

DroidPerf: Profiling Memory Objects on Android Devices

Bolun Li, **Qidong Zhao**, Shuyin Jiao, Xu Liu
NC State University

October 5, 2023

Motivations

Existing popular profilers on ART (Android runtime)

- *PowerTutor, PowerScope* - Energy consumption analysis
- *Android profiler* - Hotspot analysis
- *Perfetto* - Trace analysis
- *Other profilers* - Cross-layer inefficiency or app crash analysis

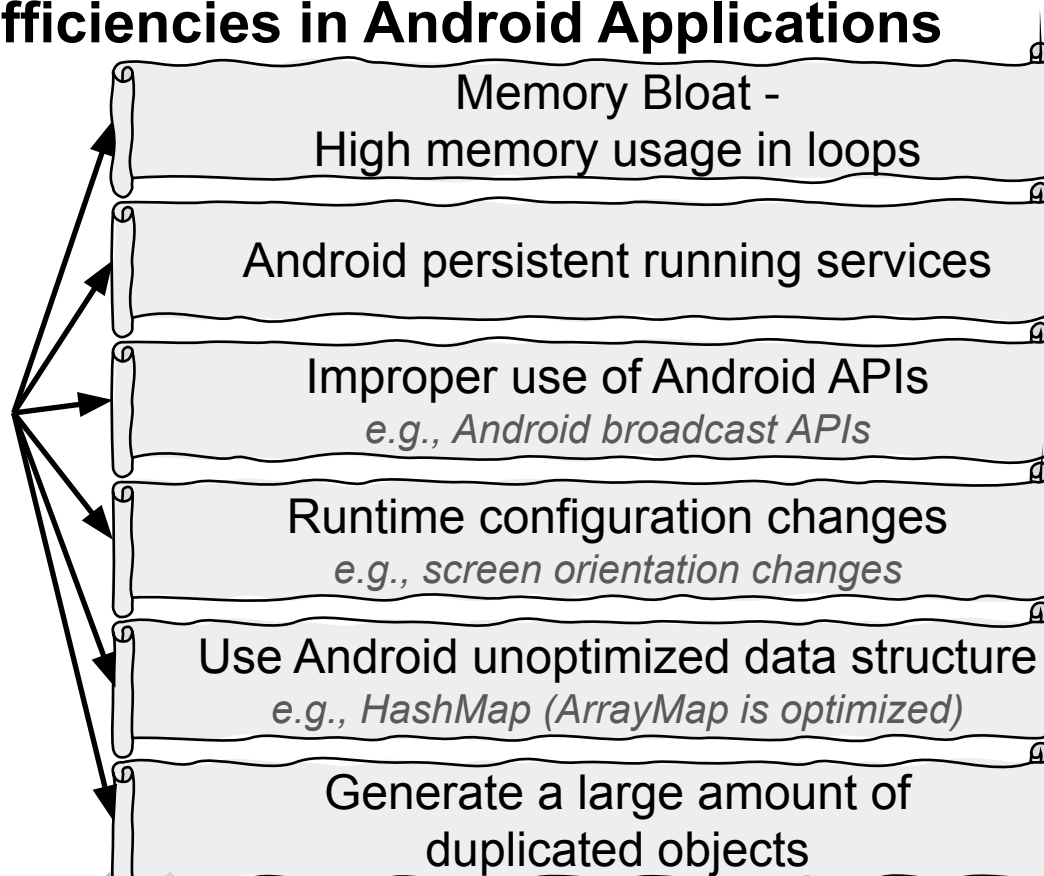
Why we need a profiler on ART (Android runtime)

- Existing tools mainly focus on hotspot analysis profiling
- Existing tools fail to tell whether a resource is being used productively and contributes to a program's overall efficiencies
- Android applications are highly susceptible to object locality issues

