Best practises for 5G App Developers

Version 1.0

23 November 2022

This is a White Paper of the GSMA

Table of Contents

1. Introduction

1.1. Overview

Prior generations of mobile networks have been limited by network elements and designs that added dozens or even hundreds of milliseconds to each transaction, making delivery of 'real-time' wireless applications impossible. Emerging 5G networks, combined with Edge Computing, are poised to enable entirely new types of apps for users. These will take advantage of extremely low latency and high throughput to deliver experiences that feel 'human-time' – so responsive that they feel as natural as a human response. These may include: real-time language translation, AR-assisted guidance, intra-vehicle traffic safety, cloud-rendered gaming, and other yet-to-be-envisioned experiences that may one day become essential to daily life.

When the first wireless applications became part of daily life, developers (assisted by Operators and Platform providers) learned that a mobile network has different capabilities and requirements than a wired network. They implicitly or explicitly learned a set of 'Best Practices' for mobile development. However, just as application developers had to learn how to create for mobile networks in the early days of 4G, developers now need to learn a different bag of tricks to make the most of 5G networks.

Security guidance is not covered within this document, however developers are recommended to use best practice security measures in all their work, for which there are a number of freely available resources available from a range of organisations. There is an increasing need to ensure the security of networks and future work is intended to be published in order to support developers to take adequate measures to prevent security issues.

1.2. Document Purpose

The purpose of this document is to provide concise and specific tips to developers building latency sensitive applications for emerging 5G Networks. This document tries to provide guidance to get around some of the pitfalls that developers may encounter. The main target audience for this document is application developers, but also may be useful for Mobile Network Operators and Device Manufacturers.

The Best Practices that follow are not simply 'academic'. These guidelines were derived by GSMA member operators based on detailed interviews with actual developers. These early-phase developers described successes and challenges they faced and what they 'wished they had known' at the outset of their project. The GSMA is making this living document available to developers in order to remove barriers to development and accelerate the growth of a new generation of applications that take advantage of emerging 5G and Edge Compute capabilities.

By following these guidelines and recommendations, developers will be better equipped to create network-efficient mobile applications, benefiting all players:

Developers can be aware of and avoid common problems, saving valuable iteration cycles.

Customers will see a more responsive user experience and better battery life.

Mobile Operators will see a reduced strain on network resources, lowering congestion and increasing reliability and cost effectiveness.

This document is intended to be a 'living' document, and GSMA members and partners will provide updated suggestions and learnings based on real world experiences. Any readers with additional suggestions for the document can contact the GSMA at prd@gsma.com.

The document is for information and feedback to the GSMA or the Editor is very welcome.

1.3. Scope

This document is designed to provide guidance to all developers to encourage a better approach to creating mobile apps. Whilst it is recognised that security is a key concern when it comes to interaction with the mobile network, it is considered beyond the scope of this document.

The scope of these developer guidelines is limited to:

- General guidelines for native apps that require mobile network connectivity
- Specific guidelines for iOS and Android.

There are no plans for testing or certification based on this document.

1.4. Definitions

Term	Description
API	Application Programming Interface is a source code based specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables

1.5. Abbreviations

Term	Description
API	Application Programming Interface
APN	Access Point Name
AR	Augmented Reality
CDNs	Content distribution Networks
CI/CD	Continuous Integration and Continuous Delivery/Continuous Deployment
GSMA	Global System for Mobile Communications, originally Group Special Mobile Association
MEC	Multi-access edge computing
MNO	Mobile Network Operator
PRD	Permanent Reference Document
RAT	Radio access technology
RNC	Radio Network Controller
SDK	Software Development Kit

1.6. References

Ref	Doc Number	Title
[1]	GSMA TS.09	Battery Life Measurement and Current Consumption Technique
[2]	GSMA AA.32	GSM Association Intellectual Property Rights Regulations

2. Best Practices for Application Developers

The following set of Best Practices is offered as an initial draft. We expect that as new applications are launched taking advantage of these new capabilities, new Best Practices will be discovered, and ideally become part of this open and collaborative body of work.

2.1. Best Practices Summary

The following table summarizes and categorizes the Best Practices that follow in order show how these best practices can help developers achieve key goals:

Application Resilience: Best practices to facilitate an application to react to changing conditions while providing the best possible experience.

