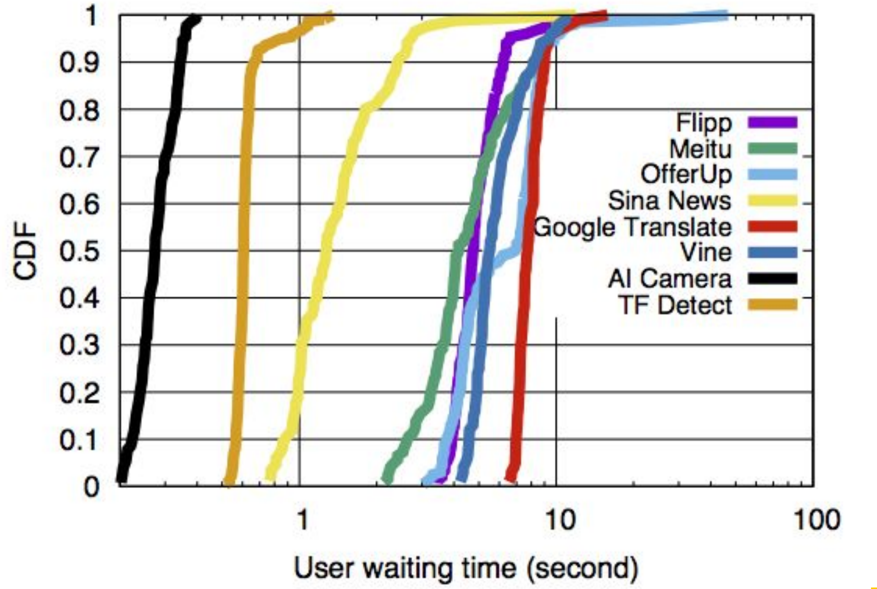
PerfProbe: A Systematic, Cross-Layer Performance Diagnosis Framework for Mobile Platforms

David Ke Hong, Ashkan Nikravesh, Z. Morley Mao, Mahesh Ketkar, and Michael Kishinevsky.

Unpredictable performance problem

- How to effectively diagnose the cause of *unpredictable performance problems* in mobile apps?
    - Study on 100 popular apps
    - Tail latency: **2~8x increase**



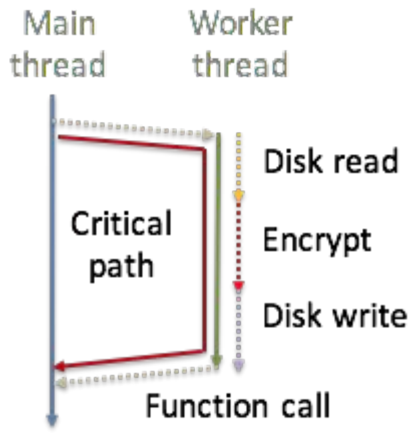
Related work

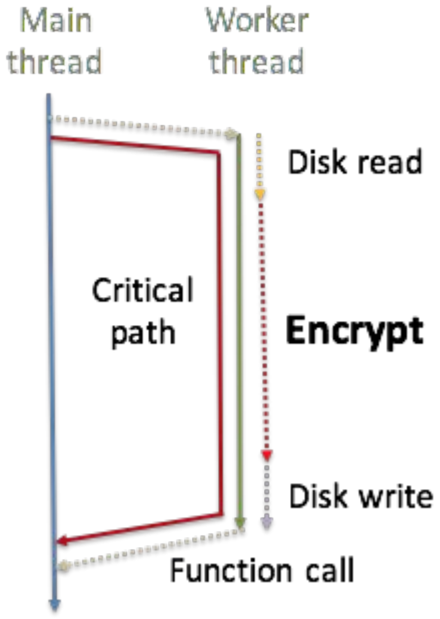
- App performance profiling
  - Existing work: AppInsight [OSDI ‘12], Traceview, etc.
  - Lack of understanding on system resource bottleneck
- OS event tracing
  - Existing work: Panappticon [CODES '13], Systrace, etc.
  - Hard to localize the source of code-level execution slowdown based on low-level OS events

[1] AppInsight: Mobile App Performance Monitoring in the Wild. In OSDI ‘12.
[2] Panappticon: Event-Based Tracing to Optimize Mobile Application and Platform Performance. In CODES+ISSS ’13.

