 4.4: Prędkość nadawania QR kodów na iPhone 6s. Dolna oś przedstawia górną granicę przedziału dla transferu danych.



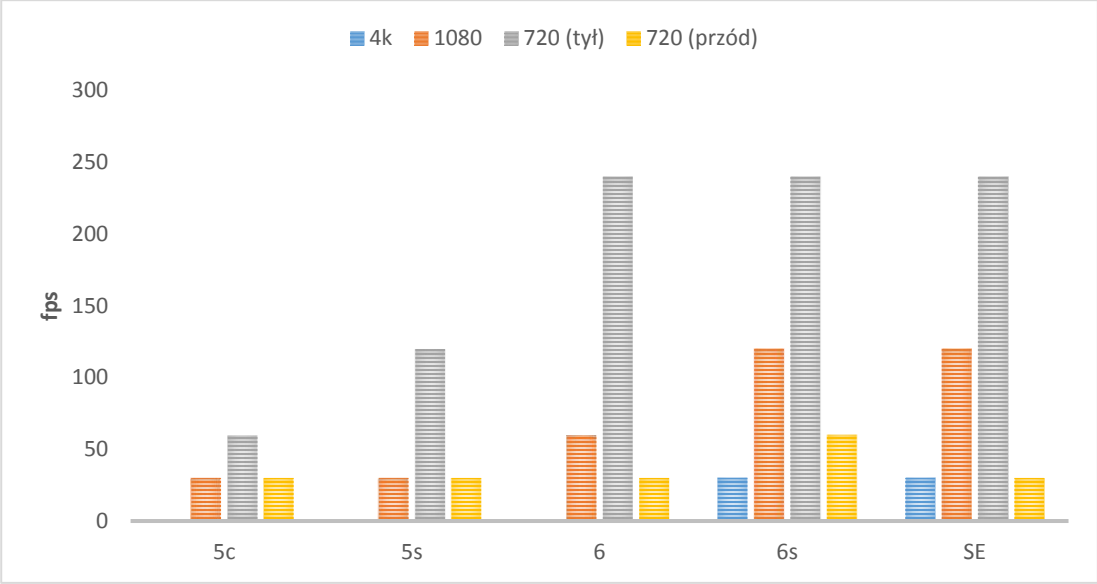
Rysunek 4.5: Prędkość nadawania QR kodów na iPhone SE. Dolna oś przedstawia górną granicę przedziału dla transferu danych.

Uśredniona względem zbadanych odległości liczba kilobitów na sekundę dla nadawania w czasie rzeczywistym danych prezentując tą samą ilość danych:
- Urządzenie 5c - 0,838 kB/s,
- Urządzenie 5s - 1,0 kB/s,
- Urządzenie 6 - 1,662 kB/s,
- Urządzenie 6s - 2.0 kB/s,
- Urządzenie SE - 1.425 kB/s.

Urządzenia 5c i 5s różnią się od siebie jedynie procesorem. Ich wyświetlacze są takie same. Można zauważyć, że w średnich wynikach dla nadawania kanałem optycznym wielkość wyświetlacza ma spore znaczenie, widać różnicę między najnowszym modelem SE, a starszym o dwie wersje iphonem 6. Mimo, że SE posiada procesor dwurdzeniowy A9 o mocy 1,85 GHz, a iphone 6 ma dwurdzeniowy A8 o mocy 1,4 GHz, to prędkość nadawania jest wyższa dla modelu o większym wyświetlaczu - iphone 6. Wynika to z jakości kodów QR, ponieważ odbierający ma dostęp do wyższej rozdzielczości kodów oraz dystansu między urządzeniami. We wszystkich przypadkach badań urządzenie odbierające było oddalone o 30cm.

