Demystifying Application Performance Management Libraries for Android

ASE'19;
IEEE TSE'21

Yutian Tang

Outline

- Motivation;
- Application Performance Management Library for Android Apps;
- Functionalities;
- Findings;
- Conclusion;

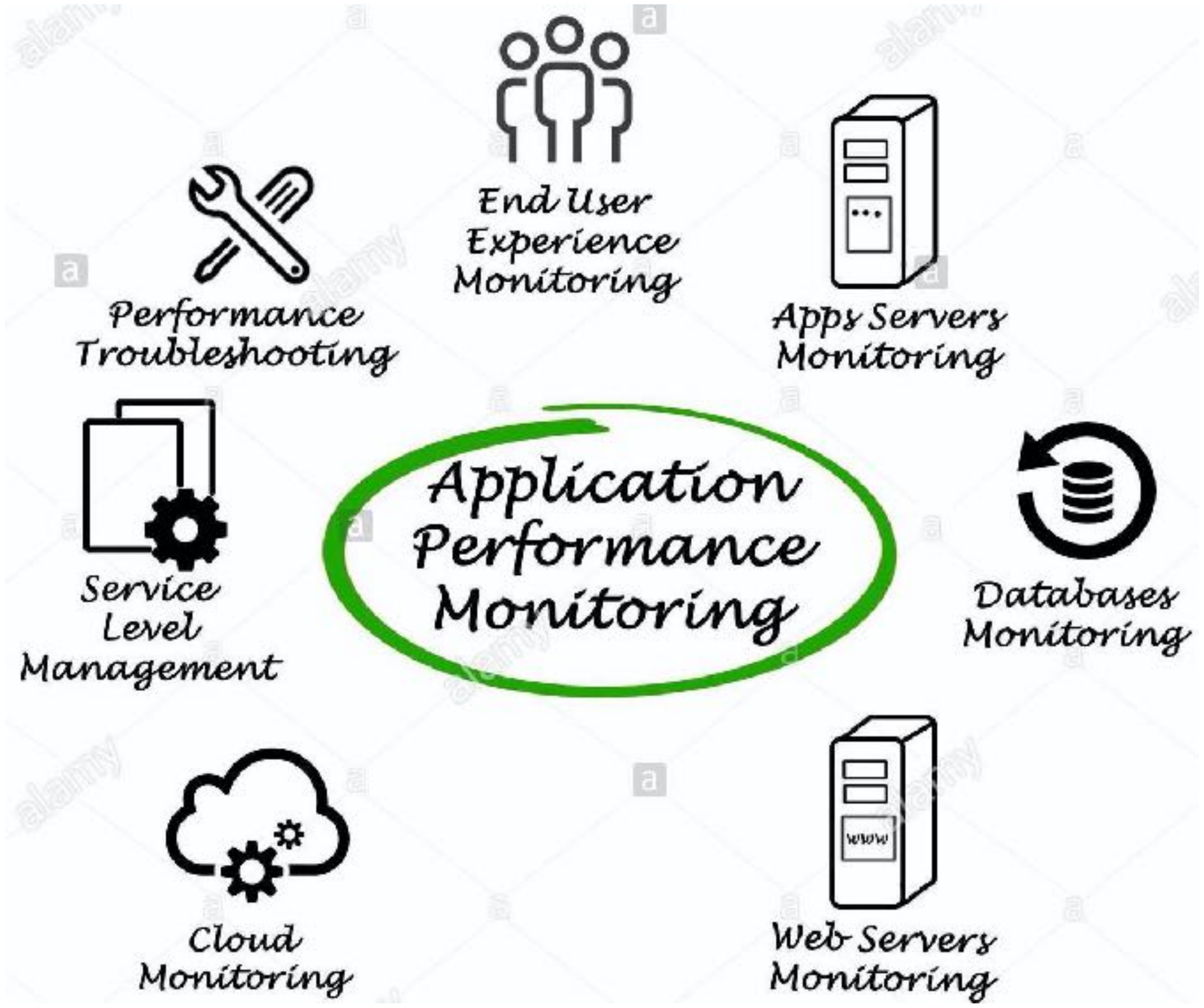
Motivation

- Performance
- All Stakeholders:
  - Developers;
  - End-users;

Motivation (2)

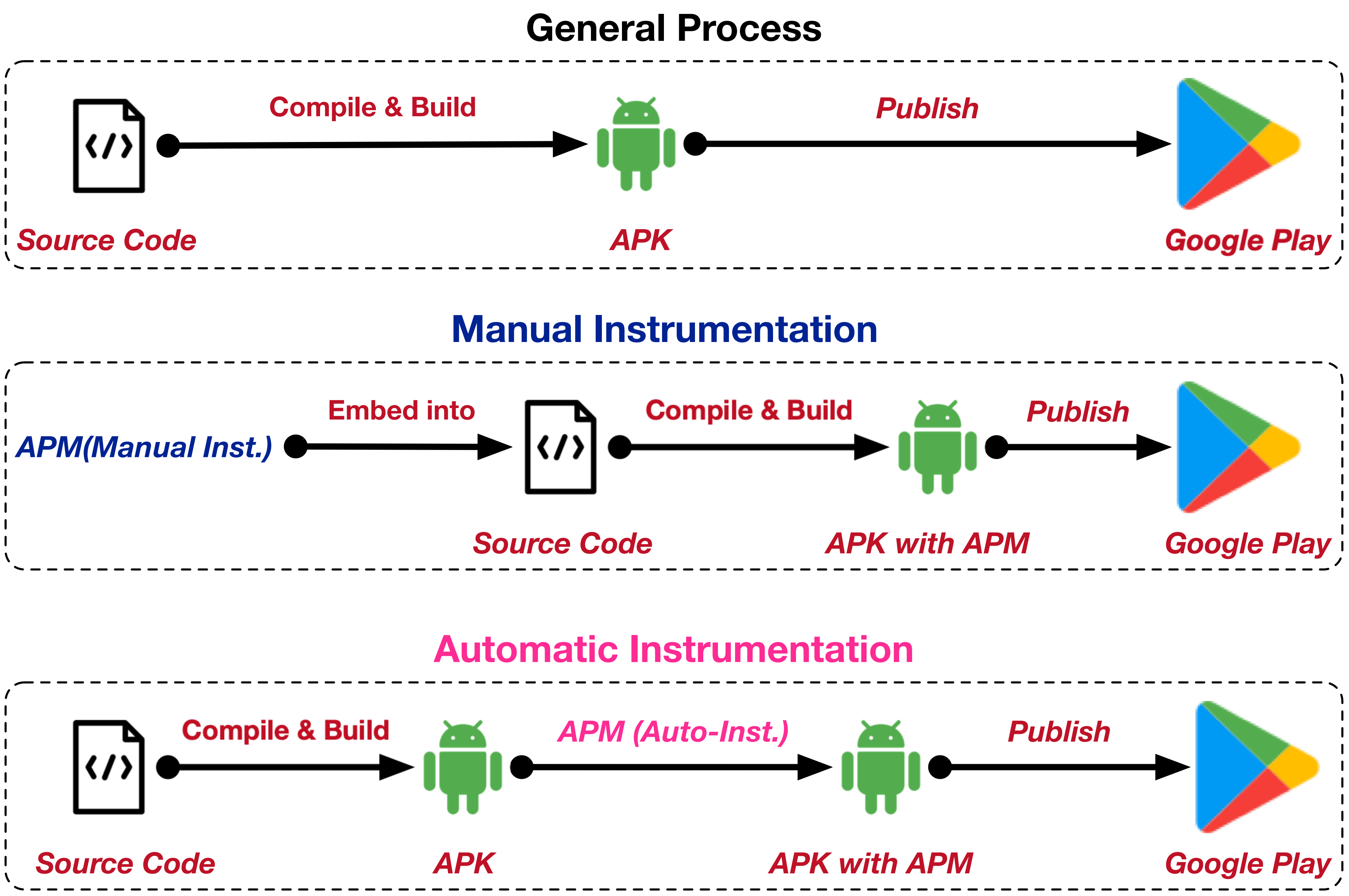
- In today’s application economy, you cannot afford to have problems that can bring an app down.
- You need to find an effective way to:
  - Build …
  - Deploy …
  - …

Motivation (3)