User Experience: Best practices for designing application behaviors to best fit human needs.

Optimize Resources: Best practice to facilitate an application to use network and energy resources in the most efficient manner.

Data Resilience: Best practice to ensure data continuity in varied conditions.

(Future versions of this document posted online may include keyword tags associated with each Best Practice to make indexing and cross-referencing easier in the future.)

Best Practice Category	Application Resiliency	User Experience	Optimize Resources	Data Resilience
Know the Network	X		X	
Know The Application	X	X		
Expect Latency Gaps	X	X		
Expect throughput fluctuations	X	X		
Look out for Upstream congestion	X	X		
Push, Don't Poll!			X	

Keep Cool – Avoid Thermal Mitigation			X	
Have a good fallback source of data	X			X
Plan revenue technologies from the outset		X	X	
Crash Gently	X	X	X	
Application portability and deployment			X	X

2.2. BEST PRACTICE - Know Thy Network

2.2.1. Introduction

Each generation of mobile network has material differences in capabilities. For your app to behave optimally, a key starting point is knowing the network Radio Access Technology (RAT) the device is connected to and its capabilities at any given moment. For example, if your user is in an extremely remote area, or even on an older network such as 3G, attempting to deliver an experience requiring a robust network may frustrate the user.

Other aspects like ultra low latency or very high throughput are features of a 5G network, and will by far exceed other RATs like 4G or 3G. Thus, if your app needs any of those features, you must be sure your device is under 5G coverage with sufficient available resources.

2.2.2. Background Context and Rationale

As a developer, it is critical to keep in mind the physical realities of the mobile network delivery chain. In addition to the radio connection between your user's device and the base station, there are a number of network segments that packets will need to traverse to communicate with your application servers. This includes the Radio Access Network, Backhaul Transport, Mobility Cores, Internet Links and Application Servers (and back again). While traffic usually navigates these segments along a 'happy path', there are corner cases where certain segments may experience unexpected delays.

Additionally, because the Radio Access Network is just one of the segments, users may see fantastic coverage (5 bars, etc), but have a poor experience due to problems in an underlying segment.This means that as a developer, you should expect that your application may have a very different set of conditions over time, and that this may be confusing to your users in the field.

2.2.3. Issue Details

This variability is simply a reality of even the best mobile networks. Regardless of how technologies advance, physical realities (eg walls, trees, user's distance from mast, etc) will always prevent you as a developer from assuming perfect coverage.

It is critical that your applications be constantly measuring the network and modifying the application experience accordingly. Adaptive Bitrate Streaming is a common technology that addresses this issue on video applications, and a core component of these applications is the measurement logic that estimates throughput during use.

2.2.4. Recommendations

For emerging applications on 5G networks, at the current time, it will likely be a best practice to build similar logic into your application. While it is possible to determine the network (4G vs 5G) via an API call on the device, even with that knowledge, there may be wide variations that could impact your application.

Therefore, we suggest building in measurement logic that can actively calculate the bandwidth your users are getting as they use the app, as well as latency calculations for apps that require low latency.

NOTE – be cautious to not perform arbitrary speed tests that generate excess traffic for minimal value – it is better to test using files that are required to support the user experience.

If your user is not in an area that supports your required bandwidth or latency, modify your app behavior rather than default to that experience. As mentioned in another Best Practice, consider offering the advanced experience as a 'Bonus' option or 'Beta', and continually monitor for the desired levels.

A key point is that this requires that as a developer you clearly have tested your application under varying levels and understand exactly what is required (see next Best Practice).

2.2.5. Pseudo-Code/Implementation

There are a few practical steps here:

- Check RAT (Android):
  - Android 11 and higher:
    - https://developer.android.com/about/versions/11/features/5g
  - iOS:
    - https://developer.apple.com/documentation/coretelephony/cttelephonynetworkinfo/3024510-servicecurrentradioaccesstechnol
    - https://developer.apple.com/documentation/coretelephony/cttelephonynetworkinfo/radio_access_technology_constants
- Psuedo-code to Build a Throughput Calculator
  - As part of app, designate a particular file of known size to act as a reference object.
  - Record time stamps each time this file is requested, when download starts, and when download is completed.
  - Divide download time by object size to estimate throughput speed. This can provide a general idea of the throughput available to your application over time

- Psuedo-code to Build a Latency Calculator
    - Use the same reference object as above.
    - In addition to recording time stamps of file request, download start, and download completed, record the time the server receives the request.
    - Compare server download start time to Time to First Byte for the object on the client.
    - This can provide a general idea of the latency available to your application over time. Note that this latency includes the Internet latency as well as any mobile network latency.
- NOTE – in certain cases (rapid user mobility, congested urban environments) the available bandwidth over the RAT can vary by an order of magnitude within a second. Consider re-checking throughput and latency periodically for longer application sessions (videos, games etc.)