Need a new profiler for **profiling memory objects** on Android devices

Study of Memory Inefficiencies in Android Applications

Android unique
memory inefficiency
causes

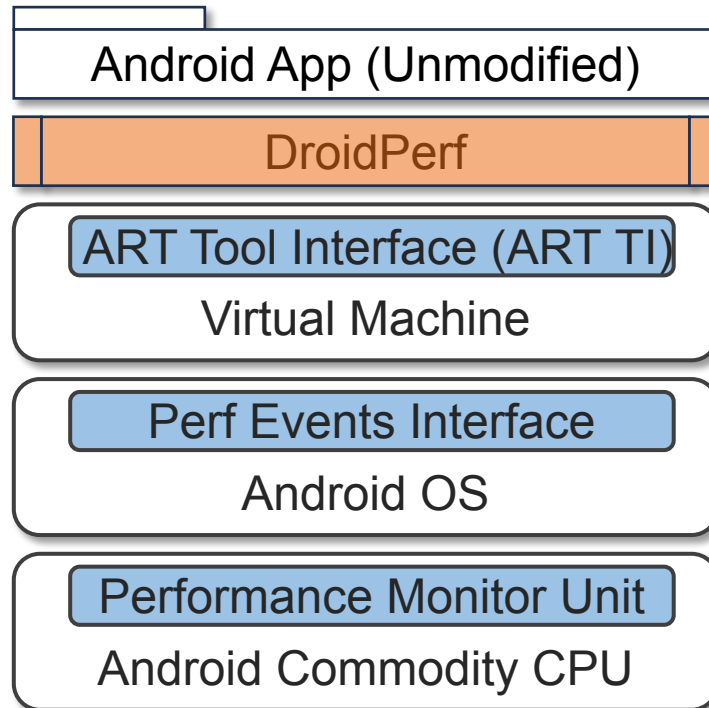


Challenges of Building an Android Memory Profiler

- ❑ Lack of library to support hook the object allocation on Android
- ❑ Limited JVM Tool Interface (JVMTI) support
 - ❑ Missing important callbacks, e.g., callbacks when a method is compiled and loaded into memory, or unloaded from memory
- ❑ Lack of async unwind facility
 - ❑ Missing AsyncGetCallTrace facility, which is used to get calling stack inside a signal handler to avoid the safe point
- ❑ Unable to obtain sampled PMU effective address on ARM
 - ❑ ARM only provides an instruction pointer on each PMU sample

Methodology

DroidPerf in system stacks



Methodology

Capture object creation and destruction

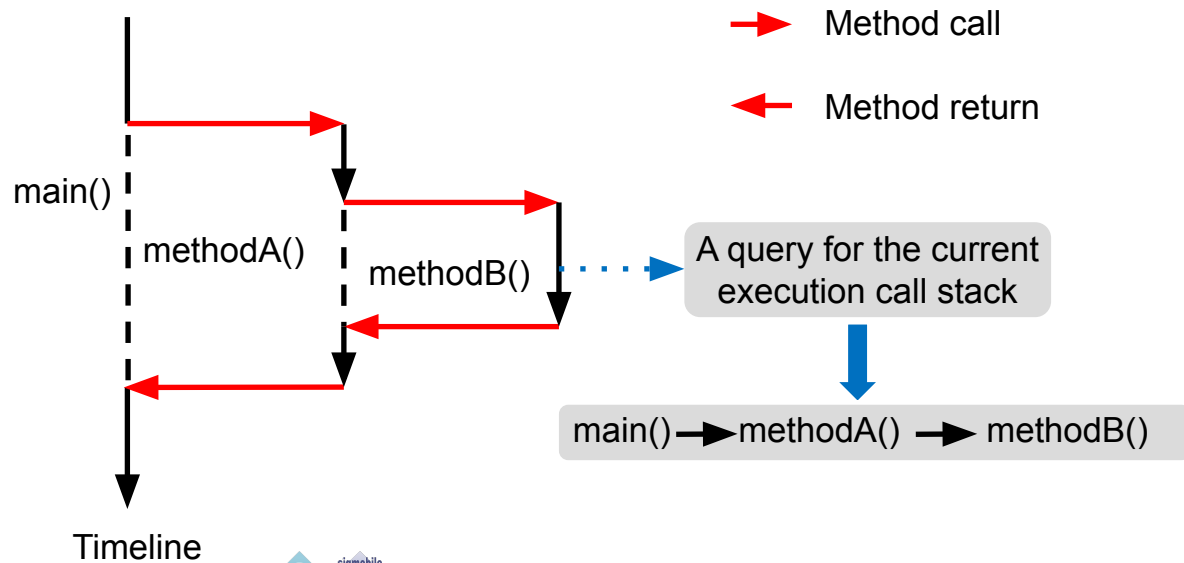
- **JVMTI ObjectAlloc Callback**
 - Allocated thread
 - Reference to the allocated object
 - Reference to the class of the allocated object
 - Object Size
- **JVMTI ObjectFree Callback**
 - Tag of the freed object