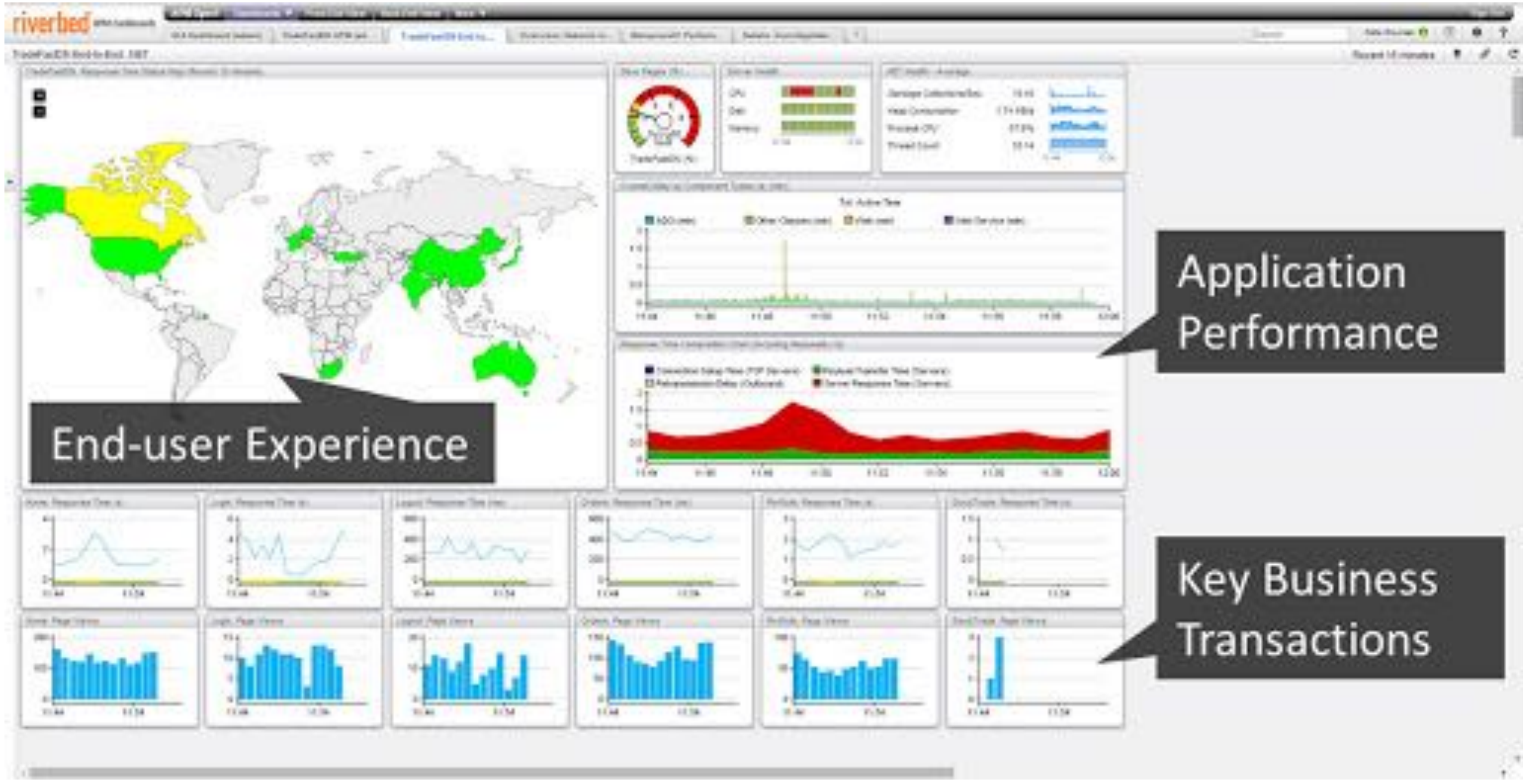
APM

- Application Performance Management (APM)



APM (2)

- Developer ID;
- APM SDK;
- Project ID <—-> app;
- Integrate & Distribute;
- APM Console;



APM Libraries Studied

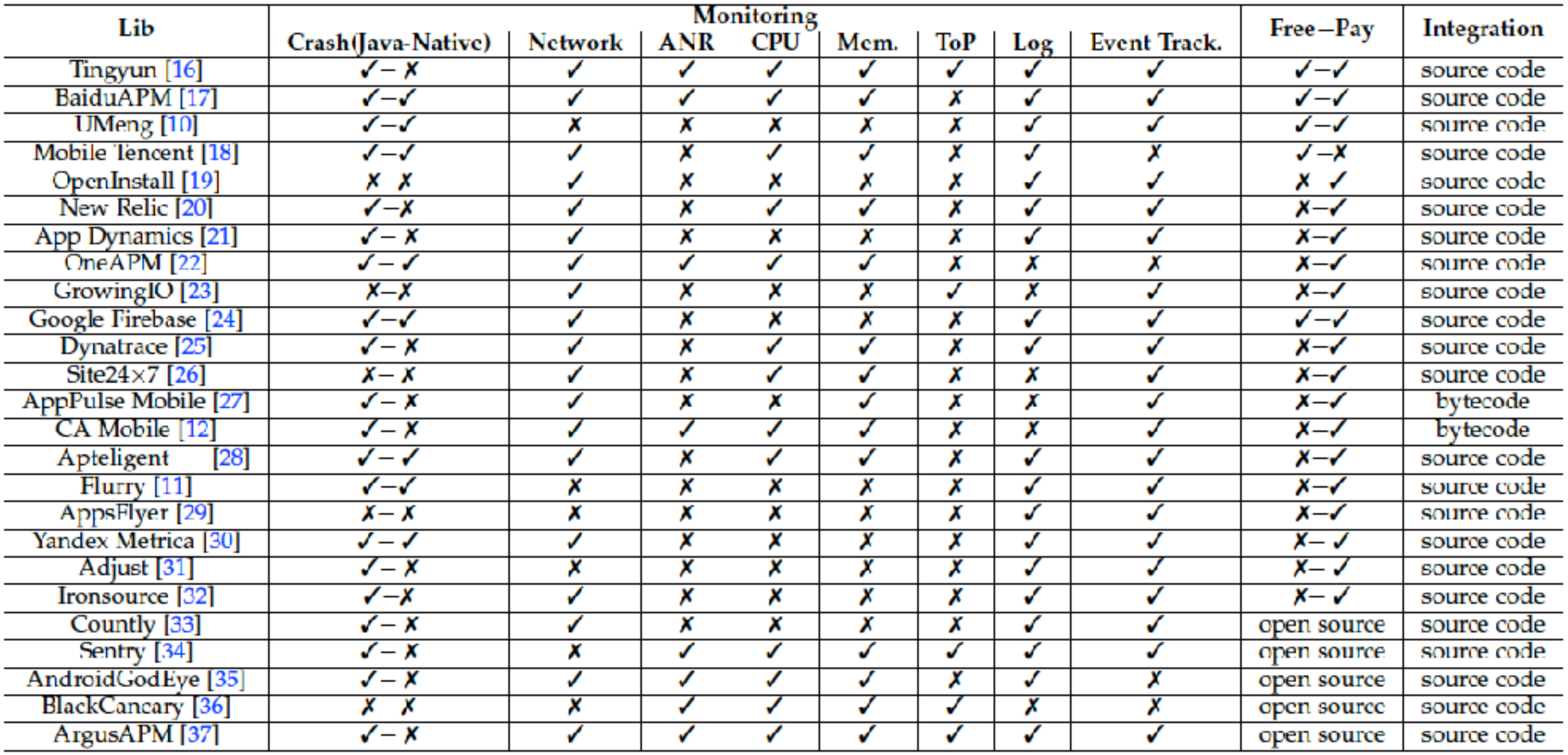
- Tingyun Baidu UMeng
- Mobile Tencent OpenInstall
- New Relic App Dynamics One APM
- GrowingIO Google Firebase Dynatrace
- Site24*7 AppPulse Mobile CA Mobile
- Apteligent Flurry AppsFlyer
- Yandex Metrica Adjust Ironsource
- Countly Sentry AndroidGodEye
- BlackCancary ArgusAPM

Functionalities

- Crash (Java, Native)
- Network
- ANR (Android Not Responding error)
- CPU
- Memory
- Time-on-Page
- Logging
- Event Tracking

Functionalities (2)