2.3. BEST PRACTICE – Know Thy Application

2.3.1. Introduction

As mentioned above, it is critical to understand the capabilities of the network your application is running on. However, it is just as important to understand exactly what your application demands from the network.

2.3.2. Background Context and Rationale

Understanding in detail the proper operating range for your application is essential to make sure that customers consistently have a satisfactory experience. Much like the idea behind Adaptive Bit Rate Video, in most cases it is better to offer a limited experience than an experience that appears broken to the user. However, if your application truly requires very high throughput and ultra low latency to deliver any value to the user, it is critical that you know these functional limits and design the user experience accordingly.

2.3.3. Issue Details

Given that most developers are typically working in areas with good mobile coverage, the key issue here is creating an appropriate test bed to allow easy testing across a wider range of conditions.

2.3.4. Recommendations

As part of setting up your CI/CD process, be sure to plan to build out a mechanism to test your application across a variety of network conditions. As early as possible in your project design and development cycle, design an approach to how your application will respond to varying network conditions – do you automatically adapt and offer a 'low-res' version of your application when network resources are limited? Do you only enable access to the experience when the network is at a certain level? It is critical to understand the functional breakpoints of your app, and create an ongoing test plan that exercises these limits at each test iteraton throughout your product lifecycle.

2.3.5. Pseudo-Code/Implementation

Test your application under varying conditions to understand requirements:

- Use a Network Attenuation mechanism to vary throughput and/or latency. These tools include:
  - Apple Network Link Conditioner
  - PCAPDroid
  - Charles Proxy
  - AT&T Video Optimizer (Network Attenuator feature)
  - ns-3 5G Network Simulators
- Conduct tests to determine the required range for your application performance.
- Implement as part of the CI/CD process

2.4. BEST PRACTICE – Expect Latency Gaps

2.4.1. Introduction

Ultra-low latency mobile networking is a key characteristic of 5G networks, and this capability is likely to enable previously unimaginable mobile applications.

However, as these networks rollout and mature, developers should expect and plan for inconsistent conditions – both jitter and packet loss.

2.4.2. Background Context and Rationale

As 5G networks roll out, coverage may not be ubiquitous, so you can expect that users may move between 4G and 5G networks – even while using your application. Additionally, for various reasons related to routing and location of your edge servers, network latency and throughput may vary or experience gaps.

2.4.3. Issue Details

The specific behaviour that developers should expect is both wide variation in latency that could vary by 10's or even 100's of milliseconds periodically. It is CRITICAL to design and test your application to function appropriately across these varying conditions.

2.4.4. Recommendation

This Best Practice has two specific recommendations:

2.4.4.1. Have Realistic Expectations for Your Application

From the initial concept of your application, design with the understanding that latency may vary. Do not design a self-driving car application that will be unable to spot an obstacle and stop safely if network latency unexpectedly increases.

Build a resilient and intuitive experience into the core of your application for users. Your application WILL fail – it is up to you to design it to fail elegantly.

2.4.4.2. Test Your Application Against Varying Conditions

Simply testing your application at your workstation is not sufficient. It is critical to use a network emulation tool that can simulate varying network conditions – both variable throughput and varying latency. Your application must be hardened and tolerant of a world that is still under construction. Be sure by implementing a highly rigorous testing regime.

2.4.5. Pseudo-Code/Implementation

In order to implement this Best Practice, it is critical to set up a test bed for your application that allows you to observe your application over time as latency varies. Ideally, you should use a tool that enables you to configure a profile that sets latency at specific levels over time.

Regardless of which tool you use, it is critical to at least document the specific latency-over-time patterns that you have used so that you can go back and repeat the same tests over time as you tune your app.