Methodology

Construct call path

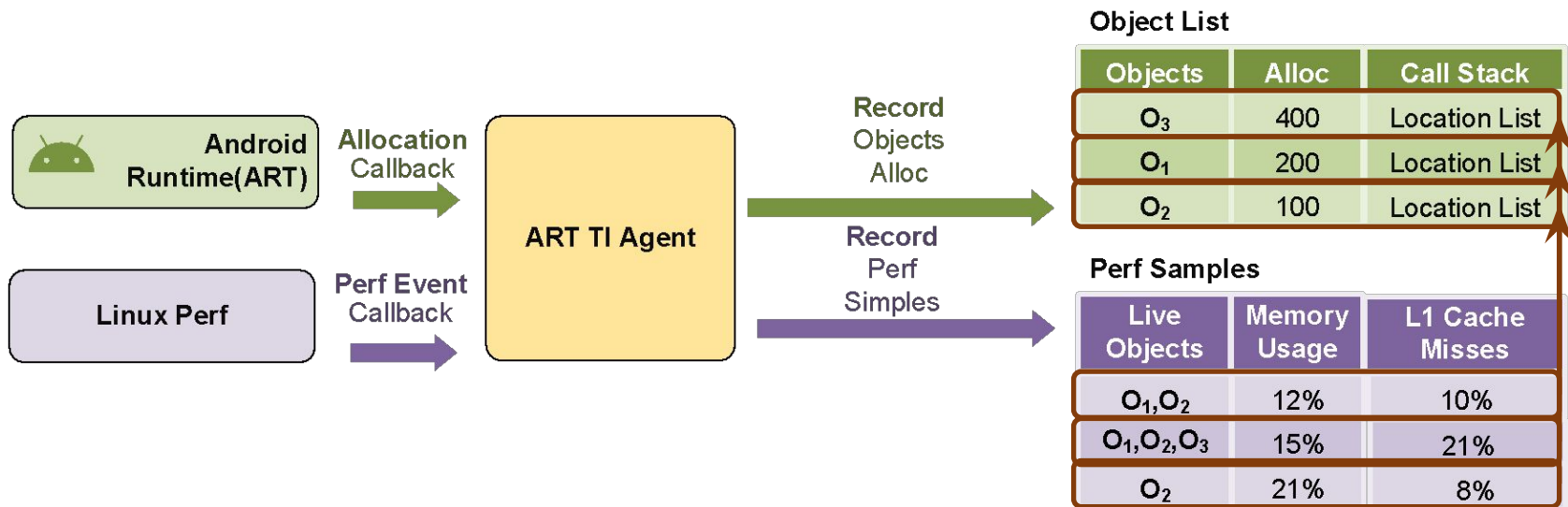
- Monitor Executed Function: **MethodEntry & MethodExit**
- Obtain line number: **GetLineNumberTable** API

```
main() {  
    methodA();  
}  
methodA() {  
    methodB();  
}
```