TABLE 1: APM Libraries Studied

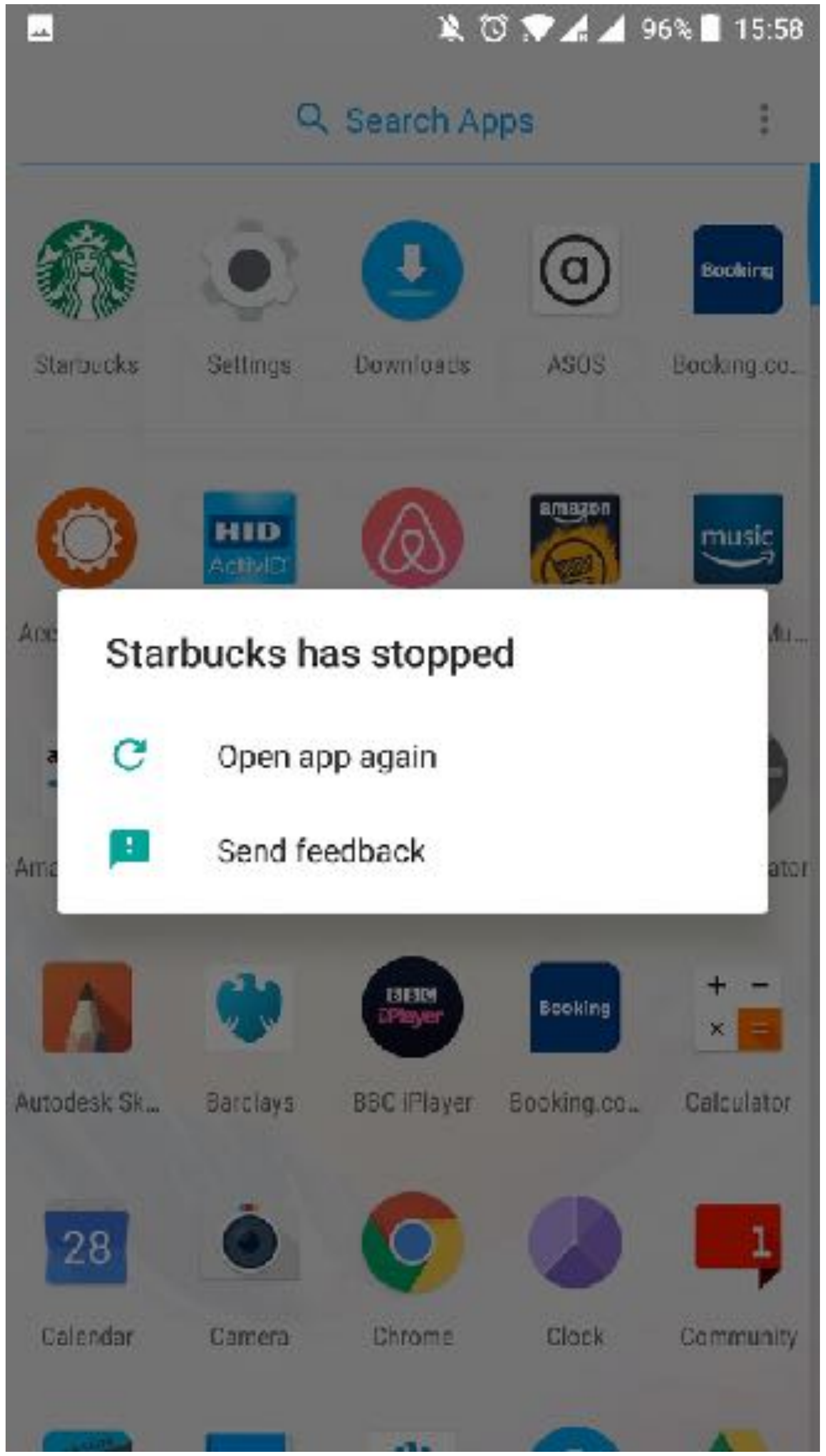


Lib	Monitoring								Free–Pay	Integration
	Crash(Java-Native)	Network	ANR	CPU	Mem.	ToP	Log	Event Track.		
Tingyun [16]	✓– ✗	✓	✓	✓	✓	✓	✓	✓	✓–✓	source code
BaiduAPM [17]	✓–✓	✓	✓	✓	✓	✗	✓	✓	✓–✓	source code
UMeng [10]	✓–✓	✗	✗	✗	✗	✗	✓	✓	✓–✓	source code
Mobile Tencent [18]	✓–✓	✓	✗	✓	✓	✗	✓	✗	✓–✗	source code
OpenInstall [19]	✗ ✗	✓	✗	✗	✗	✗	✓	✓	✗ ✓	source code
New Relic [20]	✓–✗	✓	✗	✓	✓	✗	✓	✓	✗–✓	source code
App Dynamics [21]	✓– ✗	✓	✗	✗	✗	✗	✓	✓	✗–✓	source code
OneAPM [22]	✓– ✓	✓	✓	✓	✓	✗	✗	✗	✗–✓	source code
GrowingIO [23]	✗–✗	✓	✗	✗	✗	✓	✗	✓	✗–✓	source code
Google Firebase [24]	✓–✓	✓	✗	✗	✗	✗	✓	✓	✓–✓	source code
Dynatrace [25]	✓– ✗	✓	✗	✓	✓	✗	✓	✓	✗–✓	source code
Site24×7 [26]	✗– ✗	✓	✗	✓	✓	✗	✗	✓	✗–✓	source code
AppPulse Mobile [27]	✓– ✗	✓	✗	✗	✓	✗	✗	✓	✗–✓	bytecode
CA Mobile [12]	✓– ✗	✓	✓	✓	✓	✗	✗	✓	✗–✓	bytecode
Apteligent [28]	✓– ✓	✓	✗	✓	✓	✗	✓	✓	✗–✓	source code
Flurry [11]	✓–✓	✗	✗	✗	✗	✗	✓	✓	✗–✓	source code
AppsFlyer [29]	✗– ✗	✗	✗	✗	✗	✗	✓	✓	✗–✓	source code
Yandex Metrica [30]	✓– ✓	✓	✗	✗	✗	✗	✓	✓	✗– ✓	source code
Adjust [31]	✓– ✗	✗	✗	✗	✗	✗	✓	✓	✗– ✓	source code
Ironsource [32]	✓–✗	✓	✗	✗	✗	✗	✓	✓	✗– ✓	source code
Countly [33]	✓– ✗	✓	✗	✗	✗	✗	✓	✓	open source	source code
Sentry [34]	✓– ✗	✗	✓	✓	✓	✓	✓	✓	open source	source code
AndroidGodEye [35]	✓– ✗	✓	✓	✓	✓	✗	✓	✗	open source	source code
BlackCancary [36]	✗ ✗	✗	✓	✓	✓	✓	✗	✗	open source	source code
ArgusAPM [37]	✓– ✗	✓	✓	✓	✓	✓	✓	✓	open source	source code

Network: network diagnosis; ANR: Android Not Responding; CPU: CPU utilization; Mem.: Memory usage; ToP: Time on page; Log: customized logging.

Crash

- Java
- Native (C/C++)

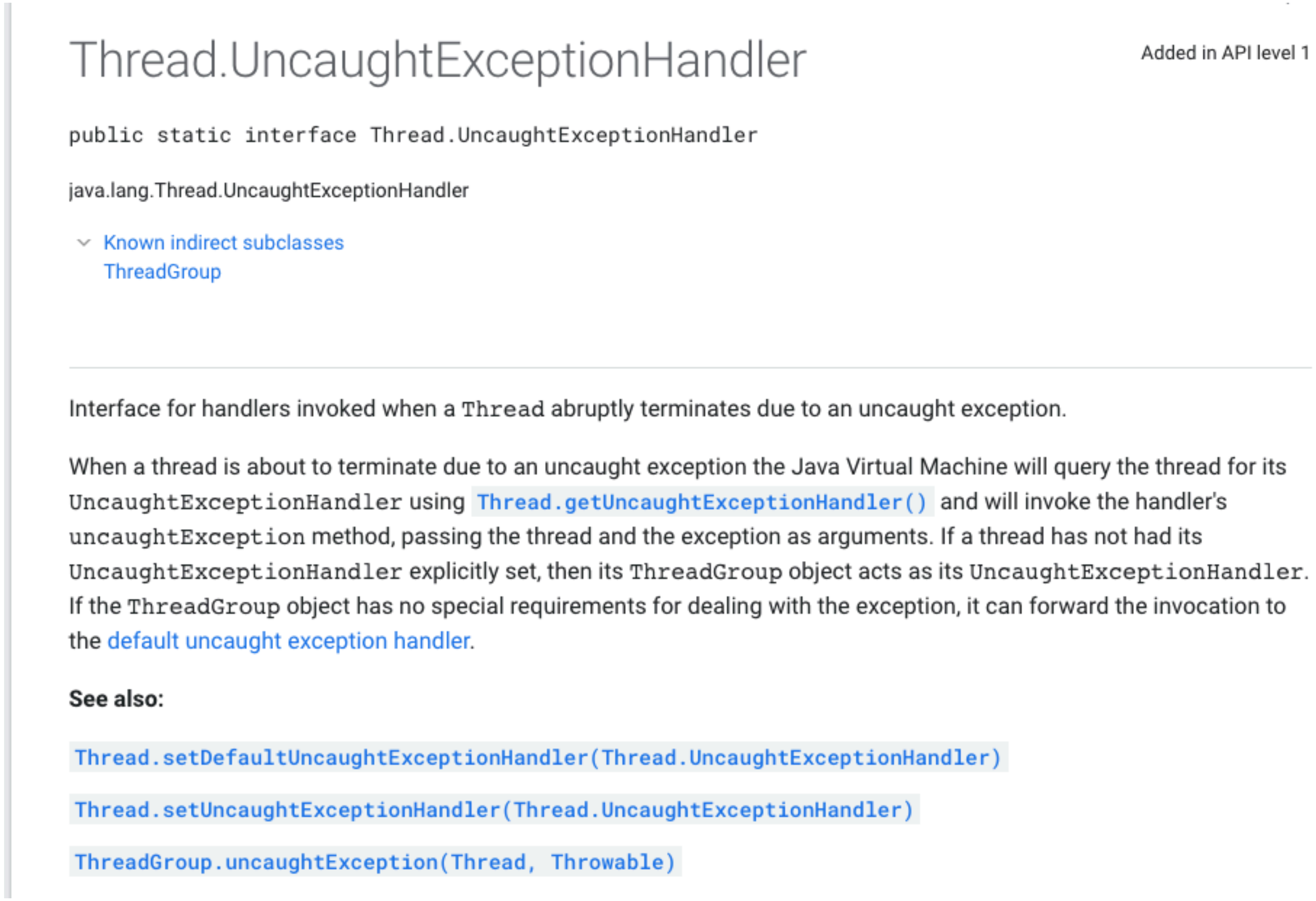


Starbucks has stopped

Open app again

Send feedback

Crash — Java



Thread.UncaughtExceptionHandler

Added in API level 1

`public static interface Thread.UncaughtExceptionHandler`

java.lang.Thread.UncaughtExceptionHandler

Known indirect subclasses
ThreadGroup

Interface for handlers invoked when a `Thread` abruptly terminates due to an uncaught exception.