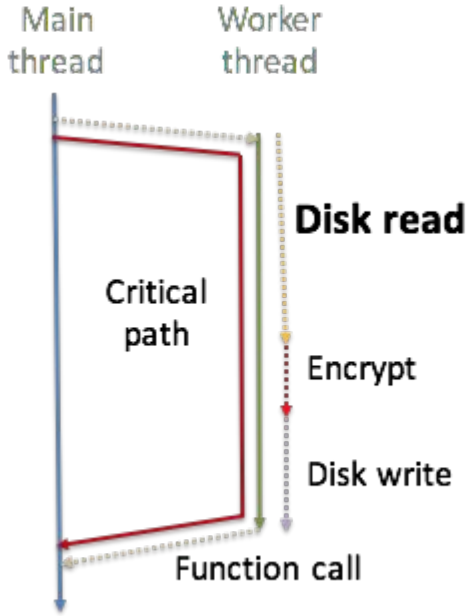
Why cross-layer profiling & analysis

- **Motivating example**: encrypt a file on SD card



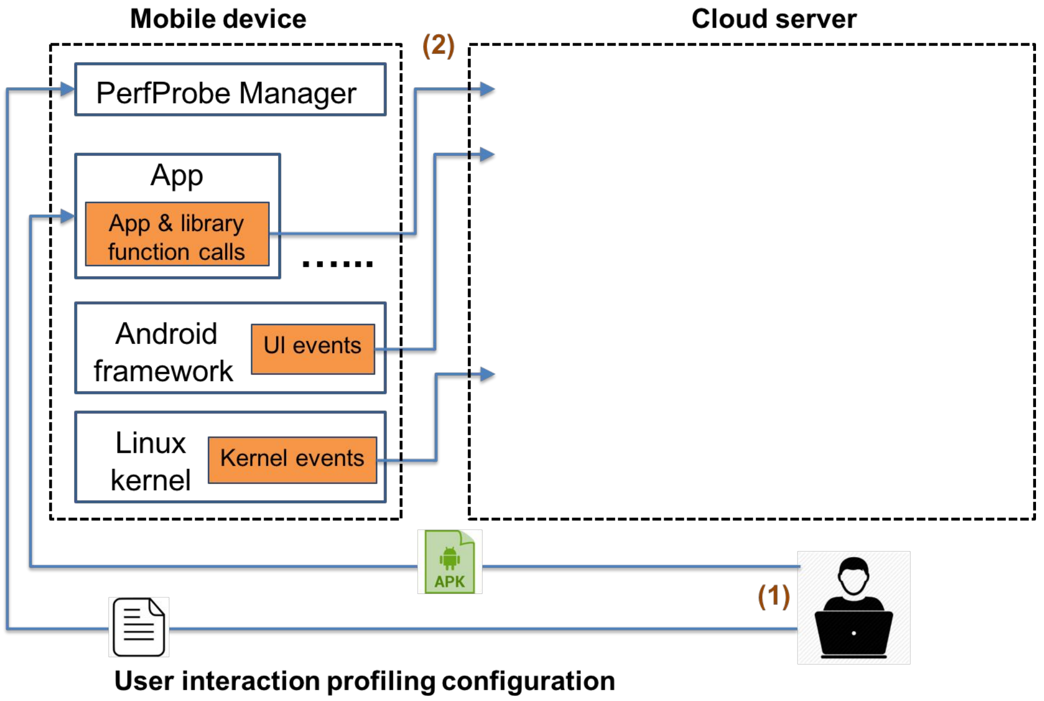


Slow due to computation



Slow due to disk I/O

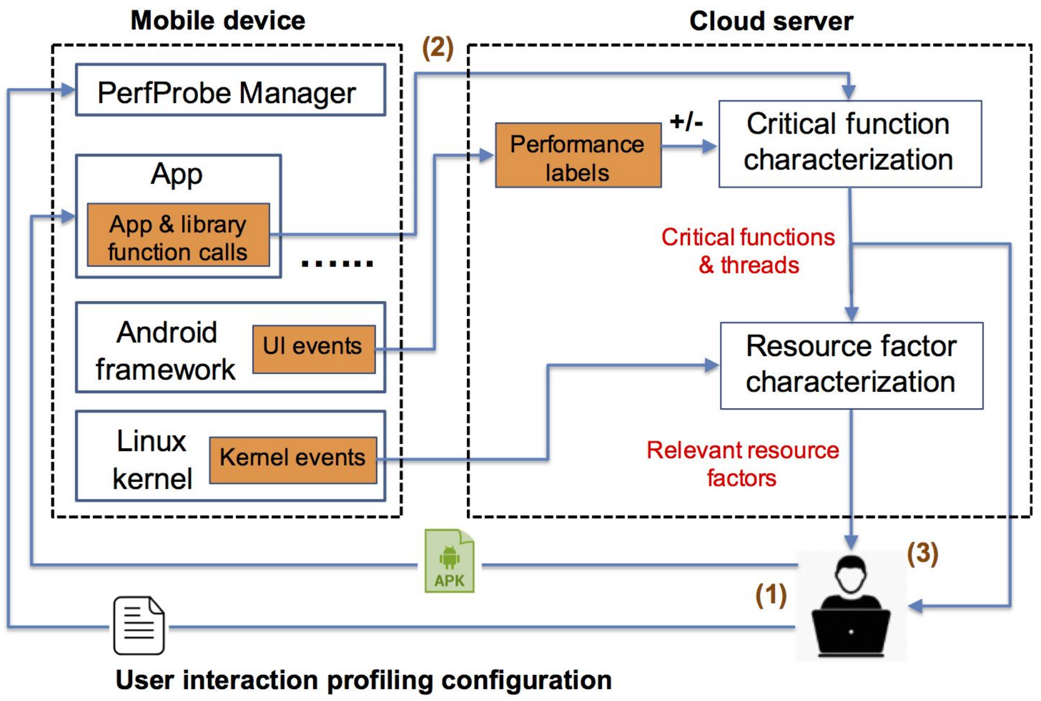
PerfProbe overview



On-device: runtime profiling for **performance monitoring**

1. App's call stack
2. UI event trace
3. OS event trace

PerfProbe overview


Mobile device
Cloud server
(2)
PerfProbe Manager
App
App & library function calls
......
Android framework
UI events
Linux kernel
Kernel events
Performance labels
+/-
Critical function characterization
Critical functions & threads
Resource factor characterization
Relevant resource factors
APK
(1)
(3)
User interaction profiling configuration

On-device: runtime profiling for **performance monitoring**

1. App's call stack
2. UI event trace
3. OS event trace

Server-side: offline trace analysis for **problem diagnosis**

Research contribution

- Low-overhead, cross-layer **runtime monitoring**
    - *Sampling frequency adaptation for app profiling* along execution to limit the performance overhead
- **Problem diagnosis** through associating app and OS-layer runtime events
    - *Trace analysis based on decision tree learning* to pinpoint both code and system-level diagnosis hints

Runtime performance monitoring

- Android UI framework instrumentation
  - To measure user-perceived latency
- Traceview [1]
  - Time spent in each function at an app’s call stack
  - Code-level function execution
- Panappticon [2]
  - OS events over time on each thread during execution
  - System resource usage

[1] Android Traceview. https://developer.android.com/studio/profile/traceview.html
[2] Panappticon: Event-Based Tracing to Optimize Mobile Application and Platform Performance. In CODES+ISSS ’13.

High overhead with app profiling

- Observation on call stack sampling in Traceview
  - Android runtime periodically pauses all threads of an app to dump its call stack => extra app latency (> 20% increase)
- **Relative profiling overhead *O(n)***: percentage of increase in app latency due to a pause for sampling
  - P(n): observed app pause duration in nth sampling round
  - S(n): sampling period in nth sampling round