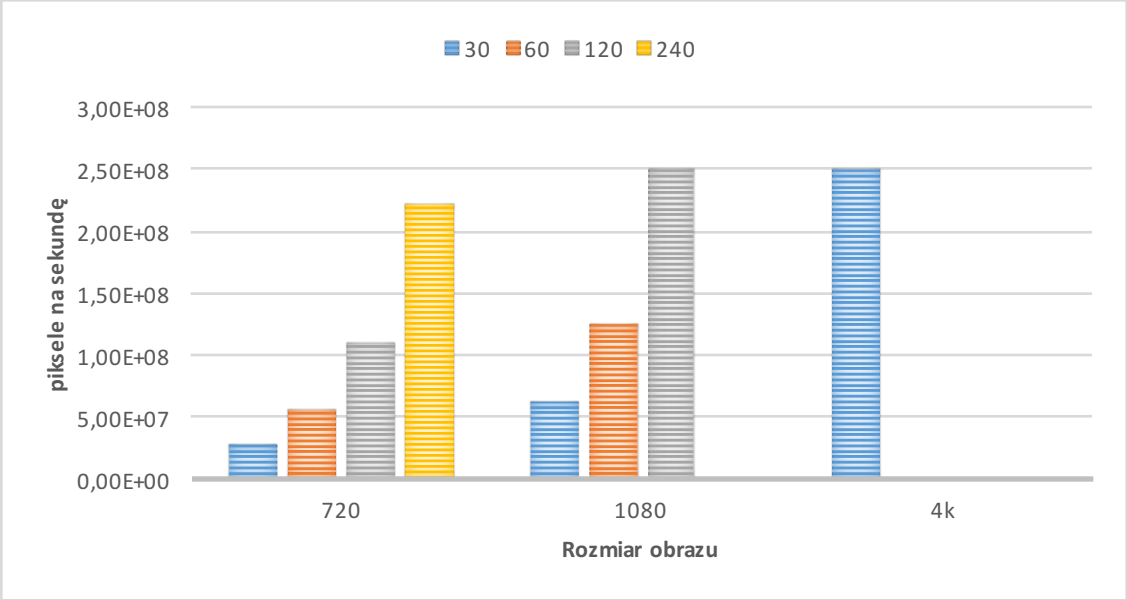
4.2 Badanie przepustowości odbioru

Ze względu na środowisko badania należy przyjąć, że urządzenie nadające jest w stanie propagować poprawnie wszystkie bity transmisji oraz połączenie między urządzeniami jest nieprzerywalne. Do badania wymagane jest urządzenie, które w sposób stały wyświetla kod QR oraz prototyp aplikacji, umożliwiającej zbadanie technicznych możliwości urządzeń wziętych pod uwagę w tym badaniu. W ten sposób pozwalamy by odbierający miał możliwość przetestowania swojej prędkości odbierania bez zakłóceń zewnętrznych.

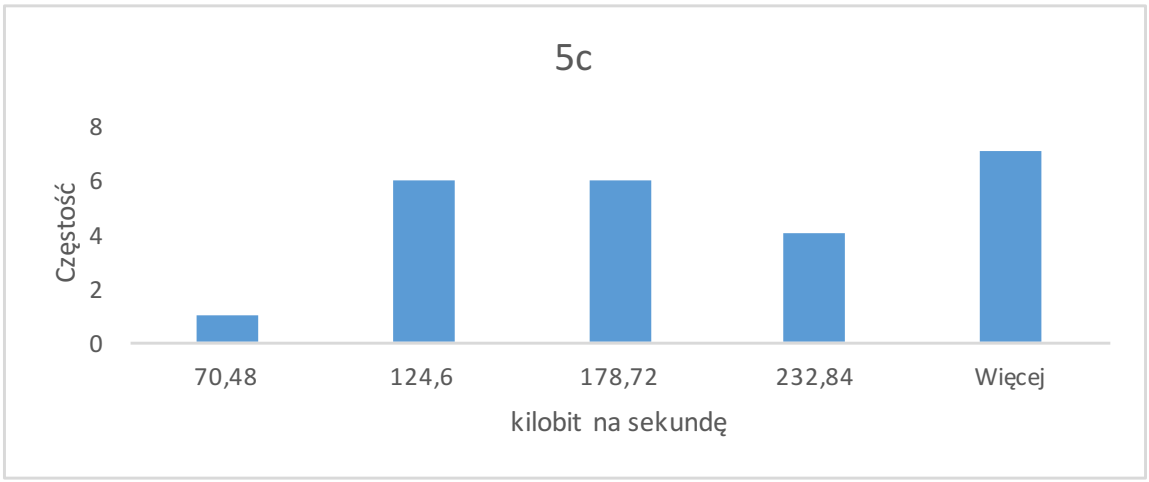


Rysunek 4.6: Wykres przedstawia możliwości poszczególnych ustawień kamery dla odbierania danych. Oś pionowa przedstawia ilość klatek na sekundę, pozioma natomiast rozdzielczość nagrywania.

Na podstawie danych z wykresu (4.6) przedstawiono liczbę pikseli jaką mogą czytać poszczególne ustawienia kamery w badanych urządzeniach na rysunku (4.7). Maksymalne parametry urządzeń przedstawiono w tabeli (1.2).

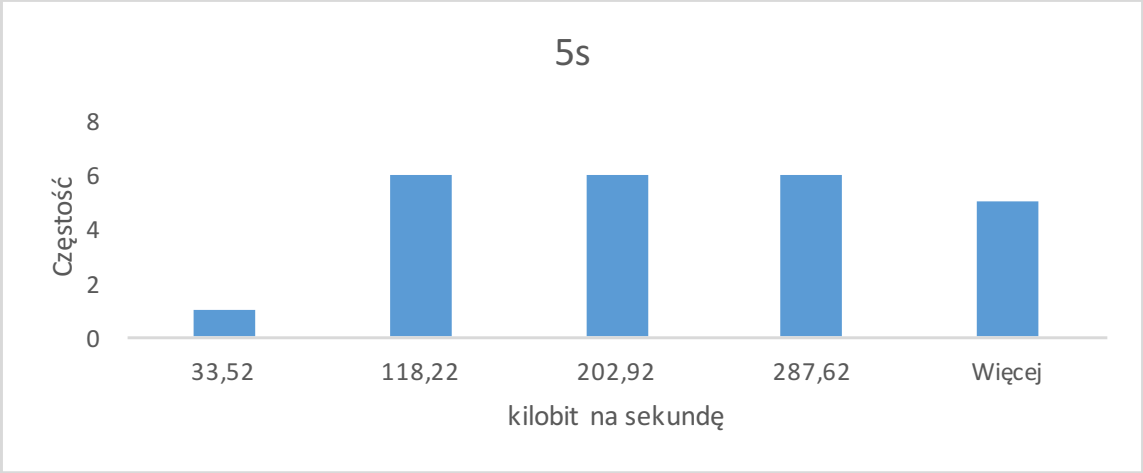


Rysunek 4.7: Wykres przedstawia całkowitą liczbę pikseli na sekundę, jaką może czytać jedna z trzech rozdzielczości kamer przy: 30, 60, 120 lub 240 klatkach na sekundę.