When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for its `UncaughtExceptionHandler` using `Thread.getUncaughtExceptionHandler()` and will invoke the handler's `uncaughtException` method, passing the thread and the exception as arguments. If a thread has not had its `UncaughtExceptionHandler` explicitly set, then its `ThreadGroup` object acts as its `UncaughtExceptionHandler`. If the `ThreadGroup` object has no special requirements for dealing with the exception, it can forward the invocation to the default uncaught exception handler.

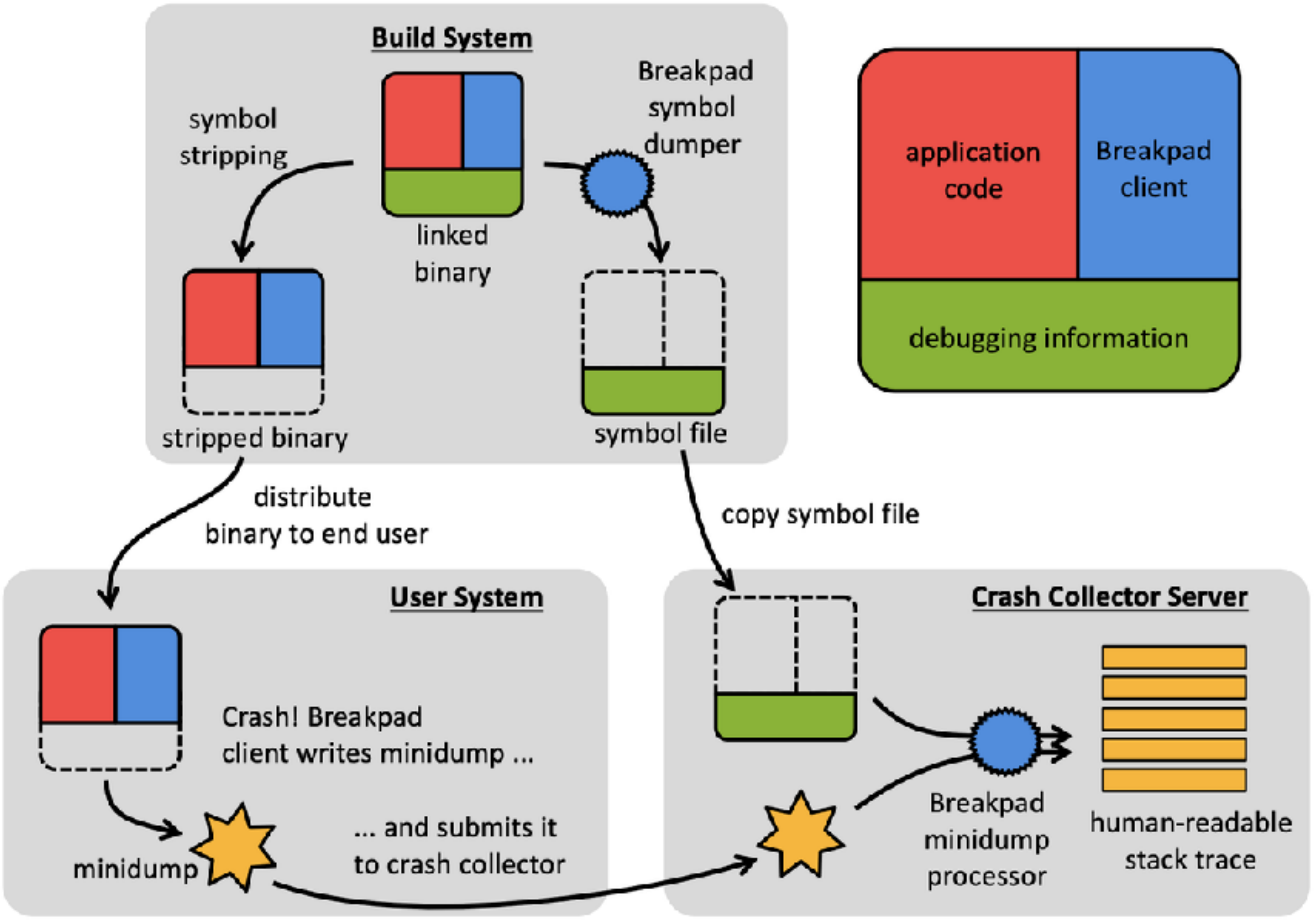
See also:

`Thread.setDefaultUncaughtExceptionHandler(Thread.UncaughtExceptionHandler)`

`Thread.setUncaughtExceptionHandler(Thread.UncaughtExceptionHandler)`

`ThreadGroup.uncaughtException(Thread, Throwable)`

Crash – Native



Crash — Java & Native

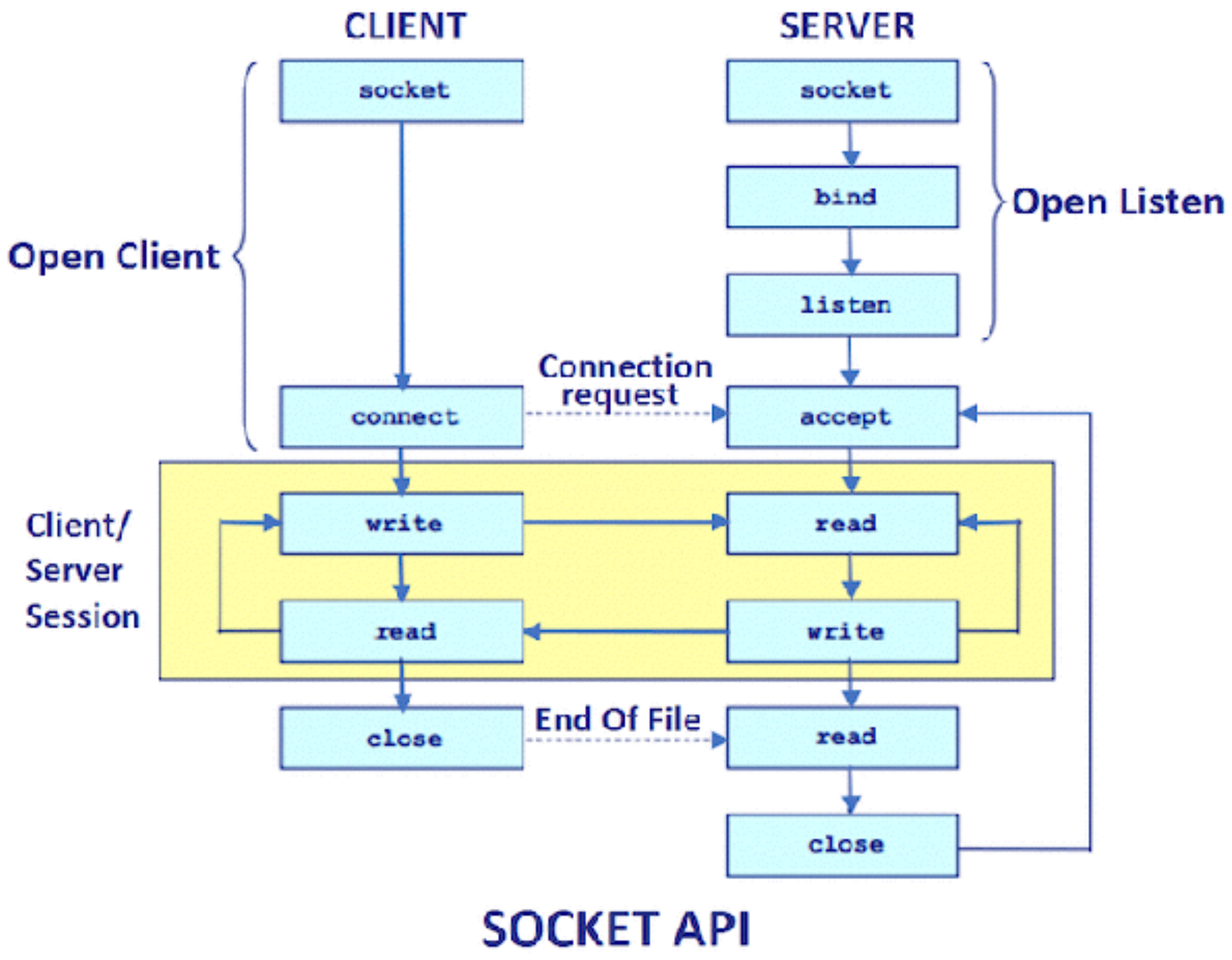
- The main limitation is that using setDefaultUncaughtExceptionHandler can update the UncaughtExceptionHandler for the Android framework. If an app uses two APMs, only the last initialized APM can capture uncaught exceptions, because only one UncaughtExceptionHandler can be defined as the default handle.