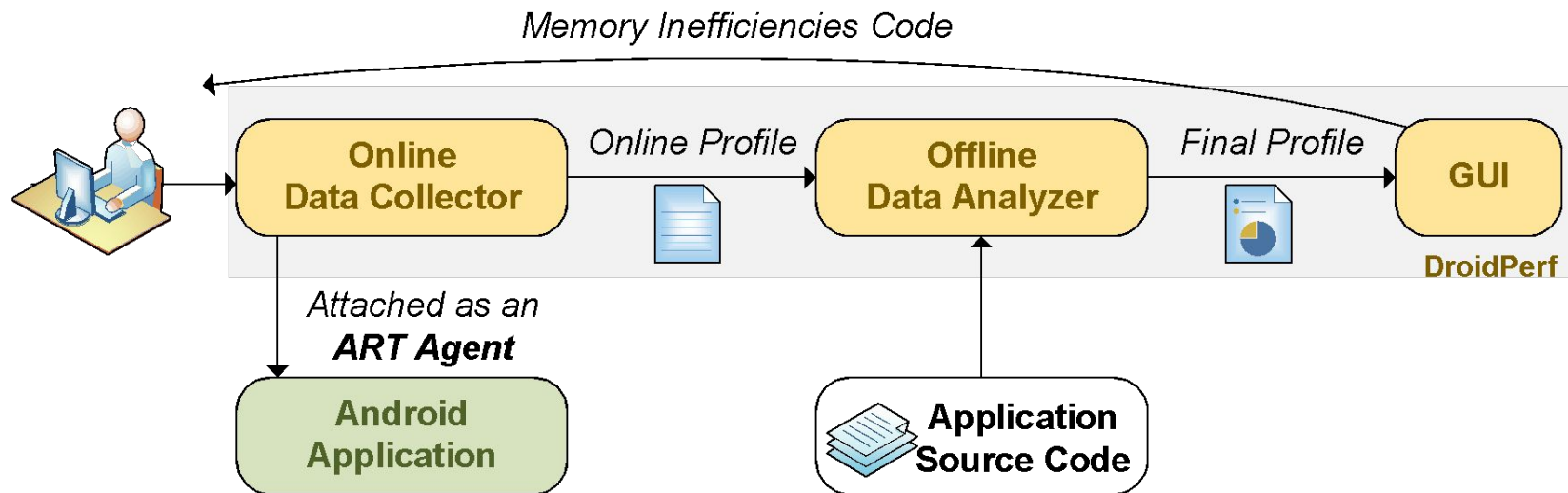
Methodology

DroidPerf's object-centric analysis



Overview of DroidPerf

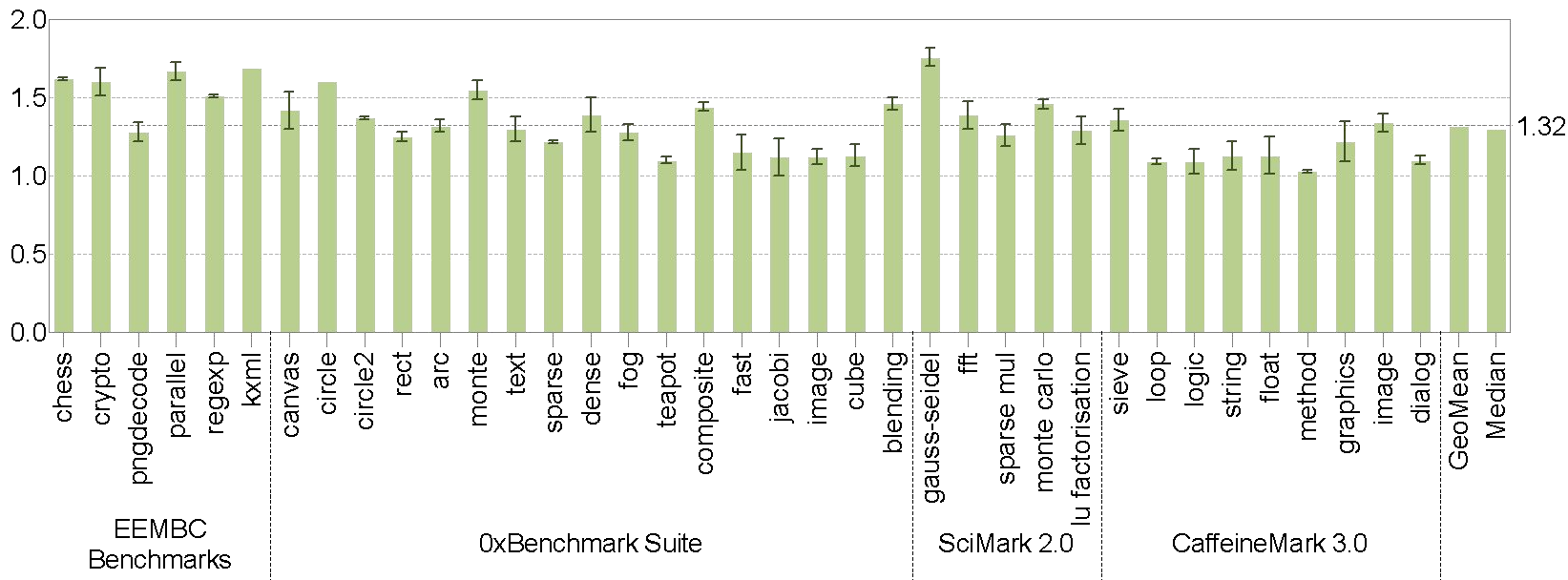
Workflow of DroidPerf



Evaluation

Time overhead

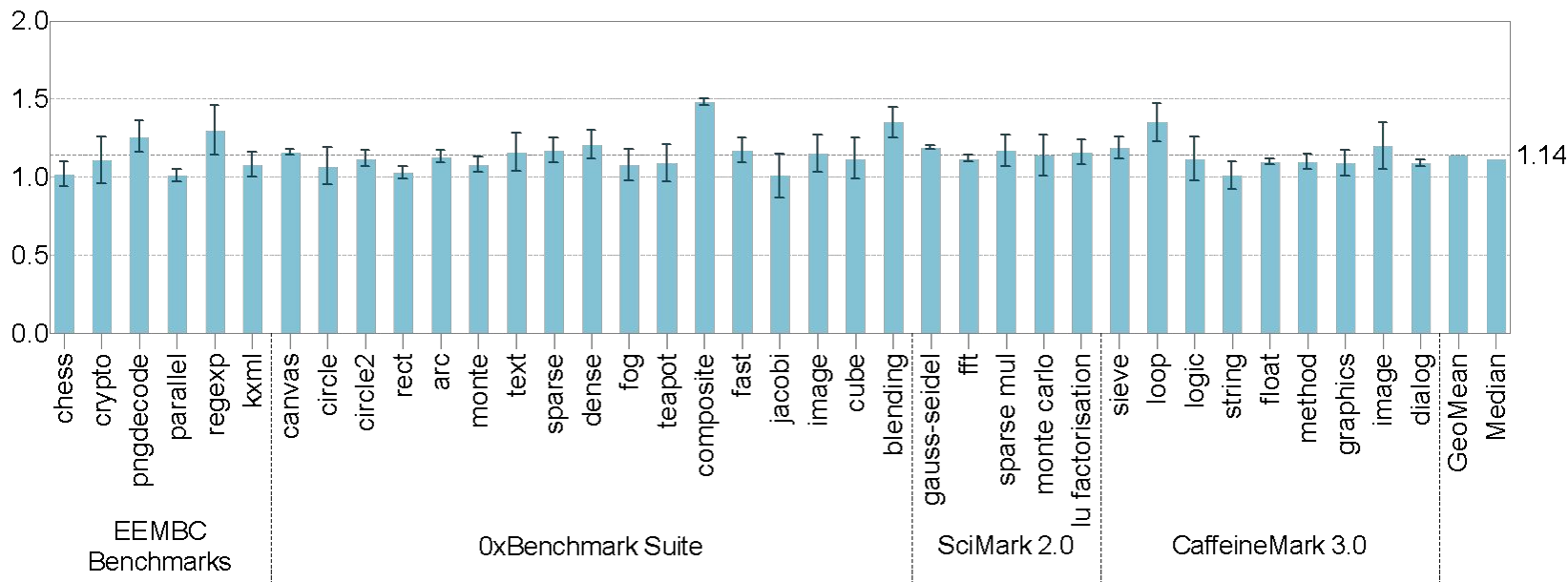
- Google Pixel 7 (Google Tensor G2 chipset, 8 GB RAM)



Evaluation

Memory overhead

- Google Pixel 7 (Google Tensor G2 chipset, 8 GB RAM)



Case Study of DNS66

VSCode GUI on Android Application DNS66

The screenshot displays the VSCode GUI for the Android application DNS66. The interface is divided into two main sections: the Flame Graph on the left and the Tree Table on the right.

Flame Graph (Left):

- Search:** Search...
- Filters:** L1 cache miss (L1 Cache miss), Top Down, Bottom Up, Flat.
- Active heap usage and L1 cache miss of the entire program:** ActiveHeapUsage: 32.6% L1 (19.698%, 1824340 samples).
- Virtual Root:**
 - java.lang.Thread.java.run():L923
 - org.jak_linux.dns66.vpn.AdVpnThread.java.run():L163
 - org.jak_linux.dns66.vpn.AdVpnThread.java.runVpn():L216
 - org.jak_linux.dns66.vpn.AdVpnThread.java.configure():L509
 - org.jak_linux.dns66.vpn.AdVpnThread.java.configurePackages():L399
- Live objects info:**
 - ActiveHeapUsage: 32.6%
 - <org.jak_linux.dns66.vpn.AdVpnThread.java (Line:396, AllocTimes:252)>
 - <org.jak_linux.dns66.vpn.AdVpnThread.java (Line:397, AllocTimes:252)>
 - <org.jak_linux.dns66.Configuration.java (Line:220, AllocTimes:37)>
 - <com.google.gson.stream.JsonReader.java (Line:923, AllocTimes:17)>
 - <com.google.gson.stream.JsonReader.java (Line:146, AllocTimes:17)>

