QR-Code

Internet and Security Project
Universität Basel

Luc Kury, Jonas Rudin and Nicolai Rutz

Frühjahrsemester 2019

Inhaltsverzeichnis

---

1 Einleitung

Ziel unseres Projektes war, eine bidirektionale Chat-Applikation zu schreiben, in welcher die Nachrichten via QR-Code übermittelt werden. Dabei werden die zu sendenden Nachrichten in einen QR-Code encodiert und auf dem Bildschirm des Senders dargestellt. Der QR-Code wird dann von der Kamera des Empfängers gelesen, decodiert und wieder als Text ausgegeben. Um die Nützlichkeit solcher Datenübertragung einzuschätzen, testeten wir, wie schnell man Textdaten übermitteln kann. Dabei verglichen wir die Wiedergabegeschwindigkeit von verschiedene Grössen (Versionen) von QR-Codes, um die schnellste, verlässliche Übertragungsgeschwindikeit zu finden.

2 Einführung in QR-Codes

Ein QR-Code (Quick Response Code) ist ein zweidimensionaler Code, der von der japanischen Firma Denso Wave im Jahr 1994 entwickelt wurde. Er besteht aus einer quadratischen Matrix aus schwarzen und weissen Quadraten, die binär enkodierte Daten darstellen (vergleiche Abbildung 1). Jeder QR-Code enthält Informationen über die Version, das Datenformat und den effektiven enkodierten Datenanteil. An drei seiner Ecken befinden sich bestimmte Muster, die zur Feldbegrenzung dienen. Verschiedene Versionen des QR-Codes indizieren die entsprechenden Kantenlängen. Je höher die Version, desto grösser ist die Kantenlänge und dementsprechend sind mehr Module enthalten. Mit höheren

Versionen werden weitere Muster hinzugefügt, die zur Ausrichtung der Codes dienen.

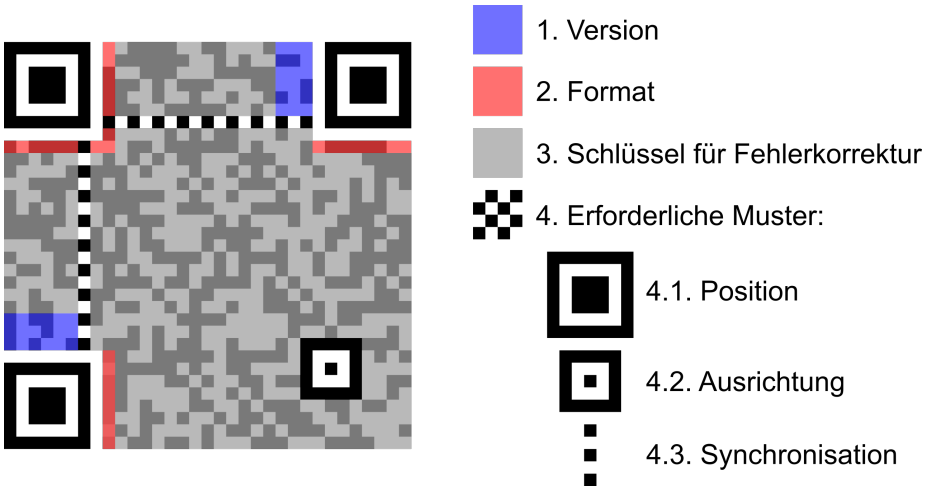

Abbildung 1: Struktur eines QR-Codes[1]

QR-Codes verfügen ausserdem noch über verschiedene Stufen von Fehlerkorrektur. Mit der tiefsten Fehlerkorrektur `LOW` können 7%, während mit der höchsten 30% der Daten wiederhergestellt werden können. Je höher die Fehlerkorrekturanteil ist, umso weniger enkodierte Daten kann ein QR-Code enthalten.

3 Anwendungsbereiche

Heutzutage werden QR-Codes in den verschiedensten Bereichen unseres Lebens verwendet. Neben ihrem ursprünglichen Gebrauch in der Produktionslogistik werden sie auch als Navigationshilfe an Haltestellen, Hilfe für den Einkaufszettel, mobile Visitenkarten und vieles mehr benutzt. Für unser Projekt war gezielt ein Nutzungsbereich von Bedeutung.