Network Diagnosis

- Socket based Solution;
- Aspect-oriented Programming (AOP);
- OKHttp 3;

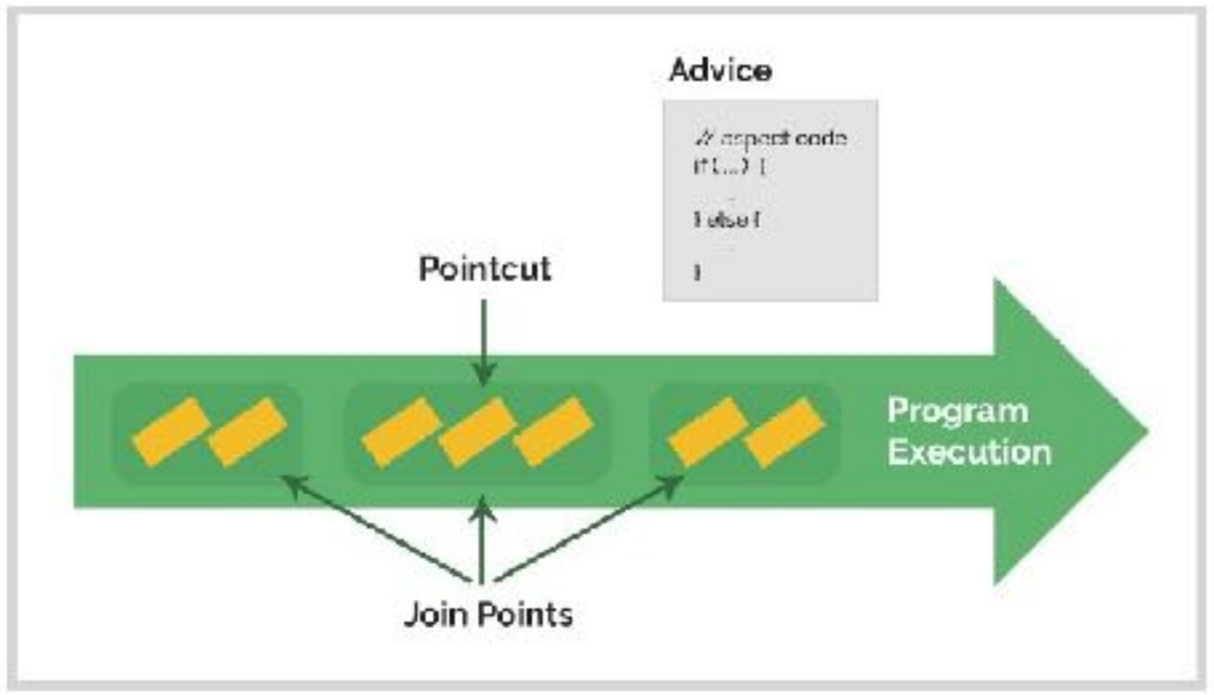
Network Diagnosis — Socket based

- Socket programming;
- Reflection;

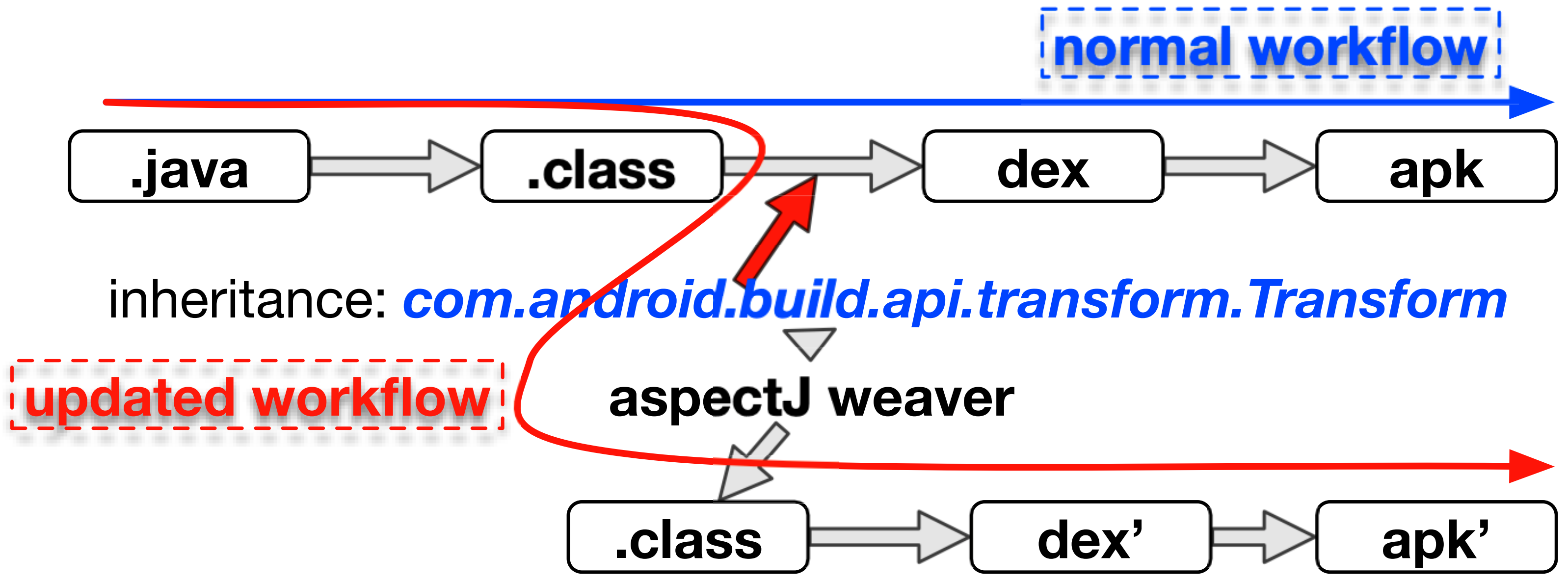


Network Diagnosis — AOP

- Aspect-oriented Programming (AOP)
- AOP is an approach to programming that allows global properties of a program to determine how it is compiled into an executable program.
- **AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns.**



Network Diagnosis — AOP (2)



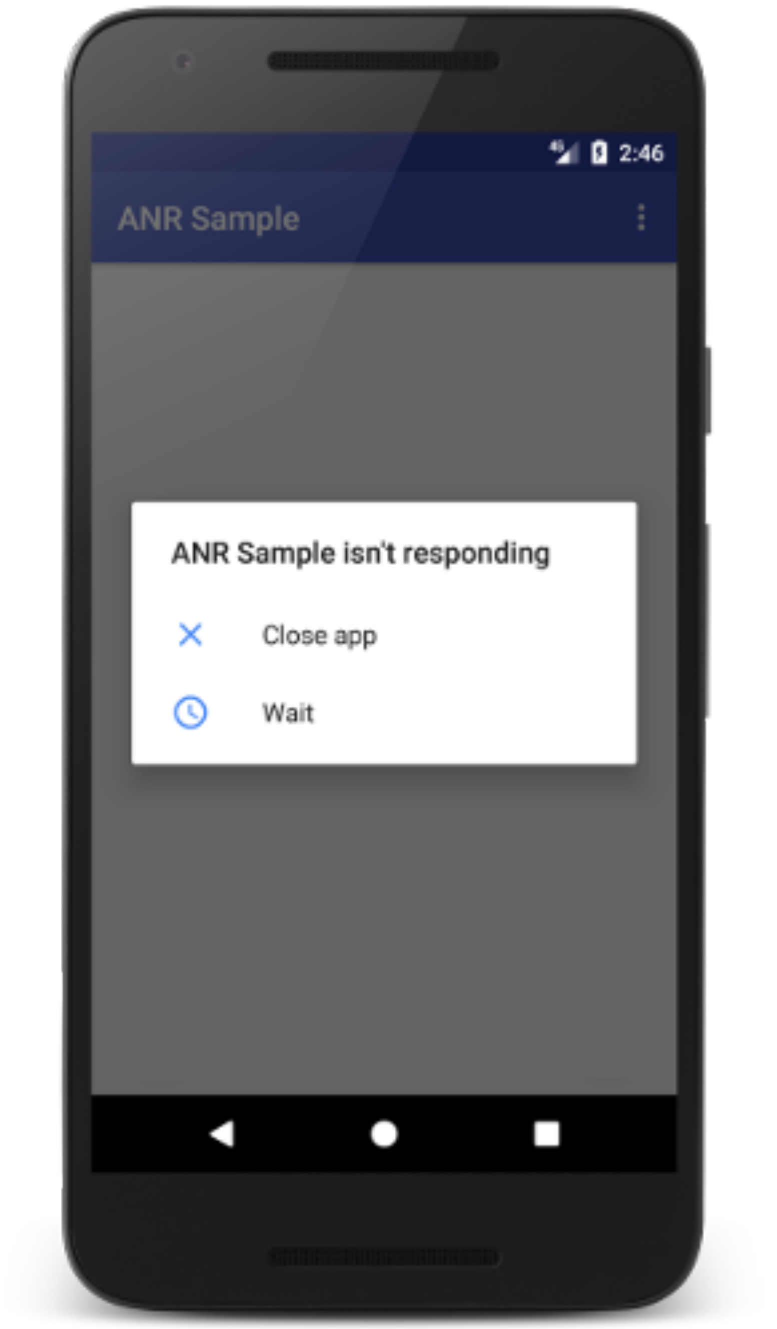
```
@Pointcut("call(org.apache.http.HttpResponse
 org.apache.http.client.HttpClient.execute(org.apache.h
 ttp.HttpHost, org.apache.http.HttpRequest)) &&
 (target(httpClient) && (args(target, request) &&
 baseCondition()))")
void httpClientExecute(HttpUriRequest
request) {
       …
}
```