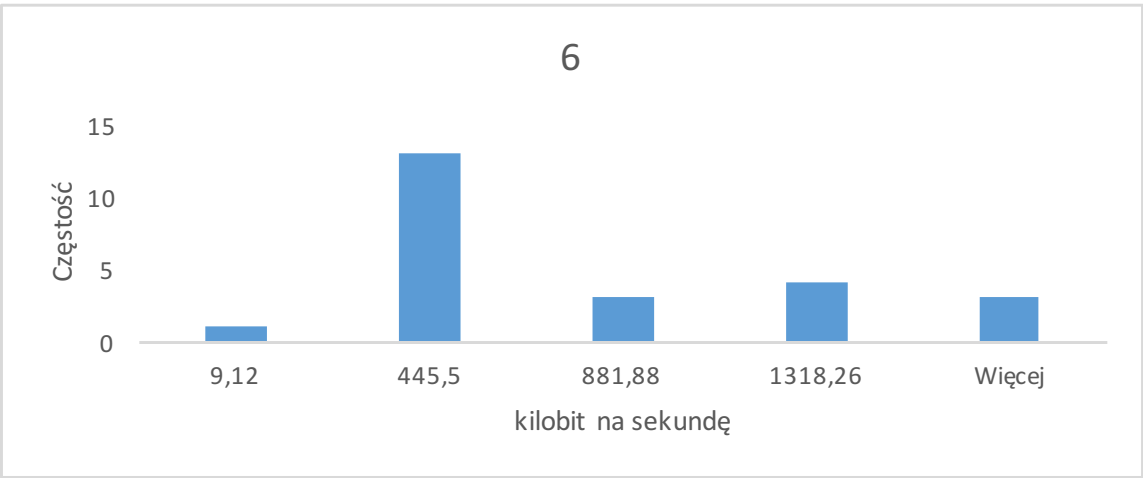
Rysunek 4.8: Prędkości czytania kodów QR na urządzeniu iphone 5c.

Iphone 5c jest w stanie odebrać maksymalnie 35 kilobajtów na sekundę ze statycznego kodu QR. Urządzenie to posiada dwurdzeniowy procesor A6 o 32 bitowej architekturze i 1,3 GHz. Tylna kamera to Sony Exmor R/MX145 8Mpix.



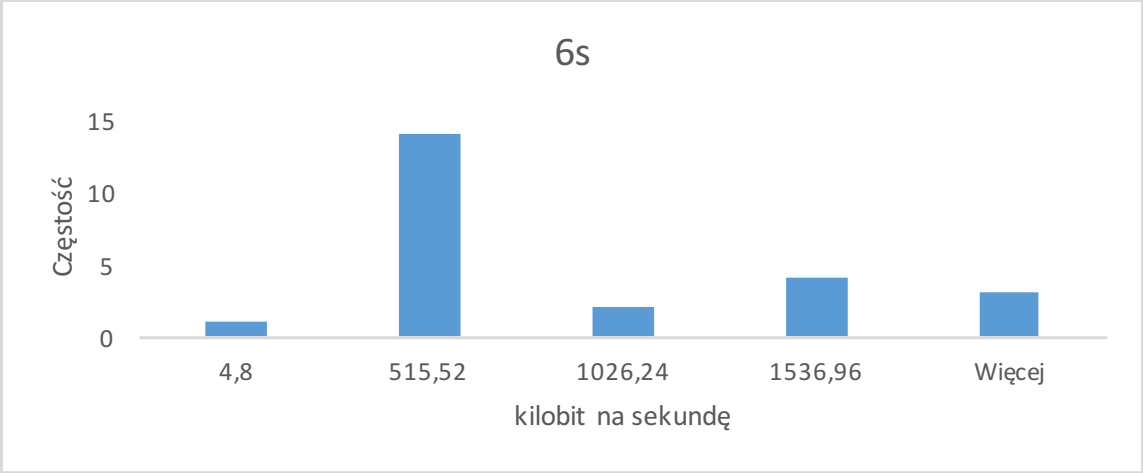
Rysunek 4.9: Prędkości czytania kodów QR na urządzeniu iphone 5s.

Iphone 5s jest w stanie odebrać maksymalnie 40 kilobajtów na sekundę ze statycznego kodu QR. Urządzenie to posiada dwurdzeniowy procesor A7 o 64 bitowej architekturze i 1,3 GHz. Tylna kamera to Sony Exmor RS 8Mpix.