3.1 Punkt-zu-Punkt-Chat

Hauptziel unseres Projekts war es, ein Punkt-zu-Punkt-Chat zu entwickeln, der zwei Parteien über QR-Codes kommunizieren lässt. Dabei sollen Nachrichten beliebiger Länge auf der einen Seite in QR-Codes enkodiert werden und von der anderen Partei mit Hilfe einer Kamera erfasst und dann dekodiert werden, worauf die Nachricht in ihre ursprüngliche Form zusammengesetzt werden soll.

4 Software Implementation

Nach einer kurzen Recherche bezüglich vorhandener QR-Software, stiessen wir auf Python Programmbibliotheken, welche sich mit dem Enkodieren und Dekodieren von QR-Codes auseinandersetzen. Wir haben uns entschieden, `Python`[2] in

[1] `https://de.wikipedia.org/wiki/QR-Code`
[2] Python programming language `https://www.python.org/`

Version 3.6 als Implementationssprache für den QR-Chat zu nutzen, da Python als untypisierte Programmiersprache das schnelle Entwickeln von Prototypen erlaubt. Für die Enkodier- und Dekodierfunktionalität wurden jeweils eigene Klassen geschrieben. Diese verfügen jeweils über eine `run()` Methode, mit welcher die gesamte Funktionalität der Klasse aufgerufen wird. Die Klassen können entweder als Modul importiert und anschliessend instanziert werden oder alternativ verfügen beide Module über eine `if __name__ == "__main__"` Anweisung, so dass das Modul direkt von der Kommandozeile ausgeführt werden kann. Der Source-Code unserer Implementation ist frei auf `https://bitbucket.org/tairun/ias-qrcode-fs2019/` verfügbar.

4.1 Enkodierung

Für das Enkodieren der Nachrichten in QR-Codes, haben wir uns an der `qrcode`[3] Library bedient, die uns die Version, als auch den Fehlerkorrekturlevel frei wählen lässt. Damit Nachrichten beliebiger Länge gesendet werden können, müssen diese in einzelne Nachrichtenstücke zerlegt, als zusammengehörend gekennzeichnet und enkodiert werden. Die Versiongrösse bzw. Fehlerkorrekturlevel, sind massgebend für die Grösse der einzelnen Nachrichtenstücke. Enkodierte Nachrichtenstücke werden mit Hilfe der `opencv`[4] Library in wählbaren Zeitintervallen nacheinander angezeigt.

4.2 Dekodierung

Durch die `opencv` Library, die es ermöglicht, einzelne Bilder mithilfe der Kamera zu speichern, können diese mit Hilfe der `pyzbar`[5] Library dekodiert werden, falls sich darin ein QR-Code befindet. Damit Nachrichten wieder zusammengesetzt werden können, erkennt der Dekodierer den von uns definierten Präfix, der wie folgt aussieht:

$$\texttt{xx/xx/}, \text{ wobei } x \in \{0..9\}$$

Nachrichtenstücke werden so in einer Liste an ihrer entsprechenden Stelle eingespeichertt und am Ende wieder als komplette Nachricht angezeigt, wobei maximal 99 QR-Codes pro Nachricht gelesen werden können.

4.3 Benchmarking

Für die Auswertung der Resultate unserer Tests wurde die `matplotlib`[6] und `pandas`[7] Library benutzt, welche es möglich macht, unsere aufgezeichneten Daten statistisch auszuwerten und in verschiedenen Plots darzustellen. Der Enkoder und Dekoder verfügen jeweils über eine eigene Benchmark-Implementation, welche das Ausführen und Aufzeichnen von Benchmarks grösstenteils automatisiert.

[3] qrcode Library von PyPI: `https://pypi.org/project/qrcode`
[4] opencv Library von PyPI: `https://pypi.org/project/opencv-python`
[5] pyzbar Library von PyPI: `https://pypi.org/project/pyzbar`
[6] matplotlib Library von PyPI: `https://pypi.org/project/matplotlib`
[7] pandas Library von PyPI: `https://pypi.org/project/pandas`

5 Resultate

Für die Auswertung unseres Projektes testeten wir die verschiedenen Version der QR-Codes mit unterschiedlichen Anzeigezeitintervallen auf der Senderseite.