Network Diagnosis — AOP (3)

- The AOP-based approach relies on Transform, which only supports the transformation from classes to dex file which Gradle build. It means that AOP-based approach is only suitable for apps built with Gradle.

ANR

- Android Not Responding error;
- ANRs are a problem because the app's main thread, which is responsible for updating the UI, can't process user input events or draw, causing frustration to the user.



ANR (2)

- Solution 1:
  - Watchdog
  - Watchdog itself is a thread. It checks the status of the main thread periodically.
  - If the main thread is frozen for over n seconds, the watchdog reports the ANR error.
- Solution 2:
  - The main thread (a.k.a Looper thread) is responsible for looping the message queue and handling messages in the message queue continuously. When the Looper is blocked (ANR error), Android outputs the ANR error into a certain trace file (data/anr/traces.txt). By rewriting the Looper.getMainLooper().setMessageLogging(Printer printer) API, APMs can capture the ANR.

Time-on-page Analysis

- Time-on-page (ToP) analysis aim at monitoring the time spent on a page (e.g., Activity).
- To compute ToP, APM must be able to capture UI display transitions.
- When a UI display changes, a new page is loaded. It suggests a display transition.
- APMs apply Choreographer.FrameCallback.doFrame() to monitor UI display transitions.
- In Android, the Choreographer component receives timing pules from the display, and then it schedules the rending work for the next display frame.

Time-on-page Analysis (2)

- The limitation of the ToP analysis is that the Choreographer API is introduced since Android 4.1 (API 16). Thus, it cannot be used for devices with an API level lower than 16.

Logging and Tracking

- Developers can employ the logging functions provided by APMs to collect users' execution traces during runtime.
- Developers can also leverage APMs to track any concerned event. For example, in New Relic APM, developers can use recordCustomEvent (type, name, attributes) to record an event at runtime. In practice, developers can use such APIs to collect user behaviors, such as preference and execution paths.

Memory, CPU, Time Consuming

- Memory
  - Memory usage data can be used to diagnose potential memory leaks in an app.
  - Approaches:
    - (1) API: ActivityManager.MemoryInfo;
    - (2) System file: /proc/meminfo;
    - (3) API: Activitymanager.getProcessMemoryInfo;
  - (1) and (3) can provide memory usage information of the target app.
  - (2) provides memory usage of all processes running in Android;

Memory, CPU, Time Consuming (2)

- CPU
  - Approaches:
    - File: /proc/cpuinfo
    - File: /proc/<pid>/stat
    - File: /proc/stat
    - File: /sys/devices/system/cpu/cpu0
  - Since Android 8.0 (API 26), the file /proc/stat cannot be visited without the root permission.

Memory, CPU, Time Consuming (3)

- Time Consuming
    - Approaches:
        - API: currentTimeMillis
        - API: TimeUnit.MILLSECOND
        - (both are defined in Java SDK)

Limitations & Drawbacks of APMs

- Requesting unnecessary or dangerous permissions.
  - READ_LOGS
  - READ_PHONE_STATE;
  - GET_TASK;
  - BLUETOOTH;
  - SYSTEM_ALERT_WINDOW;
  - SYSTEM_OVERLAY_WINDOW;
- READ_LOGS and GET_TASK are deprecated;

Limitations & Drawbacks of APMs (2)

- Accessing sensitive data and files
  - CPU
    - File: /proc/cpuinfo
    - File: /proc/<pid>/stat
    - File: /proc/stat
    - File: /sys/devices/system/cpu/cpu0
  - Memory
    - File: /proc/meminfo;

Performance Anti-patterns in Android

- Addressing performance anti-patterns

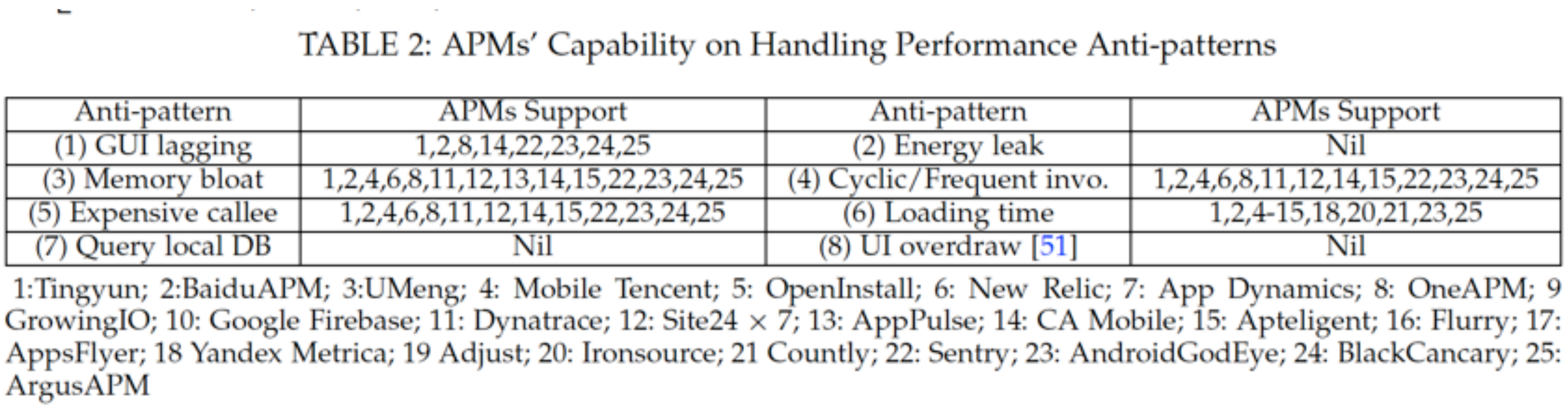


TABLE 2: APMs' Capability on Handling Performance Anti-patterns

Anti-pattern	APMs Support	Anti-pattern	APMs Support
(1) GUI lagging	1,2,8,14,22,23,24,25	(2) Energy leak	Nil
(3) Memory bloat	1,2,4,6,8,11,12,13,14,15,22,23,24,25	(4) Cyclic/Frequent invo.	1,2,4,6,8,11,12,14,15,22,23,24,25
(5) Expensive callee	1,2,4,6,8,11,12,14,15,22,23,24,25	(6) Loading time	1,2,4-15,18,20,21,23,25
(7) Query local DB	Nil	(8) UI overdraw [51]	Nil

1:Tingyun; 2:BaiduAPM; 3:UMeng; 4: Mobile Tencent; 5: OpenInstall; 6: New Relic; 7: App Dynamics; 8: OneAPM; 9 GrowingIO; 10: Google Firebase; 11: Dynatrace; 12: Site24 × 7; 13: AppPulse; 14: CA Mobile; 15: Apteligent; 16: Flurry; 17: AppsFlyer; 18 Yandex Metrica; 19 Adjust; 20: Ironsource; 21 Countly; 22: Sentry; 23: AndroidGodEye; 24: BlackCancary; 25: ArgusAPM