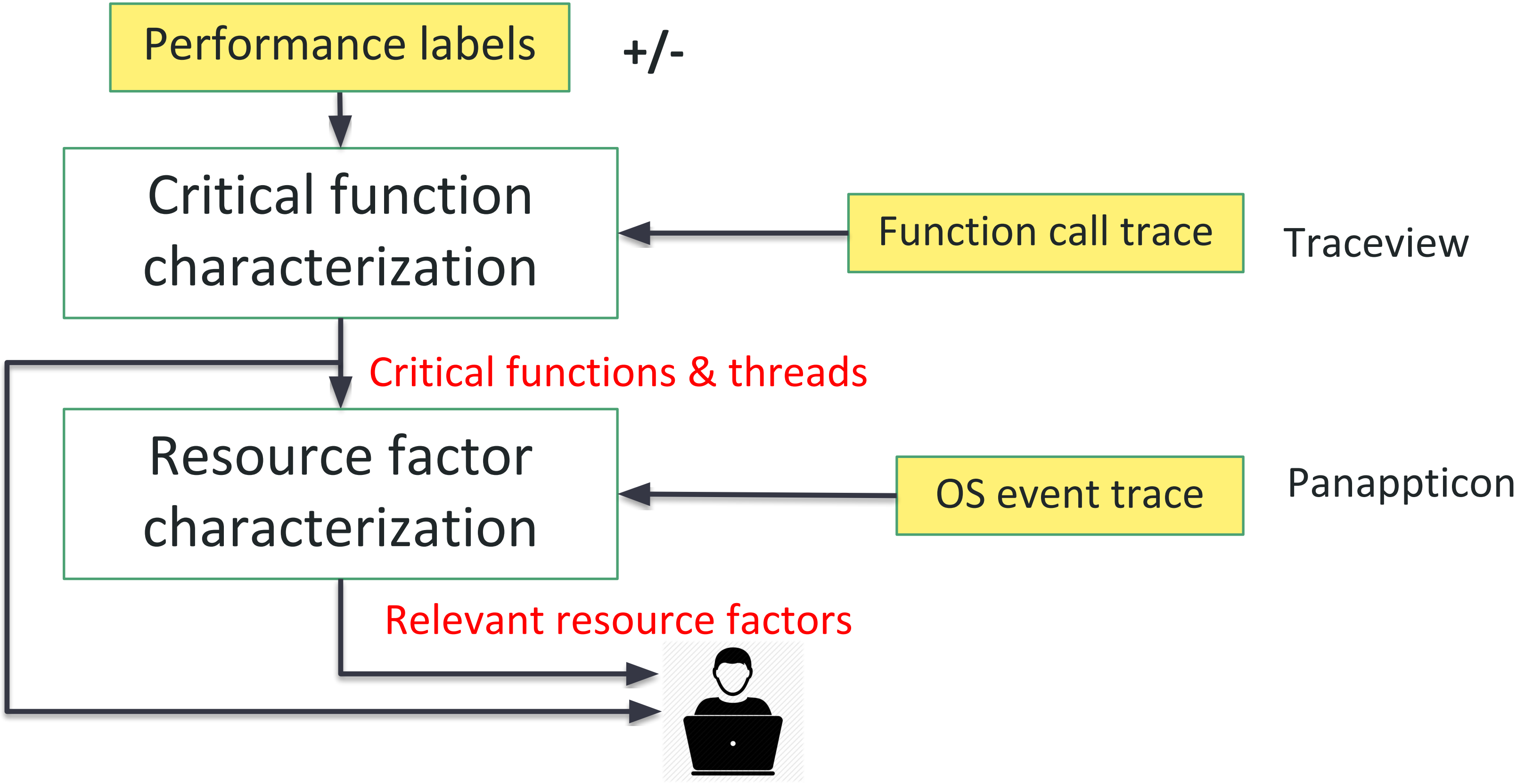
$$O(n) = \frac{P(n)}{S(n)+P(n)}$$

Sampling frequency adaptation

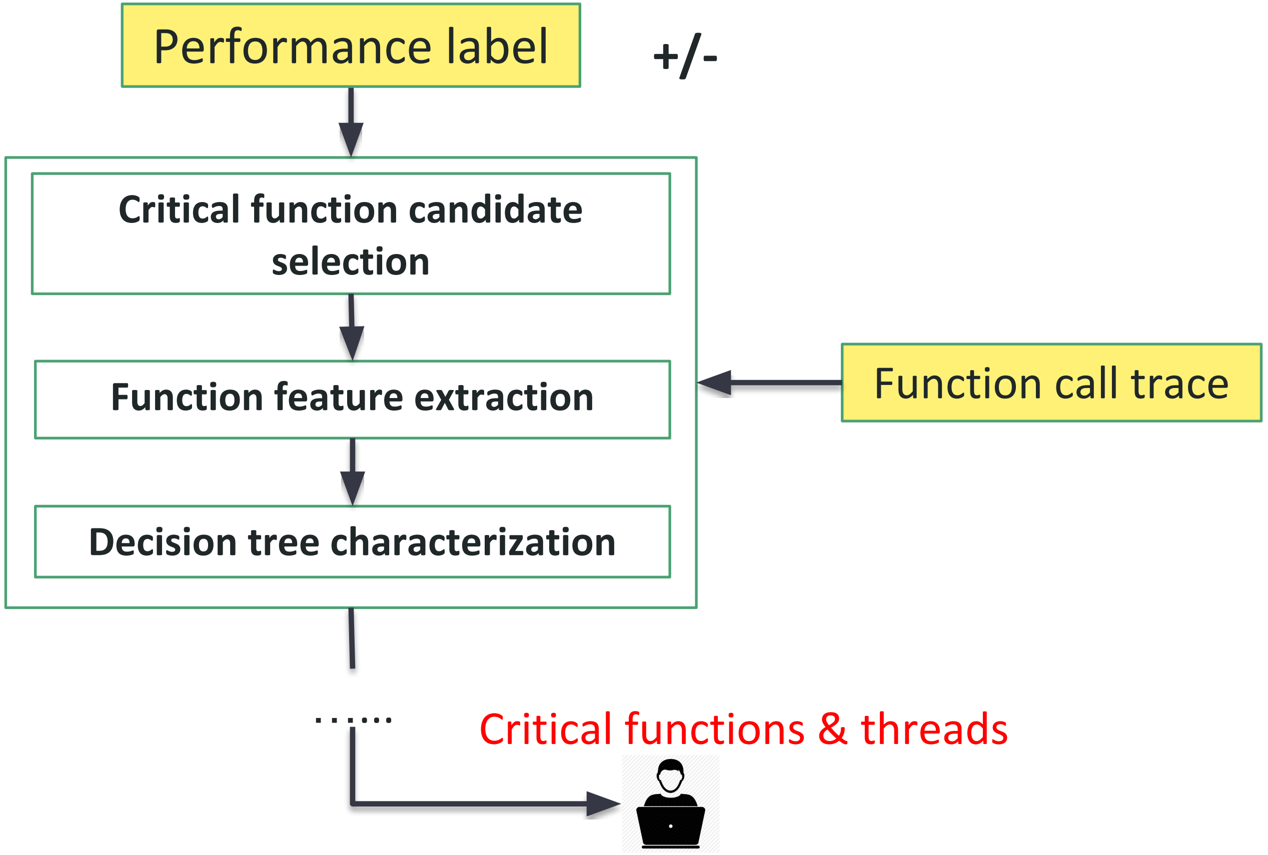
- Adaptation of an app’s call stack sampling frequency to maintain low overhead along app execution
  - A configurable bound T for relative overhead ($0 < T \leq 1$).

$$S(n+1) = \begin{cases} max(S(n), P(n), \frac{P(n)}{T} - P(n)), \text{if high load} \\ max(P(n), min(S(n), \frac{P(n)}{T} - P(n))), \text{otherwise} \end{cases}$$

