BDI on Mobile

basic data infrastructure

Topsector Logistiek

Colophon

BDI on Mobile

Authors
Nick Arema (CGI)
Sara Larsson(CGI)

© Connekt
January 2025

Summary

This document explores the possibility of integrating mobile devices in the BDI framework (www.bdinetwork.org). The BDI framework aims to facilitate flexible and controlled data exchanges in professional operational environments like logistics.

Such a framework traditionally is designed for applications on stable server environments.

The research investigates the challenges and potential solutions to integrate mobile platforms in the framework while maintaining key principles like data sovereignty, security, and performance. Mobile platforms are prevalent in operational environments such as truck-drivers on the road.

A proof of concept (PoC) was developed to test key aspects, such as connectivity, resource usage, and compatibility with mobile constraints like battery life and network stability.

The findings indicate that while there are challenges due to hardware limitations and mobile-specific requirements (e.g. app store guidelines, battery constraints and internet variability), it is feasible to use mobile devices in het BDI framework. Initial tests demonstrated promising results in terms of performance metrics like app launch time, battery consumption, and frame rate, though further testing with more complex scenarios and larger client bases is recommended.

Contents

1 Introduction

This document explores the possibility of integrating mobile devices in the BDI framework (www.bdinetwork.org). The BDI framework aims to facilitate flexible and controlled data exchanges in professional operational environments like logistics.

Typically applications using the BDI framework are designed to operate efficiently in environments with reliable and stable infrastructure, such as centralized or cloud-based servers connected via secure networks. These infrastructures ensure robust data exchange, manage access control, and provide consistent availability of resources.

However, the question arises: can this same BDI framework be applied in mobile environments? Specifically, could a mobile phone-based application be established where different clients can exchange information, all while maintaining the key principles.

Mobile platforms are prevalent in operational environments such as truck-drivers on the road.

This introduces new challenges related to network reliability, security, and scalability. This exploration will focus on whether mobile devices can effectively operate within a BDI framework ecosystem while maintaining the integrity and trustworthiness of data exchange. We will investigate the challenges and solutions necessary to achieve this.

2 Investigations

2.1 Desk research

The desk research consists of finding the differences between a server and a mobile phone, what complications come with these differences and how to solve these problems.
The following aspects are key differences between a server and a mobile phone:

Power budget

A mobile phone relies on battery power for most of the time, while a server only relies on battery power in rare emergency situations. This puts a constraint on the amount of power that can be consumed by a mobile application. Therefore, when developing the mobile app, the power consumption must be taken into consideration.

Internet connection

Servers have a stable internet connection, as they are connected via a physical cable. Additionally, the firewall rules can be configured by the manager of the internet connection. A mobile phone on the other hand is always wirelessly connected to the internet, which results in occasional dropouts due to a loss of signal. Additionally, the owner of the mobile device rarely has control over the firewall settings, as this is managed by carriers of public wifi providers.

Memory and CPU

While the memory and CPU capacity in mobile phones has increased significantly over the past years, it is generally less than what a server is generally equipped with. Additionally, a constantly high CPU usage also has an impact on the battery life.

Application distribution

Applications on mobile phones are generally installed via an Application store (App-store or Play-store). These marketplaces pose requirements on the apps that are downloadable from here. Distribution channels for applications towards servers rarely pose any requirements on the application.

Concurrency handling

A server is a lot better in concurrency handling because it is a lot more powerful compared to a mobile phone, but in this case this is not expected to be a problem because the mobile phone won't be the device running the dataspace. All it will do is sending basic messages for limited use cases.

Security

Looking at security, there is not a big difference between using a mobile phone or a server. Furthermore, no other security threats were found.

Applications using the BDI framework on mobile phones

Given the differences between servers and mobile phone applications, it is not possible to simply translate the existing BDI based implementations to a mobile application. Primarily the challenges around internet connectivity and the lack of control over the firewall settings prevent the reuse of existing implementations.

The main components of the BDI implementation as used is composed out of an API, and Apache Pulsar. A number of existing frameworks, libraries and technologies have been evaluated in order to discover if these can assist in tackling the challenges related to switching to Mobile.