Methodology

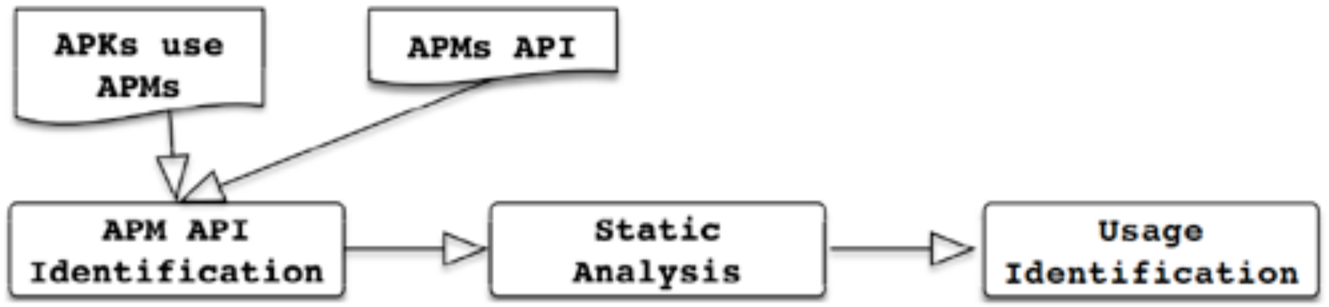
- APMHunter
- 

Fig. 6: The Overview of APMHunter

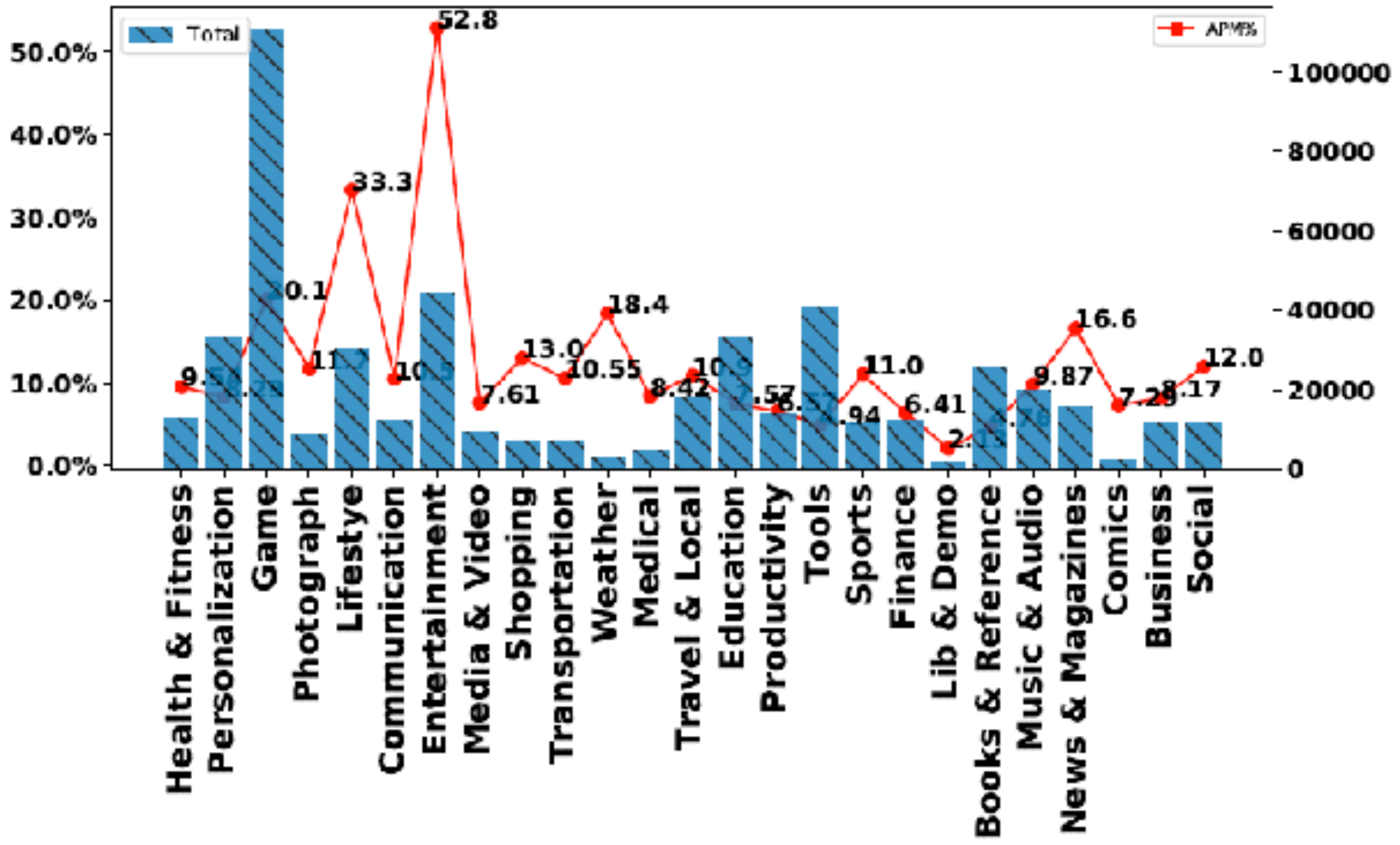
- Module 1: APM API Identification;
- Module 2: Static Analysis;
- Module 3: Usage Identification;

Evaluation

- We randomly crawl 500,000 Android apps from Google Play;
- Covers 25 main app categories;
- The game apps are grouped in different sub-categories. In our taxonomy, we consider all the subcategories of game as one (i.e., Game);
- The sizes of these apps range from 100KB to 1.2 GB.

Evaluation (2) — Popularity

- Popularity of APMs in the Wild.
- RQ1: How prevalent are APMs in Android apps?



Evaluation (3) — Popularity

- Popularity of APMs in terms of popularity.