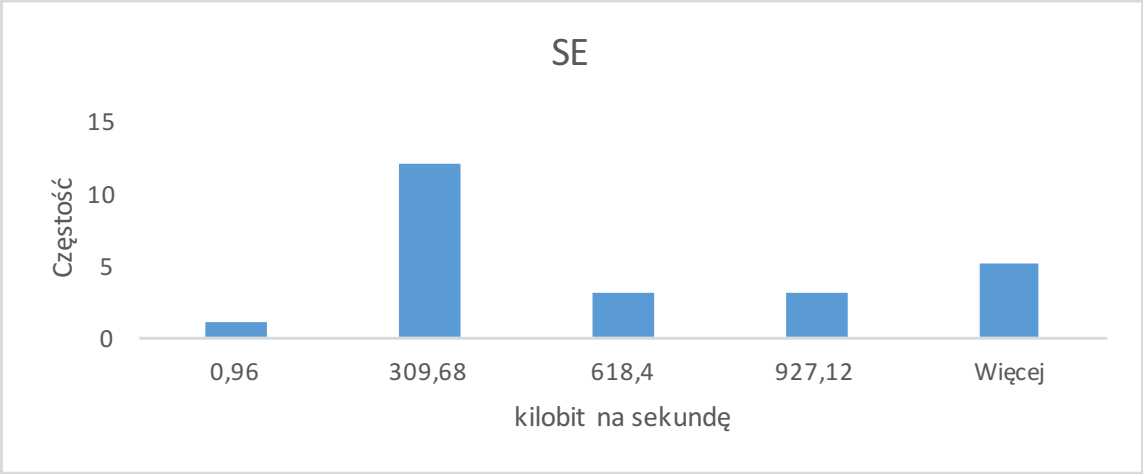
Rysunek 4.10: Prędkości czytania kodów QR na urządzeniu iphone 6.

Iphone 6 jest w stanie odebrać maksymalnie 55 kilobajtów na sekundę ze statycznego kodu QR. Urządzenie to posiada dwurdzeniowy procesor A8 o 64 bitowej architekturze i 1,4 GHz. Kamera to Sony Exmor IMX220 8Mpix.



Rysunek 4.11: Prędkości czytania kodów QR na urządzeniu iphone 6s.

Iphone 6s jest w stanie odebrać maksymalnie 60 kilobajtów na sekundę ze statycznego kodu QR. Urządzenie posiada dwurdzeniowy procesor A9 o architekturze 64 bitowej i 1,85 GHz. Kamera to Sony Exmor RS/IMX315 o 12 Mpix.



Rysunek 4.12: Prędkości czytania kodów QR na urządzeniu iphone SE.

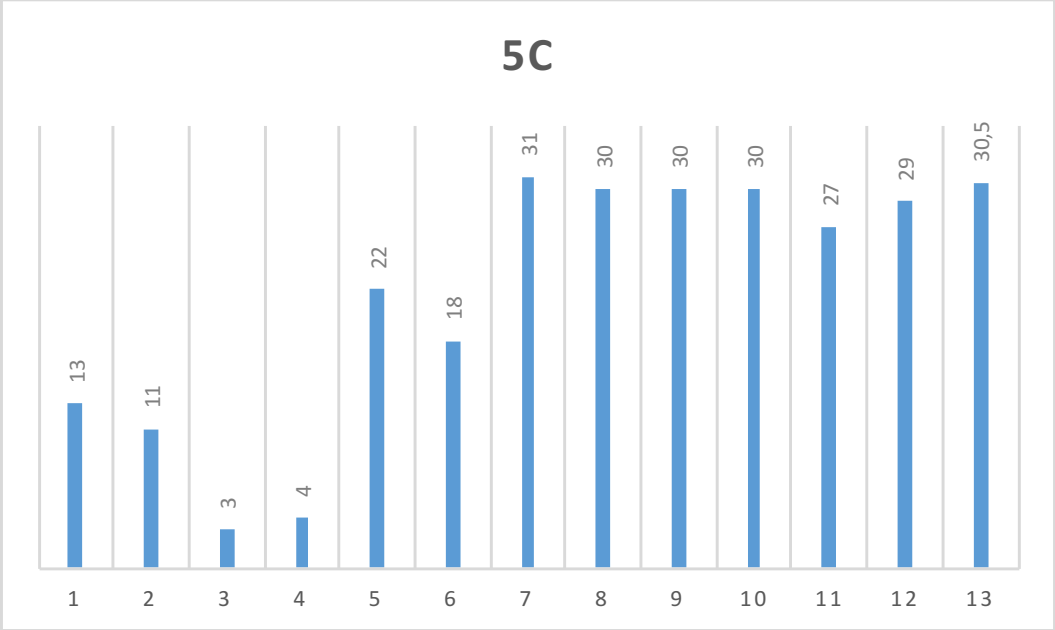
Iphone SE jest w stanie odebrać maksymalnie 60 kilobajtów na sekundę ze statycznego kodu QR. Urządzenie posiada dwurdzeniowy procesor A9 o architekturze 64 bitowej i 1,85 GHz. Kamera to Sony Exmor RS 12Mpix.

Uśredniona liczba kilobajtów na sekundę dla odbierania w czasie rzeczywistym danych z drugiego urządzenia prezentującego w sposób ciągły ten sam symbol QR:
- Urządzenie 5c - 31.0 kB/s,
- Urządzenie 5s - 34.0 kB/s,
- Urządzenie 6 - 49.0 kB/s,
- Urządzenie 6s - 58.0 kB/s,
- Urządzenie SE - 58.0 kB/s.

Powyższe wyniki potwierdzają, że urządzenia 6s oraz SE posiadają te same podzespoły, a ich kamera dla transmisji QR kodami optymalne wyniki osiąga w rozdzielczości 720p przy 240 klatkach na sekundę. Tym sposobem liczba czytanych kodów jest stała względem jednostki czasu, brak źle odczytanych QR oraz wyświetlanie klatek na urządzeniu nadającym może wzrastać do 120 klatek na sekundę. Zatem nie ograniczamy nadającego, gdyż z taką maksymalną częstotliwością możemy uzyskać obecnie na większości urządzeń (2016, 2017 rok).