Problem diagnosis

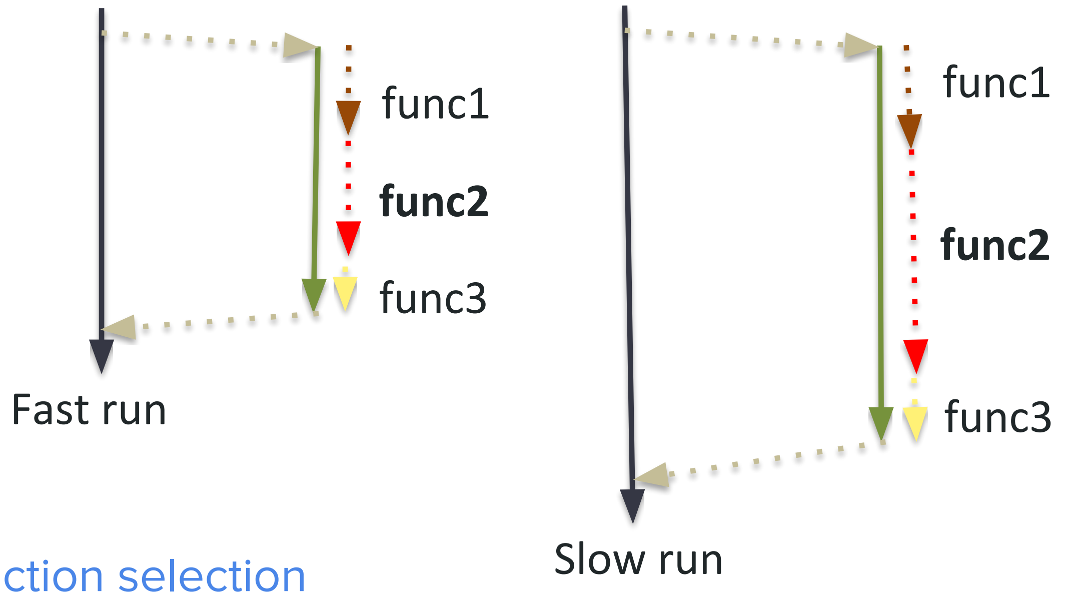

Performance labels
+/-
Critical function characterization
Function call trace
Traceview
Critical functions & threads
Resource factor characterization
OS event trace
Panappticon
Relevant resource factors

Step 1: critical function characterization


Performance label
+/-
Critical function candidate selection
Function feature extraction
Decision tree characterization
Function call trace
……
Critical functions & threads

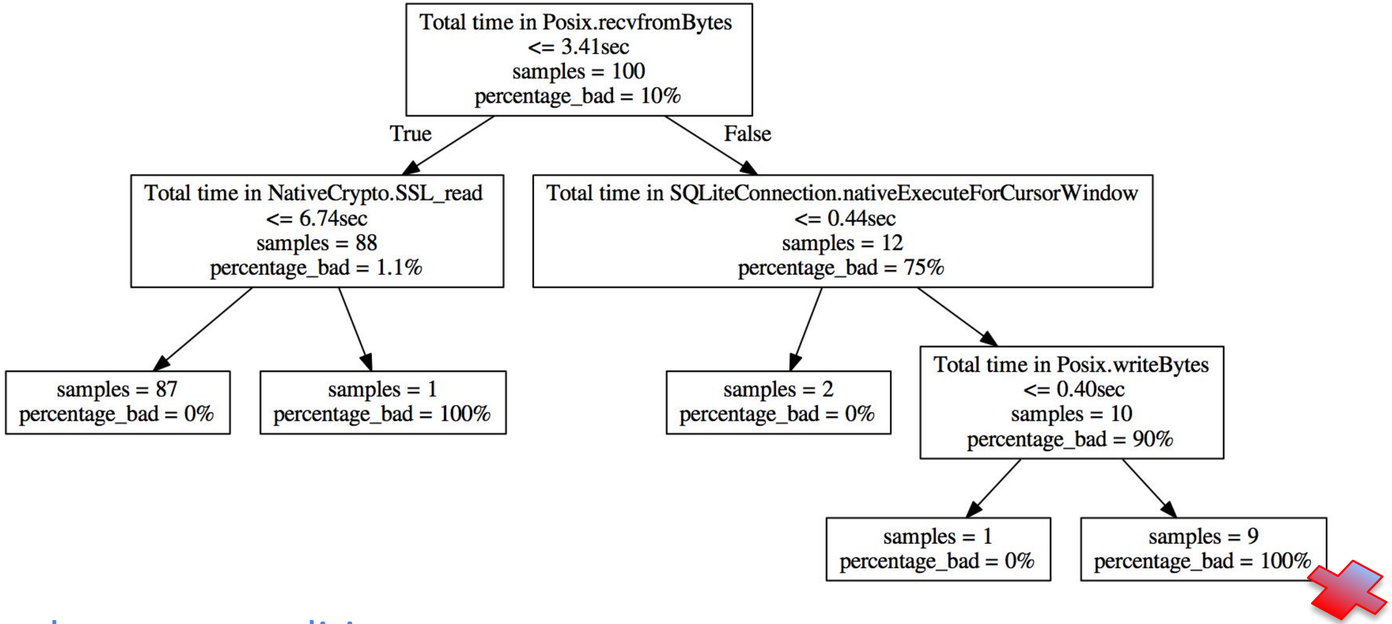
Critical function characterization

Property of critical functions
- **Time-consuming**
- **Most correlated to the performance slowdown**



Decision tree based critical function selection
- **Input features**: total time spent in a function
- **Input label**: indicator of performance slowdown