Tools that offer latency variability include:

- AT&T Video Optimizer (Network Attenuation Feature w/variable latency profile)
- Apple's Network Link Conditioner

2.5. BEST PRACTICE – Expect Throughtput Fluctuations

2.5.1. Introduction

Higher throughput is another key characteristic of 5G networks, and opens the door to a wide range of new or improved services and applications.

However, just as in the Latency variation described above, available throughput for an application might vary significantly during use due to the complexity of these many segments.

2.5.2. Background Context and Rationale

Network throughput is the amount of data that can be transferred from a server to the device that is executing the application in a given time frame. The throughput can be affected by different factors described above, such as the number of network elements the information has to cross or other physical factors, components' available, processing power in that specific moment, or user and application behaviour. Thus, the maximum throughput can suffer fluctuations.

2.5.3. Issue Details

The specific behaviour that developers should expect is that throughput can vary in a determined range, although it should not typically fail under an established limit or overpass the higher one. It is critical to design and test your application to function appropriately across these varying conditions.

2.5.4. Recommendation

Design your app so that it is robust to bandwidth variations and can work well over a range of real world network bandwidths.

2.5.5. Pseudo-Code/Implementation

Testing the application against different limits will help to stablish the thresholds or the ranks in which you can ensure the application will run optimized. It is strongly recommended to test against test harneses that can vary the available throughput to the devices.

Tools that offer throughput variability include:

- AT&T Video Optimizer (Network Attenuation Feature w/variable latency profile)
- Charles Proxy
- Apple's Network Link Conditioner

2.6. BEST PRACTICE – Let the User Opt-In to Advanced Experiences

2.6.1. Introduction

Given that the laws of physics will always introduce some variability into the network capabilities available to a user at any given moment, developers have the power to design their applications to provide the best experience for these conditions.

2.6.2. Background Context and Rationale

Human emotions are not straightforward. Game theorists understand that most people have a much more negative emotional reaction to losing something that they possess than to the idea of gaining something they do not have.

2.6.3. Issue Details

If an application assumes the best possible network conditions from the outset, it can set certain expectations to users for an 'advanced' experience. If the application then falls back to a more standard experience, the user will be much more negatively impacted emotionally.

2.6.4. Recommendation

When designing an application that can take advantage of advanced network features, it is much better to begin with a standard experience and position these advanced features as an optional or even 'trial' experience. This will make users receiving the experience feel that they have something 'extra', without severely disappointing users that may not be in an area with adequate network resources.

2.6.5. Pseudo-Code/Implementation

By following the guidance in the Best Practices above and actively monitoring the resources available to the application, you should offer these advanced experiences as an option when it looks like the network can support those features.

If the network conditions do not support the features, either hide the feature entirely, or 'grey out' the feature so that the user knows they can attempt again when they have moved to different location.

2.7. BEST PRACTICE - Look out for Upstream congestion

2.7.1. Introduction

Most mobile networks have historically been designed to allocate more resources to carry download vs upload traffic. As a result, uploads may be more unpredictable, so be sure that your application can provide an acceptable experience with a range of upload speeds.

2.7.2. Background Context and Rationale

Especially when applications are used to the performance of fixed networks the experience of mobile networks can be quite challenging. While upload/download speeds improve with every generation, in rural locations or within buildings, coverage can be poor and cellular speed vary massively or even break up regularly.

An application with constant synchronization should consider these obstacles and find strategies to adjust network usage based on those variables.

2.7.3. Issue details

Historically Networks and User Equipment were always based on improving download speeds for the UEs. Development concentrated on factors like Antenna-Design, Receivers and Baseband functionalitites, download speed was always more important. Ratios for mobile systems are often split DL:UL = 4:1 which clearly show the historic value distribution.

Private Networks and Network Slicing might offer solutions in the future but that may not offer an immediate solution.

2.7.4. Recommendations

If your application does not rely on constant upload, e.g. for live streaming or live multiplayer games, consider delaying uploads until the user is back in Wi-Fi or their Home-Network.

This could also reduce costs for users, since an upload over cellular that could be done at a later state could save money based on the users data contract.

Note that at no point should an application constantly ping/probe a network or bombard a network with requests in order to force an upload.