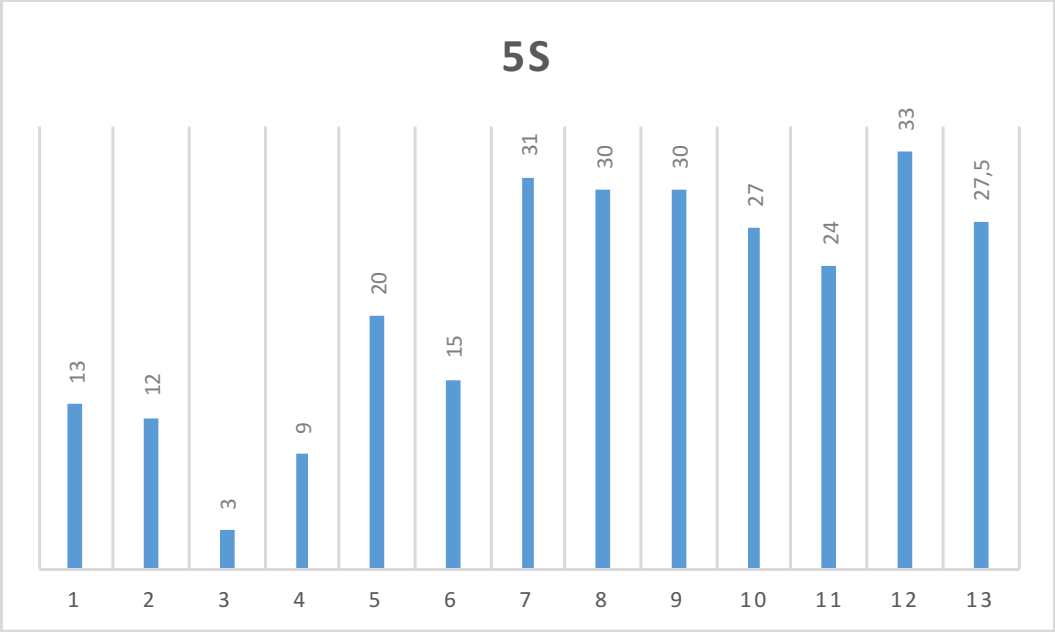
4.3 Badanie przepustowości w komunikacji

Poniższe wykresy przedstawiają maksymalną liczbę bajtów na sekundę osiągniętą w pojedynczym teście. Każdy test to jednominutowa wymiana klucza między dwoma urządzeniami tego samego typu. Dla każdego urządzenia zostało przeprowadzone od 13 do 20 testów.



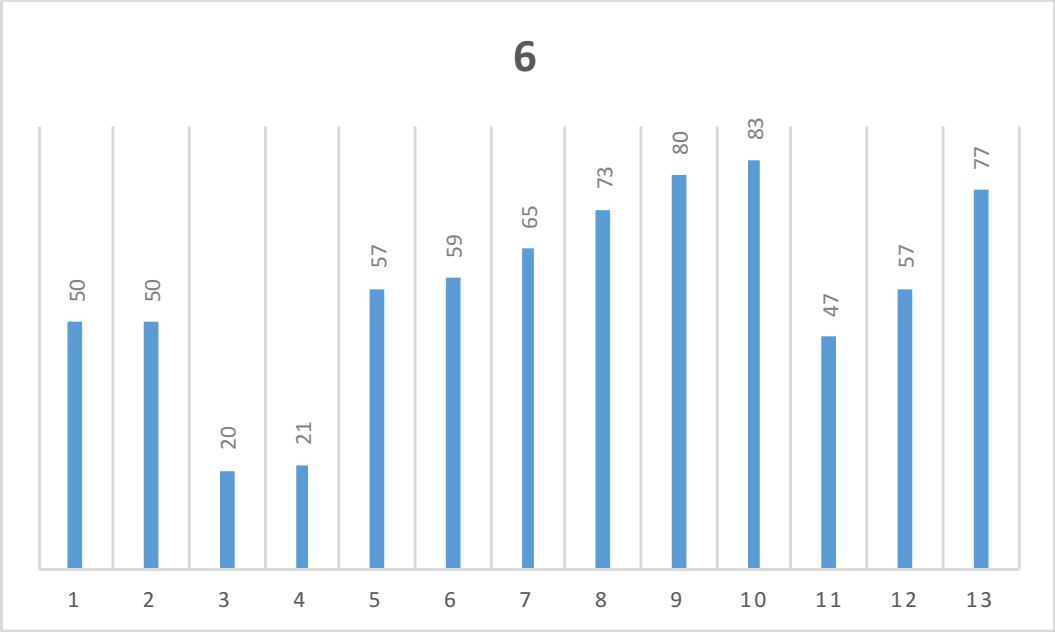
Rysunek 4.13: Test 1 do 13, względem maksymalnej liczby bajtów na sekundę. Przepustowość kanału optycznego QR kodami wyrażona w bajtach na sekundę. Czytanie QR na czterech wątkach, nadawanie w osobnej kolejce wątków, wyświetlanie dwóch krat jako potwierdzanie pominiętych QR. Urządzenie iPhone 5c.