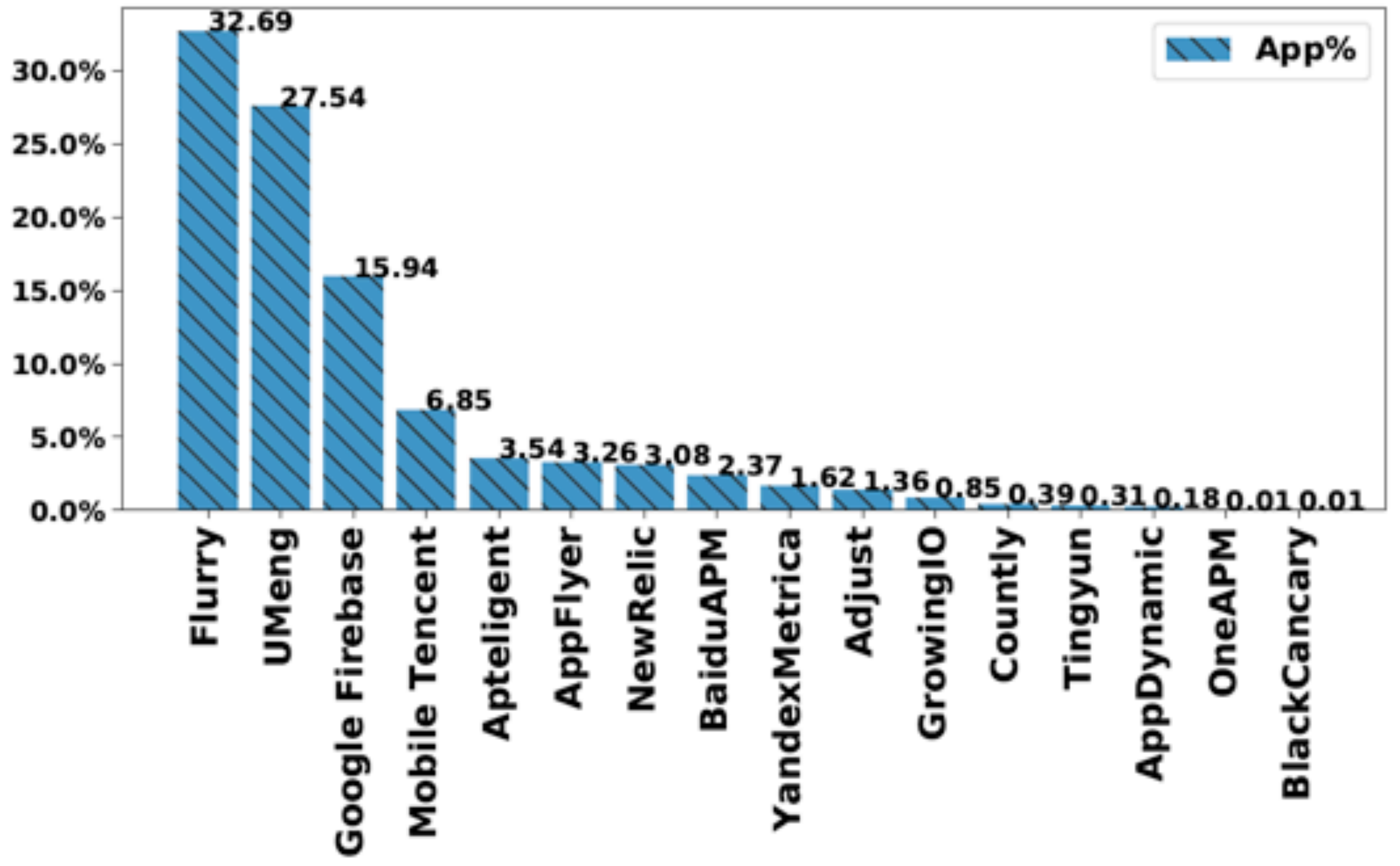


Fig. 11: Share of APMs in terms of Popularity

Evaluation (4) — Popularity

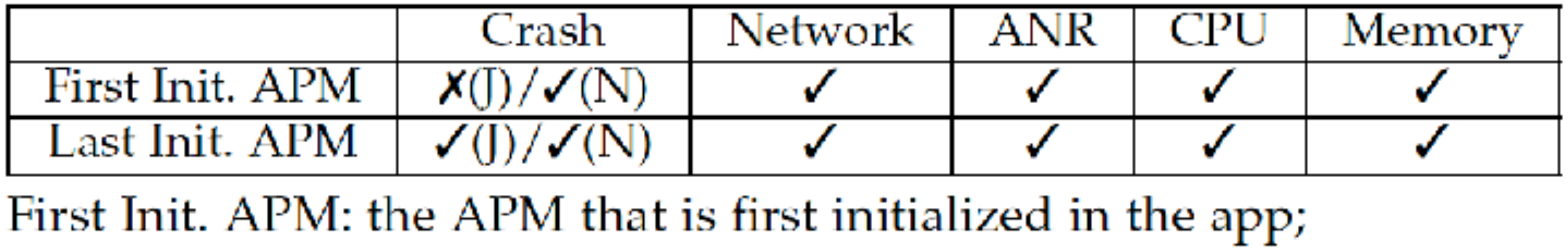
- Commercial APMs vs. Open source APMs
  - Functionality.
  - Usability.
  - Costs.
  - Technical Support.

Evaluation (5) — APMs in Practice

- APMs in Practice
- RQ2. Do developers still use logging frameworks (e.g. android.util.Log) even they have logging functions from APMs?
- 22,739 app out of 55,722 APM-integrated apps (40.8%) also adopt logging frameworks. APMs cannot fully replace logs and vice versa. The reason is thatAPMs are not suitable for local debugging whereas most log frameworks cannot collect runtime data once apps are deployed. Therefore, the intentions of using APMs and logging may be different.

Evaluation (6) — APMs in Practice

- RQ 3: What are the consequences of using multiple APMs in one app?



	Crash	Network	ANR	CPU	Memory
First Init. APM	✗(J)/✓(N)	✓	✓	✓	✓
Last Init. APM	✓(J)/✓(N)	✓	✓	✓	✓

First Init. APM: the APM that is first initialized in the app;

- In this study, we find that the key functions in each APM are the same even though the implementations can be different. Leveraging multiple APMs cannot improve the APMs' monitoring capability. Even worse, we find that using multiple APMs in one app can lead to side effects.

Evaluation (7) — Sensitive data

- RQ 4. Will app developers collect sensitive data using APMs?
- 42% app
- Developers care more about:
  - (1) hardware device ID;
  - (2) user location information;
  - (3) users' preferences;

Evaluation (8) Overhead

- RQ 5. What is the performance overhead of using APMs?

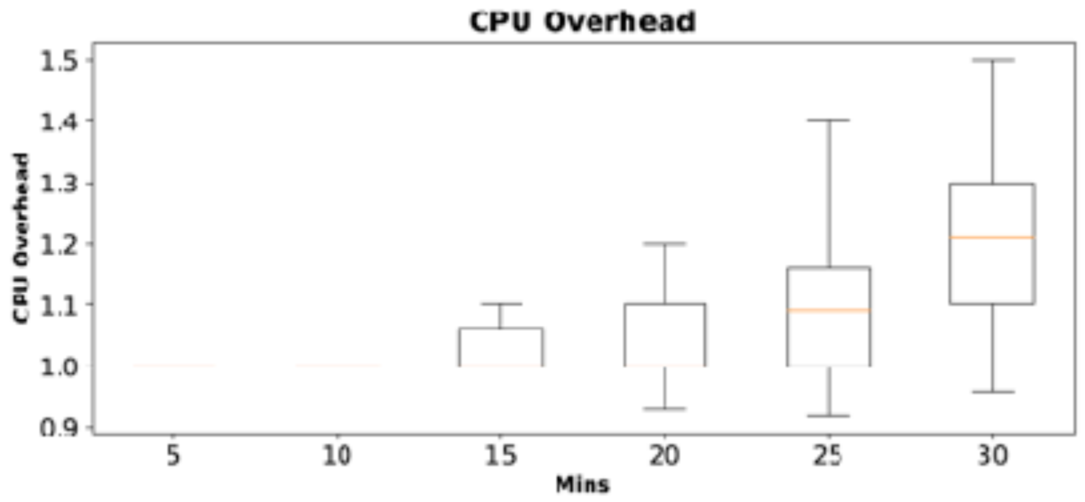


Fig. 12: CPU Overhead

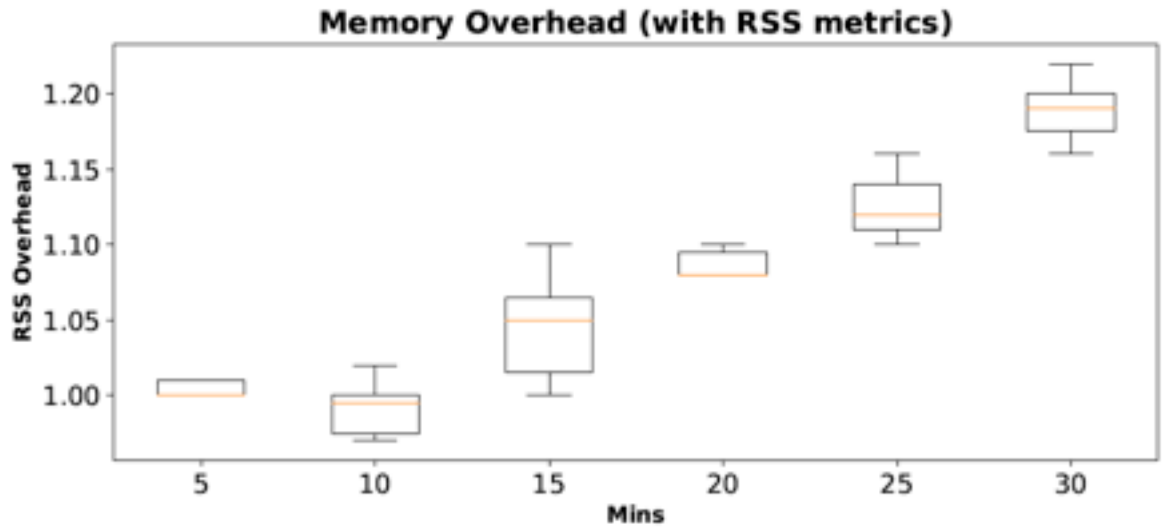


Fig. 14: Memory Overhead (with RSS metrics)

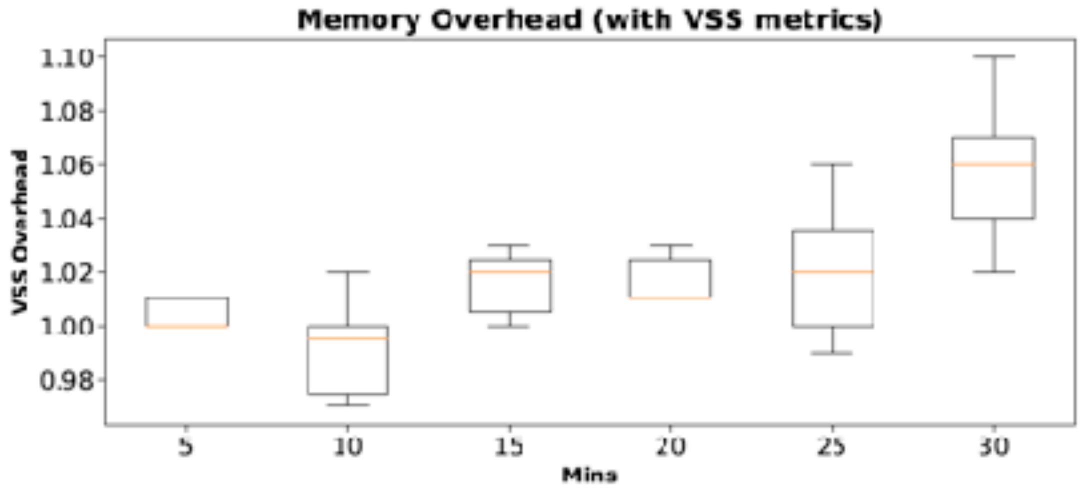


Fig. 13: Memory Overhead (with VSS metrics)

Lessons Learnt

- Suggestions for APM Vendors
  - Avoid requesting dangerous or deprecated permissions;
  - Avoid accessing sensitive files;
  - Introduce additional features to handle known performance anti-patterns;
  - Respect to the changes on Android;
  - Privacy management;
  - Prevent app developers from building malicious apps with APMs;

Lessons Learnt (2)

- Suggestions for App Developers
  - Avoid collecting private data from users via APMs;
  - Using one APM rather than more.

Q&A