Framework	Pros	Cons
Libp2p	Modular and highly flexible; built-in NAT traversal, peer discovery, and encryption; ideal for decentralized, scalable apps.	Complex setup; limited official C# support (community-driven implementations exist).
ZeroMQ	Lightweight, flexible; supports PUB-SUB and PUSH-PULL patterns; suitable for low-latency, real-time messaging.	No built-in NAT traversal or peer discovery; requires manual network handling.
WebRTC	Low latency; built-in NAT traversal (via STUN/TURN); ideal for real-time media or data transfer.	Complex setup (requires signaling servers); browser-focused but adaptable.
LiteNetLib	Lightweight, efficient, supports NAT punchthrough; optimized for low-latency game networking.	Limited features compared to other frameworks; requires manual network management.
gRPC	Strongly typed; Protocol Buffers for binary communication; supports bi-directional streaming for real-time data.	Difficult to adapt for P2P; lacks native NAT traversal.
Akka.NET	Scalable, fault-tolerant actor-based model; supports clustering and load distribution.	Heavy framework; overhead can be excessive for small-scale P2P projects.

Peer-to-Peer Frameworks

Framework	Pros	Cons
ASP.NET Core SignalR	Fully supported in .NET; simplifies realtime communication; built-in features like message broadcasting and group management.	Overkill for simple use cases; additional layers may consume more resources.
WebSockets4Net	Lightweight, easy integration into .NET MAUI apps; suitable for basic WebSocket functionality.	Lacks high-level features like SignalR; requires manual connection management.
Socket.IO (C#)	Feature-rich; ideal for event-driven apps (chats, notifications); supports fallback protocols if WebSocket unavailable.	Primarily Node.js server-side; less integrated with .NET ecosystem.
Fleck	Lightweight, simple WebSocket server for .NET; minimal overhead for basic functionality.	No advanced features like message routing or reconnection handling; infrequently updated.
SuperSocket	Modular design; supports WebSockets and other protocols; strong community support.	More setup and configuration needed compared to simpler frameworks like Fleck.
Apache Pulsar	Supports many client languages, and a iSHARE authentication plugin is available.	Initial server setup is complex.

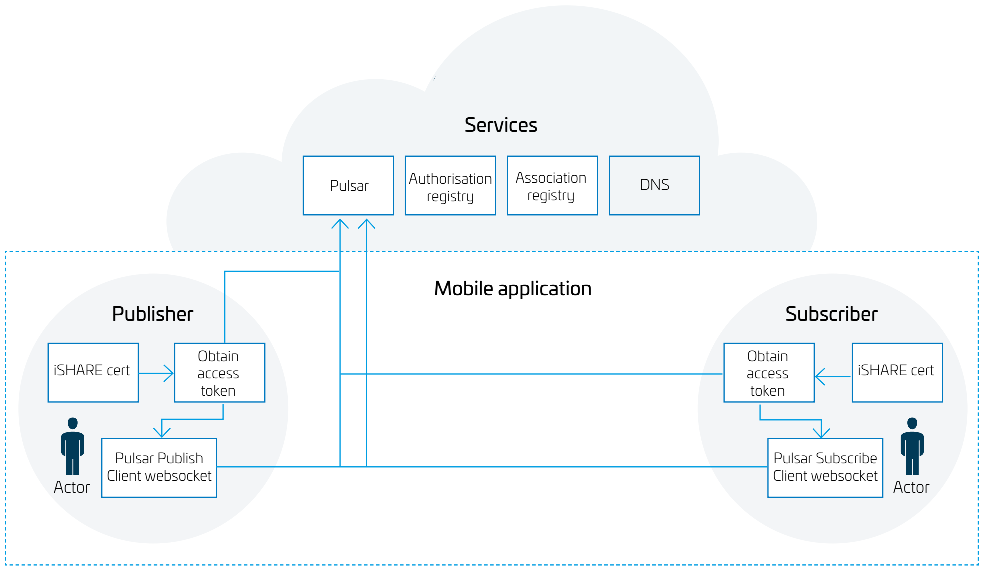
Broker based Frameworks

The desired option is one that can handle Network Address Translation (NAT), and that can cache data on server, in order to overcome temporary loss of signal.

From the framework analysis it was learned that most peer to peer frameworks rely on open ports, while a couple support mechanisms with a relay server to connect clients behind a NAT. It became also apparent that the number of actively supported client programming languages differs significantly between frameworks. The final realization is that none of the evaluated frameworks were worthwhile to use in favor of the existing BDI component with Apache Pulsar over Websockets.

2.2 Architecture

A PoC has been developed, in order to prove that technology related to BDI is possible to run on mobile. Additionally the PoC will be used as a tool to do measurements on the concerns mentioned previously (battery, performance, connectivity, etc.). For this PoC a simple mobile app with the functionality to receive and send messages is implemented.



The PoC has been developed in .NET MAUI, a Microsoft framework allowing to build multi-platform apps. Benefit is that this framework uses C# as a base, meaning some existing functionality can be reused. One negative aspect of .NET MAUI is that it might have slightly higher performance overhead and application size, but for a simple application such as this PoC it does not make a difference.