Na wykresie przedstawiającym testy z iphone 5c można zauważyć, że po ustabilizowaniu odległości urządzeń wyniki dawały około 30 bajtów na sekundę jako maksymalny transfer.



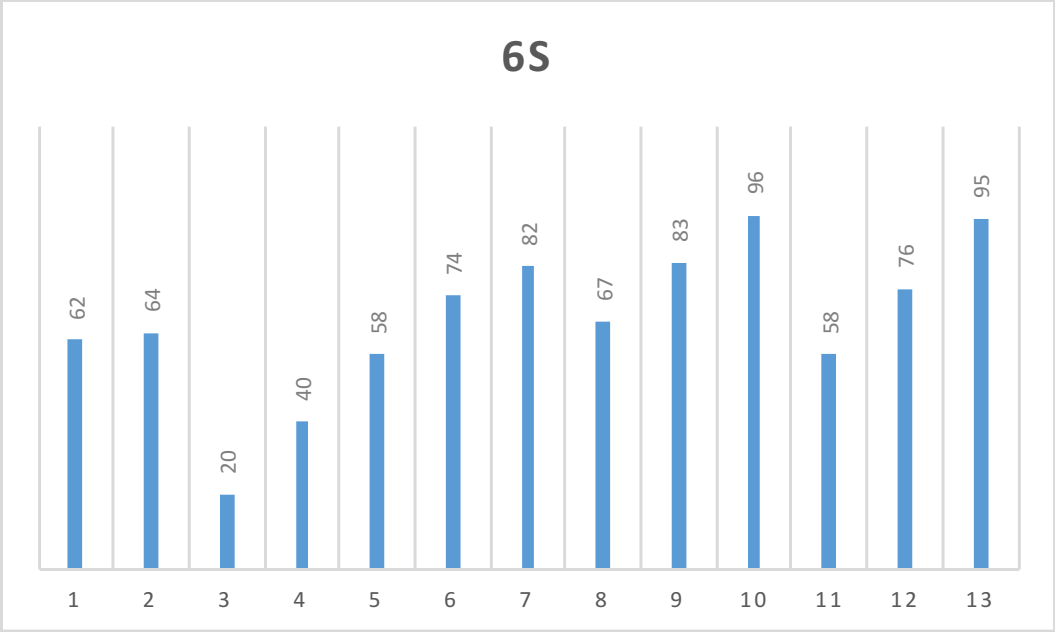
Rysunek 4.14: Test 1 do 13, względem maksymalnej liczby bajtów na sekundę. Przepustowość kanału optycznego QR kodami wyrażona w bajtach na sekundę. Czytanie QR na czterech wątkach, nadawanie w osobnej kolejce wątków, wyświetlanie dwóch krat jako potwierdzanie pominiętych QR. Urządzenie iPhone 5s.

Na wykresie przedstawiającym testy z iphone 5s również można zauważyć, że odległości między urządzeniami miały znaczenie w wymianie klucza. Wyniki oscylowały w granicach 30- 33 bajtów na sekundę, jako maksymalny transfer w kanale optycznym.



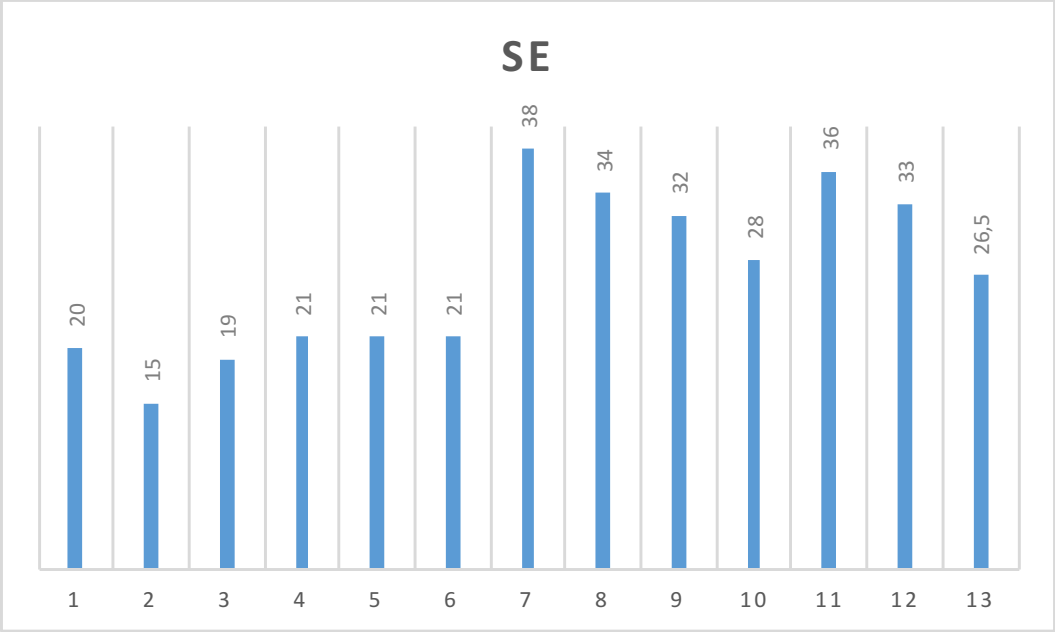
Rysunek 4.15: Test 1 do 13, względem maksymalnej liczby bajtów na sekundę. Przepustowość kanału optycznego QR kodami wyrażona w bajtach na sekundę. Czytanie QR na czterech wątkach, nadawanie w osobnej kolejce wątków, wyświetlanie dwóch krat jako potwierdzanie pominiętych QR. Urządzenie iPhone 6.