5.1 Parameter

Die folgenden Variablen waren für unsere Tests ausschlaggebend:

QR Version	2 - 10	*variabel*
Fehler Korrektur	H	*fest*
Anzahl Bytes pro Nachricht	600	*fest*
Anzahl QR-Codes	6 - 75	*variabel*
Grösse von QR-Code	400 - 484cm^2	*fest*
Zeitintervall	0.5 - 50ms	*variabel*

Der Testlauf wurde für jede QR-Code Version je fünf mal für jedes Zeitintervall getestet. Dabei wurde die verstrichene Zeit vom Eintreffen der ersten bis zur letzten Teilnachricht auf der Empängerseite gemessen. Die Nachricht bestand dabei immer aus denselben 600 Buchstaben, entnommen aus dem bekannten "Lorem ipsum"Beispieltext. Zu beachten ist ausserdem, dass die Fehler Korrektur immer `HIGH` gesetzt wurde.

5.2 Hardware Spezifikation

Für die Tests wurden zwei Studenten-Notebooks verwendet. Die wichtigen Leistungsmerkmale beschränken sich dabei auf Display und Webcam, unter der Annahme dass die CPU die benötigten Berechnungen schneller als die Refreshrate des Displays, beziehungsweise die Framerate der Webcam, ausführen kann.

Sender:	Display, 15.4-inch (2880 x 1800), 60Hz
Empfänger:	Webcam (front), 5MP (1080p), 60fps

5.3 Ergebnisse

In Abbildung 2 sieht man die Anzahl QR-Codes, welche pro Version für eine Nachricht mit 600 *bytes* versendet wurden. Die x-Achse zeigt die QR-Version, die y-Achse die Anzahl QR-Codes.

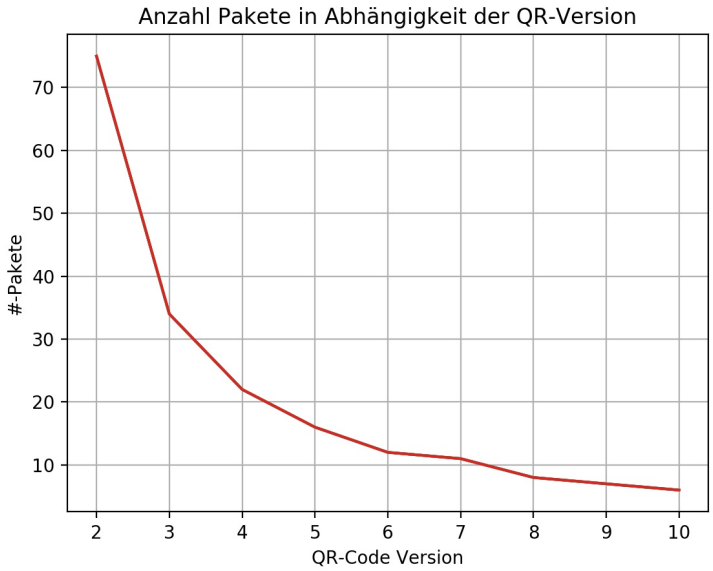

Abbildung 2: QR-Code Version vs Anzahl Pakete

Die folgende Abbildung 3 zeigt die durchschnittliche Zeit, die benötigt wurde, um die komplette Nachricht zu übertragen. Die verschieden farbigen Linien in der Abbildung zeigen die unterschiedlichen Zeitintervalle für die Anzeigedauer eines QR-Codes. Die x-Achse zeigt die QR-Version, die y-Achse die durchschnittliche Übertragungszeit der Nachricht in Millisekunden. Zu beachten ist, dass bei den Übertragungsraten 5 und 10 *ms* bei QR-Version 2 die Auswertung nicht im Plot enthalten sind, weil die Messdaten nicht repräsentativ waren.

Die Grafik in Abbildung 4 zeigt die Standardabweichung der Messdaten für jedes Zeitintervall in vier einzelnen Subplots. Die x-Achse zeigt die QR-Version, die y-Achse die Standardabweichung der 5 einzelnen Messungen jedes Zeitintervalls in Millisekunden.