Websockets are used to set up connection to Pulsar. Websockets are used since this has already been proven to work, and there are examples which can be used as base.

Authentication is done using iSHARE, in the future it might be possible to use OAuth, but that functionality is not yet ready. To use iSHARE a token needs to be uploaded to the application, in the PoC there will be a default token, and no action is needed from the user, but this needs to be kept in mind for a full-scale project.

The PoC is not published on the app stores, but the support of .NET MAUI does make this possible, so it is something which could be done for a full-scale project.

3 Results Proof of Concept

3.1 Metrics (KPI's)

In addition to proving that the technology works on mobile, the goal is also to benchmark the application to see if it is feasible for users to use this application.

The following KPIs are commonly used, and suggestions to aim for. These will be used as comparison point for the PoC.

- App launch time: max 2 seconds (2).
- Frame rate: min 30 fps.
- Memory usage: 300MB (4).
- Battery consumption: around 5% per hour (1).
- Data usage: around 50MB per hour (3).
- Offline functionality: App can handle drops in connectivity .

Sources:

1. Energy consumption of the 30 most popular mobile apps in the world - Greenspector.
2. Get your App Performance Score | App quality | Android Developers.
3. How much data do I need? Is 1GB, 4GB, 8GB, 20GB, 50GB... enough data?
4. Do You Have the Right Amount of RAM? | Zebra Blog | Zebra.

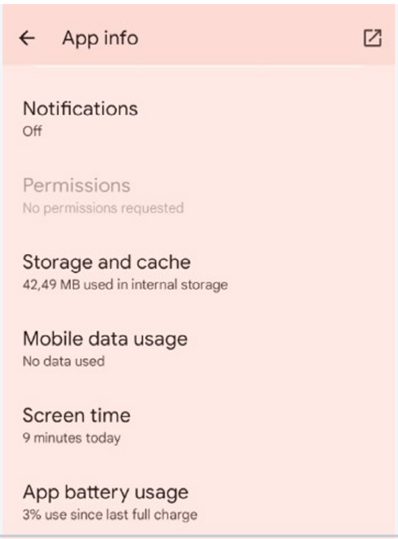
3.2 SW Development (tools)

There are multiple tools which can be used to measure the performance of an application. The PoC doesn't have full functionality (think full UI, animations, etc), therefore the measurement provides a good indication of the performance of the core functionality. We have used the following tools to analyze the performance of the PoC.

- Android performance quiz: Get your App Performance Score | App quality | Android Developers.
- Android Studio Profiler: Profile your app performance | Android Studio | Android Developers.
- Visual Studio Profiler: Measure performance in Visual Studio | Microsoft Learn.

3.3 Results

Metric	KPI	Result
App launch time	Max 2 seconds	1.6 seconds
Frame rate	Min 30 fps	60 fps
Memory usage	300MB	350MB
Battery consumption	Max 5% per hour	3% per hour
Data usage	50MB per hour	Minimal
Offline functionality	Can handle drops in connectivity	Not implemented


App info

Notifications
Off

Permissions
No permissions requested

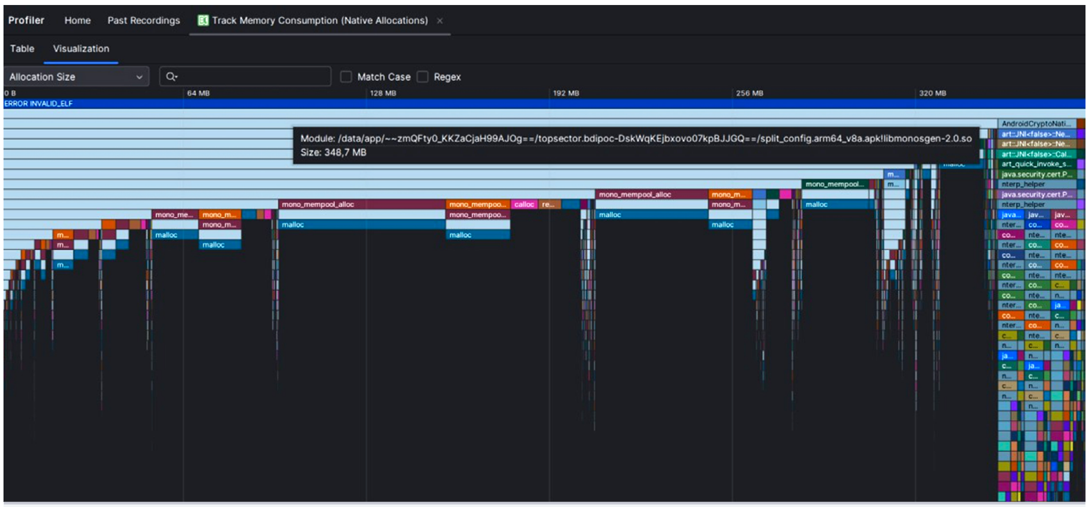
Storage and cache
42,49 MB used in internal storage