Na wykresie przedstawiającym testy z iphone 6 odległości między urządzeniami były z początku prawidłowe, jednak po próbach 3 oraz 4, kiedy technika pozycjonowania urządzeń jednego nad drugim się ustabilizowała, transfer wzrastał aż do 83 bajtów na sekundę.



Rysunek 4.16: Test 1 do 13, względem maksymalnej liczby bajtów na sekundę. Przepustowość kanału optycznego QR kodami wyrażona w bajtach na sekundę. Czytanie QR na czterech wątkach, nadawanie w osobnej kolejce wątków, wyświetlanie dwóch krat jako potwierdzanie pominiętych QR. Urządzenie iPhone 6s.

Na wykresie przedstawiającym testy z iphone 6s wyniki znacząco wzrosły. Powodem jest przednia kamera w aparacie, która posiada 5 mega pikseli oraz moze nagrywac w trybie 60 klatek na sekundę. Pozostałe urządzenia maksymalnie mogły nagrywać w 30 klatkach na sekundę przy kamerze 1.2 mega piksela. Dla tego urządzenia maksymalna przepustowość kanału wynosi 96 bajtów na sekundę, czyli prawie trzykrotność urządzeń o mniejszych wyświetlaczach.

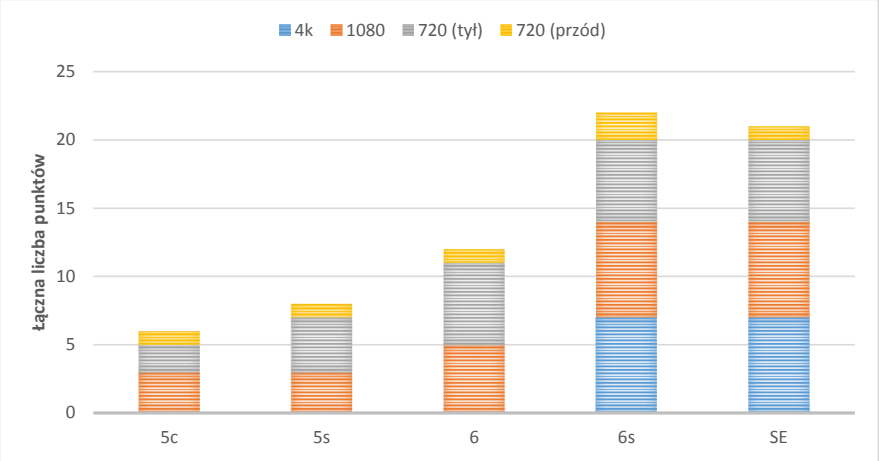


Rysunek 4.17: Test 1 do 13, względem maksymalnej liczby bajtów na sekundę. Przepustowość kanału optycznego QR kodami wyrażona w bajtach na sekundę. Czytanie QR na czterech wątkach, nadawanie w osobnej kolejce wątków, wyświetlanie dwóch krat jako potwierdzanie pominiętych QR. Urządzenie iPhone SE.

Na wykresie przedstawiającym testy z iphone SE wyniki są bardzo zróżnicowane, gdyż zostały przeprowadzone już po migracji platformy na nowszą wersję. Wyniki bardzo przypominają testy z urządzeń 5c oraz 5s, ponieważ ich wyświetlacze mają rozdzielczości 640 na 1136 pikseli. Jednak w tym przypadku testy wypadły znacznie lepiej, co może być wynikiem znaczących różnic w parametrach urządzeń. 5c oraz 5s mają procesory A6 1.3 GHz, A7 1.3GHz, natomiast SE posiada A9 1.85GHz.

Urządzenia są w stanie wymienić się danymi dowolnej długości, ograniczonej jedynie pamięcią potrzebną do zaalokowania w aplikacji. Wszystkie urządzenia mają ograniczoną pamięć RAM (5c, 5s, 6 - 1 Gigabajt; 6s, SE - 2 Gigabajt). Przez takie zróżnicowanie w badanych urządzeniach, konieczne jest czyszczenie lub ponowne używanie już niepotrzebnych komórek pamięci. Jest to ważny zabieg, gdyż zbliżając się do limitu danych jaki jest w stanie przechować urządzenie, zwiększamy ryzyko utraty innych tymczasowych danych. Urzadzenia marki Apple w odróżnieniu od innych dostępnych na rynku (2016r.) smartfonów są bardzo dobrze zoptymalizowane pod kątem dostępów do pamięci wbudowanej. Cała pamięć jest zbudowana w oparciu o technologię Flash Drive. Zapewnia ona szybszy dostep do pamięci, przez jej statyczność, która w odróżnieniu od Hard Drive, nie opóźnia transmisji danych podczas odczytu i zapisu.

Ostatecznie urządzenia, które były celem badania zostały ocenione na podstawie ich własności technicznych i można je przedstawić następująco:



Rysunek 4.18: Wykres sporządzono na podstawie ilości pikseli na sekundę w transmisji danych dla maksymalnych ustawień kamery.