2.8. BEST PRACTICE - Push, Don’t Poll!

2.8.1. Introduction

Most users have many applications installed, and many of them may be performing activities periodically in the background. If we consider two identical Android devices, one without applications (factory configuration) and other loaded with the most successful applications from the market, after one hour without end user interaction (Standby mode), the measurements show very different effect on signalling and byte consumption. The device with normal apps (Facebook, Twitter, Whatsapp, Weather, News, Traffic, etc.) generates 5500% more RNC signalling and 2800% more IP data than a terminal just restored from factory values and a Google account configured.

As a developer, it is important to keep in mind this level of activity, and try to manage background communications efficiently.

2.8.2. Background Context and Rationale

A feature that could help with your network optimisation is limiting the signalling or information exchanged to establish and manage your connections. Some of the most popular smartphone applications are also some of the greatest generators of signalling traffic, including social media, music streaming and e-mail. Extra signalling traffic or frequent “keep alive” messages overwhelm the resources of central network elements, and this is not good for your application either, as large numbers of your application users may be in a concentrated area and unwittingly create a negative experience for everyone.

Finally, because the radio is often the biggest power drain on a device, all of this constant polling can have a material impact on the expected life of the device battery.

2.8.3. Issue Details

The root of the problem is that sometimes your device needs to know when some state may change that is stored on the server, from another user, or that is coming from somewere other than the device.

The practice of 'long-polling' was common for many years as a way to periodically keep the device updated. (A metaphor here is a child in the backseat constantly asking 'Are we there yet?' for a long drive.) For both mobile networks and your application server infrastructure, this is generally no longer the best solution.

2.8.4. Recommendations

Instead of long-polling, we recommend that where possible you push to mobile applications using common practices such as Publish-Subscribe (or pub-sub). This general practice will allow your application to register for a pertinent update and get updates only if and when needed.

2.8.5. Pseudo-Code/Implementation

Many resources and implementations for pub sub exist, but here are a few general places to get started:

- https://cloud.google.com/pubsub/docs/overview#:~:text=Pub%2FSub%20enables%20you%20to,remote%20procedure%20calls%20(RPCs).

2.9. BEST PRACTICE – Have a good fallback source of data

2.9.1. Introduction

Caching is a common technique that aims to improve the performance and scalability of a system. It does this by temporarily copying frequently used data to a fast storage that is located near the application. If this fast data storage is located closer to the application than the original source (e.g. in MEC servers), caching can significantly improve the response times of client applications by serving data faster.

In the world of cloud computing, the original data typically resides on the public cloud, and the devices or edge servers can have a copy of that data. Maintaining data consistency across such distributed data stores can be a significant challenge.

2.9.2. Background Context and Rationale

As a developer, it is critical to design carefully which pieces of your application data should be stored in MEC servers and which should be stored instead in a public cloud or devices.

The most basic type of cache is an in-memory store. This type of cache is quick to access, and it can store modest amounts of static data, since the size of a cache is typically constrained by the amount of memory available on the machine hosting the application. If your application needs to cache more information than is physically possible in memory, the application can write cached data to the local file system. This will be slower to access than data held in memory but should still be faster than retrieving data from public cloud servers.

Caching can dramatically improve your application performance, scalability, and availability. The more information your application has in cache and the more users need to access that data, the greater the benefit of caching. This is because caching reduces the latency and conflict associated with processing large concurrent requests in the original data storage.

2.9.3. Issue Details

2.9.4. Recommendations

While specific recommendations vary depending on the tech stack that you use for your application, this Best Practice has a few general recommendations:

2.9.4.1. Use cache for low latency or high-bandwidth scenarios

As a developer, you should use edge computing architecture to exploit the speed of 5G and unlock key low-latency and high-throughput scenarios. Hosting storage or application logic at the edge servers or devices can make your application more responsive and ease network bandwidth requirements. Edge computing architecture avoids the latency that would result from application traffic traversing multiple hops across the Internet to reach its destination, and that allows end users to take full advantage of the latency and bandwidth benefits offered by modern 5G networks.

2.9.4.2. Be prepared that the edge cache is not always available

Be careful not to introduce strong critical dependencies on the availability of an edge cache service into your solutions. An application should be able to continue functioning if the MEC servers that provides the cache service have changed or are unavailable. The application should not become unresponsive or fail while waiting for the cache service to resume.