Tree Table (Right):

- Users > AdVpnThread.java**
- Code Snippets:**
 - 389Log.i(TAG, "configure: Adding DNS Server " + addr + " as " + i6addr);
 - 390builder.addDnsServer(i6addr);
 - 391vpnWatchDog.setTarget(i6addr);
 - 392}
 - 393}
 - 394}
 - 395void configurePackages(VpnService.Builder builder, Configuration config) {
 - 396Set<String> allowOnVpn = new HashSet<>(); **allocation sites**
 - 397Set<String> doNotAllowOnVpn = new HashSet<>();
 - 398}
 - 399config.allowlist.resolve(allowOnVpn, doNotAllowOnVpn) **PMU L1 cache miss event triggered location**
 - 400}
 - 401if (config.allowlist.defaultMode == Configuration.AllowList.DEFAULT_MODE_NOT_ON_VPN) {
 - 402for (String app : allowOnVpn) {
 - 403try {
 - 404Log.d(TAG, "configure: Allowing " + app + " to use the DNS VPN");
 - 405builder.addAllowedApplication(app);
 - 406} catch (Exception e) {
 - 407Log.w(TAG, "configure: Cannot disallow", e);
 - 408}
 - 409} else {
 - 410}
 - 411for (String app : doNotAllowOnVpn) {
 - 412try {
 - 413Log.d(TAG, "configure: Disallowing " + app + " from using the DNS VPN");
 - 414builder.addDisallowedApplication(app);
 - 415} catch (Exception e) {
 - 416Log.w(TAG, "configure: Cannot disallow", e);
 - 417}

Optimization result: reduce the total cache misses by 18.97% and the heap memory usage consumption by 6.6%

Case Studies

Android Applications	Inefficiency		Optimization			
	Problematic Code	Type	Patch	WH	WCM	WEI
DNS66	AdVpnThread.java	Excessive memory usage in nested loops	Move problematic allocation sites out of loop and reset them upon request	18.97%	6.60%	10.62%
Rajawali	RenderToTextureFragment.java			15.69%	7.57%	6.37%
RxAndroid	HandlerScheduler.java			9.67%	5.48%	3.94%
Applozic	MessageClientService.java	Long running services	Avoid using persistent services	17.94%	8.16%	9.36%
GmsCore	TriggerReceiver.java	Inefficient using Android broadcast API	Unregister the broadcast receiver which is no longer needed	8.89%	5.56%	4.98%
Twire	Service.java TopStreamsActivity.java	Runtime configuration (screen orientation, etc.) changes	Reuse the visual elements	7.25%	5.37%	14.14%
Stream Chat	QueryChannelsLogic.java			7.24%	6.57%	11.01%
Stetho	AsyncPrettyPrinterRegistry.java ChromeDevtoolsServer.java JsonRpcPeer.java ResponseBodyFileManager.java	Poor data structure	Use Android optimized data structure	6.29%	13.43%	5.01%
MediaPicker	Utility.java	Generate duplicated objects	Reuse an object that memorized from the last used point	12.88%	4.06%	8.13%
LeakCanary	AndroidDebugHeapAnalyzer.java			9.49%	3.93%	7.14%
Foxy Droid	QueryBuilder.java			8.55%	1.51%	3.92%

Many optimization patches were confirmed: *Applozic, GmsCore, Twire, etc.*
 Upstreamed optimization patches: *DNS66, Rajawali*

Conclusions

- Categorize different **Android memory inefficiencies**
- Develop **DroidPerf** to pinpoint object-level inefficiencies that occurred on Android Runtime
 - Support lightweight novel object-centric profiling
 - Apply to unmodified applications
 - Obtain nontrivial performance gains
- Contribute to the community
 - **Many optimization patches** are upstreamed to real-world applications

Future Work

- ❑ Enhance DroidPerf's Feature
 - ❑ Support instruction-level monitoring
 - ❑ Support instruction-level inefficiency analysis
- ❑ Enhance DroidPerf's Usability
 - ❑ Develop an extension for Android Studio to integrate DroidPerf into the Android Development Environment