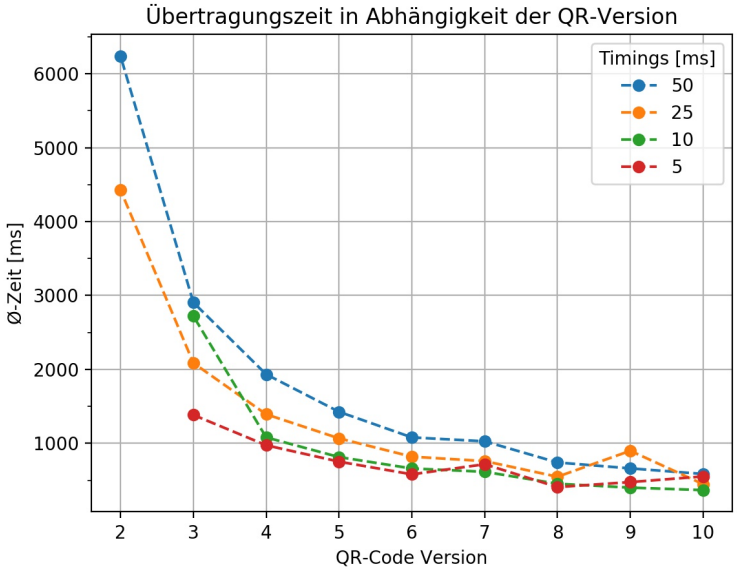

Abbildung 3: Übermittlungszeit vs QR-Version für verschiedene Zeitintervalle

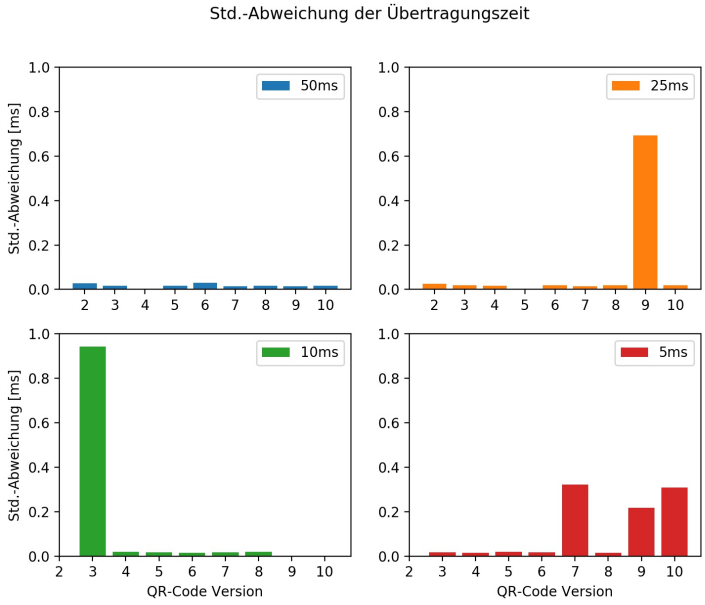

Abbildung 4: QR-Code Version vs Standardabweichung

6 Diskussion

Für die Implementation des Punkt-zu-Punkt-Chats war unser Ziel neben einer robusten Übertragung der Daten, ebenfalls eine möglichst kleine Übertragungsdauer zu erreichen. Neben unbeeiflussbaren Faktoren wie Lichtverhältnisse für die Kamera, Auslastung der CPU, usw, gibt es drei wesentliche Faktoren, welche in unserer Implementation des QR-Chats veränderbar sind. Diese sind: QR-Version, Error Correction Level und Timing. Daher können wir ableiten, dass die Übertragungsdauer eine Funktion mindestends dieser drei Faktoren ist.

$$\text{Übertragungsdauer} \sim f(\text{version}, \text{error\_correction}, \text{timing})$$

Aus unseren ersten Tests zeigte sich jedoch, dass den Fehlerkorrekturlevel `HIGH` am zuverlässigsten funktionierte (Lichtverhältnisse, verschmutzes Display, Kontrast Webcam). Somit reduziert sich die Abhängigkeit der Funktion um das Error Correction Level. Die beiden übrigen Parameter wurden jeweils in einem sinnvollen Bereich getestet. Die Resultate werden in Kapitel 6.1 erörtert.