In a distributed environment, the inability to complete an operation is often due to some type of temporary error. If such a failure occurs, an application might assume that the situation is temporary and simply attempt to repeat the step that failed. If, despite repetition, access to the function or cache on the edge server is not successful, the application should try to connect to nearby edge servers or public cloud, thus reducing or eliminating any outage.

2.9.4.3. Tell the User if the data is not up to date

The only thing worse than stale data is when a user is not aware they are looking at stale data. If the data source that is rendering data from the user is not as current as desired, consider adding an icon or some communication to your application. For example, if the data has not been synchronized, consider adding an sync icon with a strikethough so that users are aware that the data may not be current.

2.10. BEST PRACTICE – Keep Cool – Avoid Thermal Mitigation

2.10.1. Introduction

5G radios and applications can generate significant heat when running intensive processes. Testing your application under stress conditions, while monitoring in the field can prevent thermal mitigation and application failures.

2.10.2. Background Context and Rationale

5G radios have been reported to generate heat in some device models. Additionally, computationally intensive tasks, such as running AR Toolkit, can create high temperatures and send a device into 'Thermal Mitigation' and interrupt normal device functioning.

2.10.3. Issue Details

Running complex applications on some devices connected to 5G networks has proven challenging to operate because of the devices regularly going into thermal mitigation after 10-15 minutes of continued use.

2.10.4. Recommendation

In order to prevent this issue, developers should test their application in challenging circumstances while connected to tools that actively monitor the CPU temperature of the device.

If there are specific functions in the application that generate heat spikes, developers should use this process to iteratively modify their code or the application functionality until these spikes have been removed from the test runs.

2.10.5. Pseudo-Code/Implementation

There are APIs within Android that can provide the CPU temperature, and one method to test would be to instrument a call to that API and display on the screen for a test build of their application.

Additionally, there are other tools, such as AT&T Video Optimizer, that record application traffic and key device metrics along with a screen recording of user behaviour over the course of the application test. This can make it easier to go back and analyze the specific conditions leading up to the heat spike or thermal mitigation.

2.11. BEST PRACTICE - Application portability and deployment

2.11.1. Introduction

Modern application development frameworks allow building complex mobile applications with extensive flexibility both in functionality and deployment. More recently the industry started adopting DevOps practices such as CI/CD (Continuous integration/Continuous Delivery&Deployment). The delivery pipeline allows for automating the workflow and takes advantage of application containerization for packaging and runtime environment deployment.

While, server-side component topology can be backed into the application or inferred by the logic in the app, this coupling comes with scalability and reliability limitations. This is more relevant when the app is expected to take advantage of Edge Computing.

2.11.2. Background Context and Rationale

In the last decade, the use of the Internet, cloud and the ubiquity of the web as an application platform created an unprecedented increase in traffic due in part to the development of new web standards and innovations in the mobile technologies. In addition to that we see a tremendous growth in IoT applications, particularly in the automotive space.

With this growth in use, there's an increased amount of data being sent over multi-hop public links. The industry has been using a client-server model for applications in which the client resides typically on an end-user device and the server is hosted in a large data center using either private or public cloud infrastructure.

Modern distributed architectures involve sophisticated caching schemes that require fetching various objects (images, libraries, etc.) from various locations in the path to avoid latency and improve the overall user experience while reducing bandwidth use.

More recently, application developers started taking advantage of enhanced, more distributed architectures pushing the server-side components closer to the user. Edge Computing extends the client server architecture by introducing an optimally-located server endpoint with its own deployment characteristics. These concepts are not new and they have taken different shapes in the past. Split browser architectures have pushed some of the processing from the client to the cloud and CDNs (Content distribution Networks) are caching content on behalf of content owners.

Edge Computing is also both an architectural and a deployment paradigm shift as applications are edge-aware and edge-location agnostic. On one hand, from a deployment perspective it is strictly tied to the network and data center topology. From an application developer perspective, edge is a client and/or a server.

2.11.3. Issue details

A client-side application residing on a mobile device will almost always have a server-side component dealing with user management, persistent application data repository and business logic.

When the user launches the app on the device, the app will attempt to connect to the server-side module. The application developer has many options in how to convey the network information for that server including DNS or proprietary registries.

If the application does not provide flexibility in the way the server-side app is discovered, there are multiple risks:

The preconfigured server address could be unreachable due to faults on the network path or failures in the specific data center.