4.4 Dyskusja

Podczas implementacji projekt nie przewidywał migracji między trzema wersjami języka Swift i kompilatora Clang. Wymagało to zapoznania się z nowymi wymogami platformy oraz przeprojektowaniem kolejkowania wątków i dystrybucji zadań w systemie. Zmiany te były na tyle znaczące, że zmusiły do ponownego przeprowadzenia badań, w celu ustalenia poprawności obliczeń. Sama zmiana platformy na nowszą nie ulepszyła żadnej z funkcjonalności aplikacji. Zmieniło to jedynie sposób dostępu do plików systemowych i kolejkowania wątków w całej platformie iOS. Wszelkie odwołania języka Swift do Objective-C pozostały podobne z wyróżnieniem etykietowania argumentów funkcji w sposób jawny. Kompilator wspierający język otrzymał jedną znaczącą poprawkę - nie próbuje poprawiać kodu po pierwszym dużym błędzie kompilacji, co przyspiesza kompilację większych projektów.

5 Podsumowanie

Maksymalny przepływ został osiągnięty na parze urządzeń iphone 6s. Wynosi on 60 kB/s, czyli 480 kb/s. Test przeprowadzony z odległości 30 cm, przy użyciu tylnej kamery skierowanej na wyświetlacz drugiego urządzenia. Taki wynik pozwala na swobodną wymianę pliku o długości 4096 bajtów w ułamku sekundy. Pozwala to uznać badanie za zakończone pozytywnie i stwierdzić, że kanał optyczny jest dobrą drogą transmisji danych jako kanał alternatywny.

Projektując aplikację głównym założeniem było przesłanie jak największej ilości bajtów w jak najkrótszym czasie. Badanie obejmowało pięć telefonów marki Apple oraz przedziały czasu w jakich były one wydawane, można było spodziewać się znacznego wzrostu wydajności kolejnych względem poprzednich modeli. Tymczasem platforma iOS zmieniając programistyczne implementacje poszczególnych klas udostępnionych deweloperom, pozostawiała algorytmy i optymalizacje samemu kompilatorowi clang i jego domyślnym ustawieniom. Biorąc to wszystko pod uwagę, udało się zakończyć badania sukcesem i przedstawić zależności między poszczególnymi aspektami technicznymi oraz ograniczeniami samej platformy. Aplikacja oraz jej możliwości pozwalają na przesłanie niewielkiej ilości danych, dobranych proporcjonalnie do potrzeb. Umożliwia ona wymianę kluczy alternatywnym kanałem, odpornym na interferencje i nadmiar infrastruktur. Nie wymaga uruchamiania bluetooth, czy WiFi, przez co uniemożliwia wykrycie i podsłuchanie transmisji nawet z bliska. Aplikacja umożliwia dalszy rozwój przez implementację różnych algorytmów wymiany danych czy protokołów inicjalizujących bezpieczną transmisję danych. Transfer mógłby zostać zwiększony przy wykorzystaniu wielokolorowania kodów QR (zobacz [9]).

Bibliografia

[1] Apple - dane techniczne urządzeń.
https://support.apple.com/pl˙PL/specs/iphone
Dostęp z dnia 08.01.17r.

[2] Apple - secure coding guide.
https://developer.apple.com/library/content/documentation/Security/Conceptual/SecureCodingGuide/Articles/BufferOverflows.html.

[3] Apple api reference.
https://developer.apple.com/reference
Dostęp z dnia 08.01.17r.

[4] Core frameworks - foundation.
http://druzzt.github.io/QR/PROOF/StringCapacity.html.

[5] Qr codes for security.
https://web.archive.org/web/20170108222150/http://www.csoonline.com/article/2133890/mobile-security/the-dangers-of-qr-codes-for-security.html.

[6] Qr code® essentials.
https://web.archive.org/web/20170108222545/https://www.nacs.org/LinkClick.aspx?fileticket=D1FpVAvvJuo%3D&tabid=1426&mid=4802.

[7] Qrcode.com.
https://web.archive.org/web/20130129064920/http://www.qrcode.com/en/qrfeature.html.

[8] B. A. Olshausen. Aliasing
2000. strony 1–3,
https://web.archive.org/web/20170103174530/http://redwood.berkeley.edu/bruno/npb261/aliasing.pdf.

[9] M. Querini, G. F. Italiano. Reliability and data density in high capacity color barcodes.
strony 1598–1601,
http://web.archive.org/web/20170109142152/http://www.doiserbia.nb.rs/img/doi/1820-0214/2014/1820-02141400054Q.pdf.

A Zawartość płyty CD

Na płycie znajduje się projekt Xcode, zawierający aplikację testową oraz pdf zawierający treść pracy dyplomowej.

- By skompilować i uruchomić projekt należy w programie Xcode w wersji 8.1 otworzyć plik 208836.xcodeproj. Otworzy to drzewo hierarchii w Projet Navigatorze. Następnie należy podłączyć do komputera urządzenie marki Apple z wersją iOS co najmniej 9.0. Wybrać schemat budowy: ”motywacja”, następnie wybrać urządzenie na które ma się zbudować aplikacja. Wciśnięcie command i „R” uruchomi aplikację na urządzeniu.
- Treść pracy dyplomowej w formacie pdf- plik „W11_208836_2016_praca inzynierska.pdf”.