6.1 Interpretation der Messung

Aus Abbildung 2 geht hervor, dass mit grösserer QR-Version entsprechend mehr Daten pro QR-Code übertragen werden können, weswegen für das Senden Nachrichten gleicher Grösse weniger QR-Codes benötigt werden. In unserer Testumgebung haben wir jeweils Nachrichten mit 600 *bytes* verschickt, was bei Version 2 **75** QR-Codes zum Senden benötigte. Demgegenüber wurden bei Version 10 nur **6** QR-Codes benötigt. Spannend zu beobachten ist, dass der Zusammenhang zwischen QR-Version und Speicherkapazität nicht linear ist.

In Grafik 3 sehen wir, dass mit höherer QR-Version durchschnittlich weniger Zeit für die Übertragung benötigt wird. Dabei ist zu beachten, dass ab einer gewissen QR-Version, in unserem Fall ab der Version 7, das Zeitintervall der Anzeigedauer eine verschwindende Rolle einnimmt. Ausserdem haben wir festgestellt, dass ab Version 7 höhere Versionen nur eine minimale Senkung der Übertragungsdauer erreichen. Zusammen mit der vorherigen Interpretation für Abbildung 4, kann man darauf schliessen, dass kleinere QR-Versionen eine grössere Übertragungsdauer haben, weil mehr QR-Codes dekodiert werden müssen. Weil für QR-Version 2 **75** QR-Codes versendet werden müssen, tritt bei einem Zeitintervall von 5 und 10 *ms* beim Empfang Datenverlust auf, da nicht alle QR-Codes in der kurzen Zeitspanne dekodiert werden können.

Da jede Messung für eine QR-Version/Zeitintervall-Kombination 5-mal wiederholt wurde, ist in Abbildung 3 jeweils der Mittelwert dargestellt. In Abbildung 2 haben wir die Standardabweichung der Messungen dargestellt. Aus der Grafik kann abgeleitet werden, dass die Verteilung der Abweichungen keinem Muster folgt und im Allgemeinen relativ stabil sind. Die grösste Abweichung beträgt gerundet 1 *ms*, die kürzeste Übertragungsdauer etwa 300 *ms*. Daraus folgt, dass die Abweichung im Maximum 3‰beträgt - was unerwartet stabil ist.

6.2 Weiterführende Diskussion

Der Standard für QR-Codes definiert Versionen von 1..40, jedoch wurden hier nur Versionen 2..10 analysiert. Dies hat 2 praktische Gründe: 1) Version 1 kann nur 7 *bytes* speichern. Unser Präfix benötigt aber bereits 6 *bytes*, somit bliebe nur noch 1 *byte* zur Datenübertragung. 2) Bei QR-Versionen grösser als 10, wird die generierte Bilddatei so gross, dass die Darstellung auf dem Bildschirm problematisch wird. Des Weiteren wird bei grossen QR-Codes der Processing-Delay immer grösser, so dass die Wahrscheinlichkeit, den nächsten QR-Code vom Sender zu verpassen, zunimmt. Dies wird durch die Architektur unserer Implementation begünstigt, da Lesen der Webcam und Dekodieren im gleichen Thread stattfindet. Dies könnte mit Threading und Buffering der empfangenen Bilder umgangen werden.

Zum Schluss soll noch angemerkt werden, dass trotz der überraschend gut funktionierenden Implementation die Benutzerfreundlichkeit bei der Anwendung einiges zu wünschen übrig lässt. Die beiden Notebooks müssen parallel zueinander in einem Abstand von etwa 30 *cm* stehen, damit die Webcam den gegenüberliegenden Bildschirm genügend gut erkennt. Persönlich würden wir diese Methode zum Datenaustausch nur nutzen, wenn keinerlei Netzwerk zur Verfügung steht. Selbst dann ist der Anwendungsfall sehr beschränkt, da wir mit QR-Version 10 eine ungefähre Übertragungsrate von 1.2 *kbyte/s* errechnet haben. Damit würde die Übertragung einer 5 *Megabyte* Bilddatei zum Beispiel, etwa 1 Stunde 12 Minuten dauern.