If the app assumed a certain topology such as an edge deployment based on the user location, the closest geographic node might not always be the best from a performance point of view.

The client-side application being on a mobile device, will most likely have unpredictable connectivity conditions as it transitions from one network technology to another (4G to 5G, WiFi to 4G, 5G to WiFi, etc.) As a result, the app unaware of the network routing policies could provide limited optimization for user experience.

2.11.4. Recommendations

Developers, especially when expected to take advantage of the edge architecture, should design the app so it is edge-aware but location or network topology agnostic.

In practice it is recommended that:

- The client-side uses flexible discovery mechanisms for the server-side component
- The selection criteria for the match should be built on application's own non-functional requirements for bandwidth, latency, and resources.
- The server-side application should be developed with data replication/synchronization in mind
- The server-side application is packaged and configured to be deployed in a portable manner irrespective of the data-center location of the network node.

2.12. BEST PRACTICE – Plan revenue strategy from the outset

2.12.1. Introduction

Sometimes even the best application ideas can leave out a critical component – how does the developer get paid?. Thinking about how revenue strategies in advance can prevent a poor user experience down the road.

2.12.2. Background Context and Rationale

Early stage applications are often launched as free to the user. This can be especially common for emerging areas that do not yet have widescale adoption, such as AR, VR, Cloud Gaming, etc. However, if the revenue strategy is not well thought out from the very outset, the eventual implementation can be clunky and lead to rejection by users. Additionally, some revenue strategies may require a completely different technical approach, so it is critical to have a clear and realistic technical plan for future revenue streams from the outset.

2.12.3. Issue Details

The specific sorts of problems that can creep up are things like requiring that users create an account after a long period of simple and easy access with no login, or inserting a complex handoff to a paywall service that introduces a major obstacle to the user flow.

2.12.4. Recommendation

In order to avoid this issue, developers should pre-plan and build in elements of the eventual revenue framework from the outset. This may include creating a user account, and providing some incremental value to the user with that feature, such as storing user history or personalization, etc. Adding a subscription to this user account will be a simpler and smoother transition than suddenly forcing the user to create an account AND provide payment.

2.13. BEST PRACTICE - Crash Gently

2.13.1. Introduction

Despite your best efforts, applications can crash. When it does, be certain that it happens as gracefully as possible. This involves both user experience, such as showing a friendly message to users, and connection management, so the app does not continuously retry to connect to the network and cause a bigger problem impacting multiple users.

2.13.2. Background Context and Rationale

A specific situation like a network registration rejection, a network or a device failure, a sudden lost of signal, an unexpected bug, sudden loss of data, etc., could be the reason for an application to crash. In any case, it is important to manage user expectations and to ensure that your application recovers as gracefully as possible.

2.13.3. Issue Details

What it is important in this situation is to warn the user and to implement a good "retry policy". This can ensure that all the devices affected by the same app crash are not going to reattempt to connect to the network without control and all together at the same time, which can create issues for the network they are trying to reattach to. If your application is installed in many devices that happen to be in the same area (this possibility increases with the success of your app) and they crash at the same time, they will try to reconnect also at the same time. If the retry policy is not a controlled one, the network servers might not be able to process all the devices' requests at the same time, and this might generate a bigger network failure in that area(this has happened historically).

2.13.4. Recommendation

There are two recommendations here:

- One, for the final customer: to be aware of your app crash, might a friendly pop-up message like "*Something went wrong. Please try again*" or something similar, could help. Or even requesting the customer to allow a message with the failure to be sent to you, "*to improve the app*".
- The second is to prevent the network from failing in an area, by implementing a good retry policy, like for example, "*only 3 retries, being one at T0+30", the second at T0+1'15" and the third at T0+2'15"*". This example is a good policy, that controls both when the reattemps happen, and the total number of reattempts allowed.

Annex A Document Management

Annex A.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	23/11/2022	New version of GSMA Best Practice for 5G App Developers	TG	Ed Lambert, AT&T

Other Information

Type	Description
Document Owner	Internet Group
Editors / Company	Ed Lambert, AT&T Dan Druta, AT&T Esther Arellano, Telefonica Tomi Sarajisto, Telia Kay Fritz, Vodafone

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.