Mobile data usage
No data used

Screen time
9 minutes today

App battery usage
3% use since last full charge

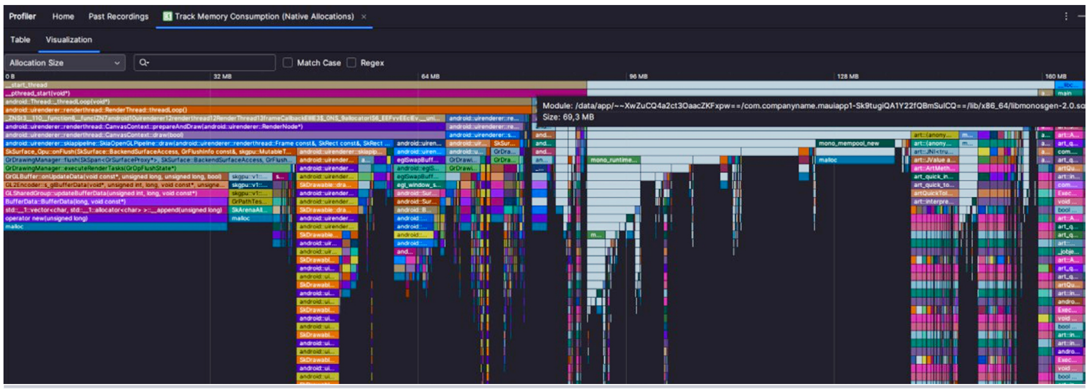
3-1: Performance values of BDI PoC on Android device



Module: /data/app/~~zmQFty0_KKZaCjaH99AJOg==/topsector.bdipoc-DskWqKEjbxovo07kpBJJGQ==/split_config.arm64_v8a.apk!libmonosgen-2.0.so
Size: 348,7 MB

3-2: Memory usage of the BDI PoC, measured in Android Studio

To further analyze memory usage the PoC has been compared to a default MAUI application, the default application has some UI elements and simple functionality. The memory usage of this default application is 160MB showing that MAUI applications not inherently have a high memory footprint.



Module: /data/app/~~XwZuCQ4a2ct3OaacZKFxpw==/com.companyname.mauiapp1-Sk9tugiQA1Y22fQBmSulCQ==/lib/x86_64/libmonosgen-2.0.so
Size: 69,3 MB

3-3: Memory usage of a default MAUI application

3.4 Observations

Overall
When transforming an application that is made for a server to a mobile phone, there are a lot of things that you should look into and really understand what the differences are. It is necessary to do a solid research before starting the development of the app, so the developers know the main points of importance to look at.

Feasibility
Looking at the results from the performance tests a BDI framework based application can function on mobile without issues, most KPI’s were better than the target value, except for memory usage which was slightly higher than preferred. There has not been any analysis on how to optimize the app, so it should be possible to decrease the memory usage by taking this into account.

5 Recommendations

Use existing SDK's

By reusing existing building blocks of Pulsar and standard client side websocket libraries only a very small amount of custom code had to be written on the client side. Due to that it is not necessary to create and offer a 'BDI on mobile SDK' for users, it is simple enough for users to implement in their own ecosystems.

Expand the application

Expanding the application will test if the KPI's will still hold while the app has more tasks to perform. This will show if the groundwork of the app is proper or if changes will fully change the outcome of the KPI's.

Test multiple clients

The app has so far been tested with only small numbers of clients connecting simultaneously. It is unclear how the system performs with tens or even hundreds of clients are connected at once. There is a need to confirm whether the connections remain stable, whether every client can still send messages to others, and whether the KPIs remain consistent under such conditions.

Authentication with OAuth 2.0

For the PoC the authentication is done with iSHARE, on mobile this is unintuitive, as the user needs to upload a certificate to their phone to be able to authenticate in the application.
Implementing something like vanilla OAuth 2.0 would create a flow which is more recognizable for the user.

BDI, Topsector Logistiek & DIL
Ezelsveldlaan 59 | 2611 RV Delft | +31 15 251 65 65
www.bdinetwork.org | www.topsectorlogistiek.nl | www.datainlogistics.org

basic data infrastructure