Critical function characterization

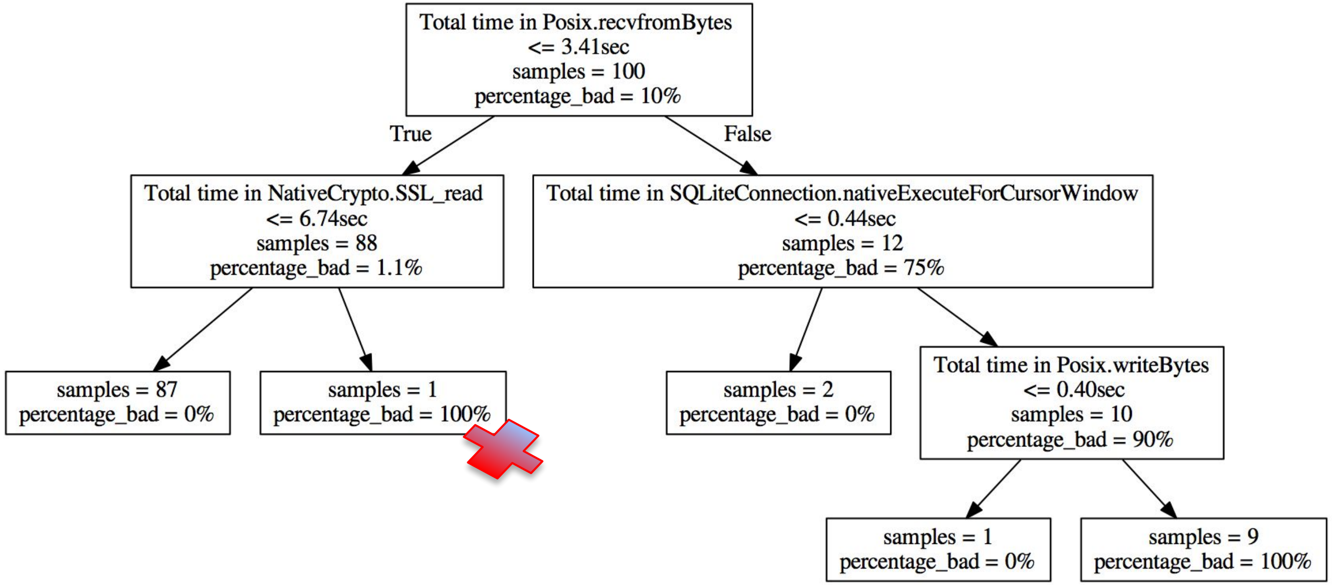


Total time in Posix.recvfromBytes <= 3.41sec samples = 100 percentage_bad = 10%
True
False
Total time in NativeCrypto.SSL_read <= 6.74sec samples = 88 percentage_bad = 1.1%
Total time in SQLiteConnection.nativeExecuteForCursorWindow <= 0.44sec samples = 12 percentage_bad = 75%
samples = 87 percentage_bad = 0%
samples = 1 percentage_bad = 100%
samples = 2 percentage_bad = 0%
Total time in Posix.writeBytes <= 0.40sec samples = 10 percentage_bad = 90%
samples = 1 percentage_bad = 0%
samples = 9 percentage_bad = 100%

Slowdown preconditions:

1) **recvfromBytes > 3.41sec** AND nativeExecuteForCursorWindow > 0.44sec AND writeBytes > 0.40sec

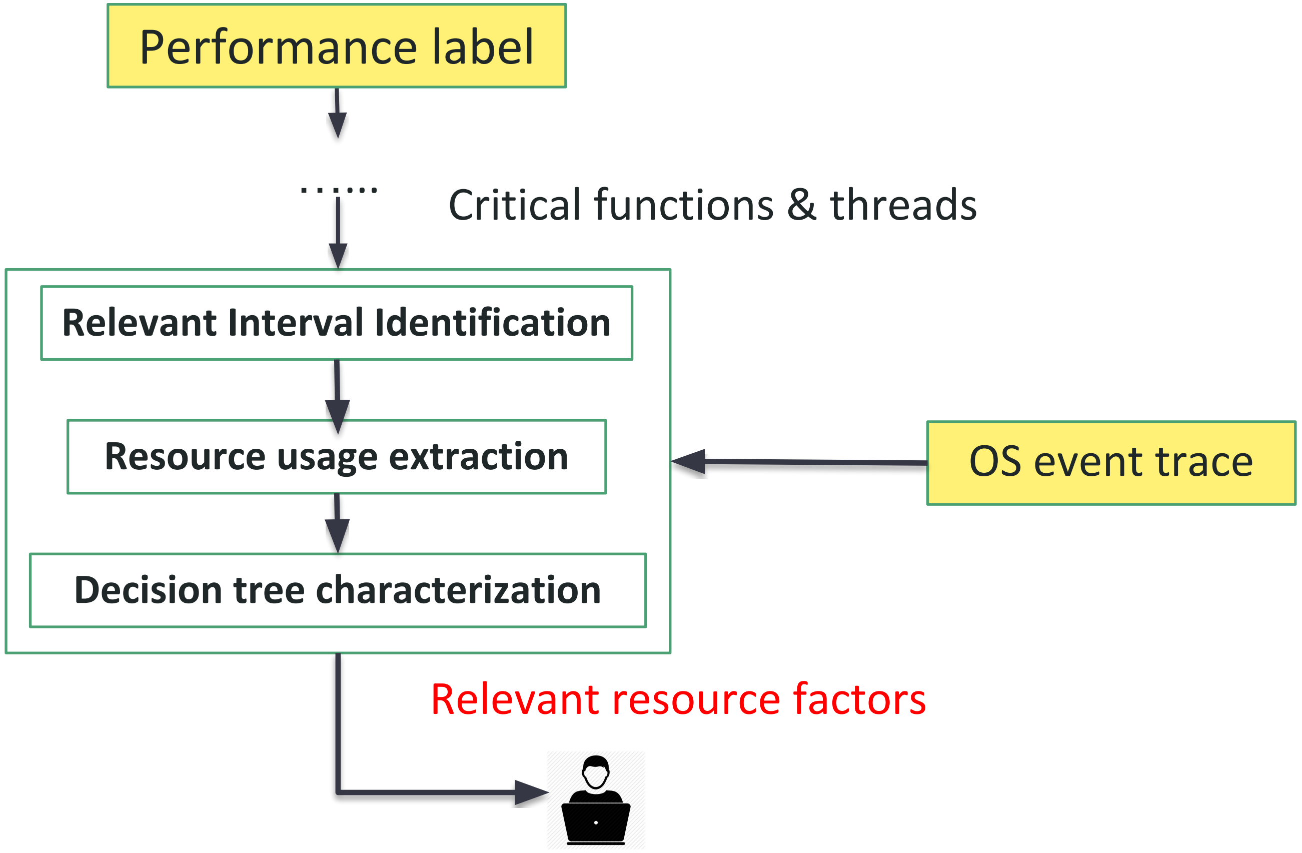
Critical function characterization


Total time in Posix.recvfromBytes <= 3.41sec samples = 100 percentage_bad = 10%
True
False
Total time in NativeCrypto.SSL_read <= 6.74sec samples = 88 percentage_bad = 1.1%
Total time in SQLiteConnection.nativeExecuteForCursorWindow <= 0.44sec samples = 12 percentage_bad = 75%
samples = 87 percentage_bad = 0%
samples = 1 percentage_bad = 100%
samples = 2 percentage_bad = 0%
Total time in Posix.writeBytes <= 0.40sec samples = 10 percentage_bad = 90%
samples = 1 percentage_bad = 0%
samples = 9 percentage_bad = 100%

Slowdown preconditions:

1) recvfromBytes > 3.41sec AND nativeExecuteForCursorWindow > 0.44sec AND writeBytes > 0.40sec
2) recvfromBytes <= 3.41sec AND **SSL_read > 6.74sec**

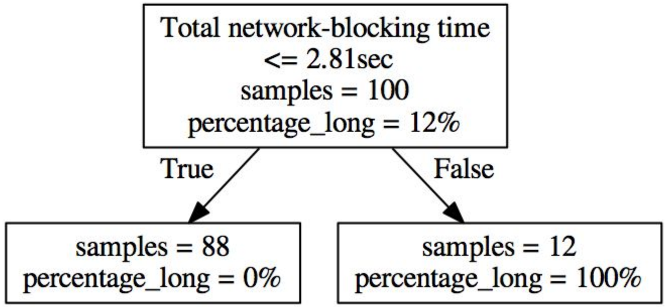
Step 2: resource factor characterization


Performance label
......
Critical functions & threads
Relevant Interval Identification
Resource usage extraction
Decision tree characterization
OS event trace
Relevant resource factors

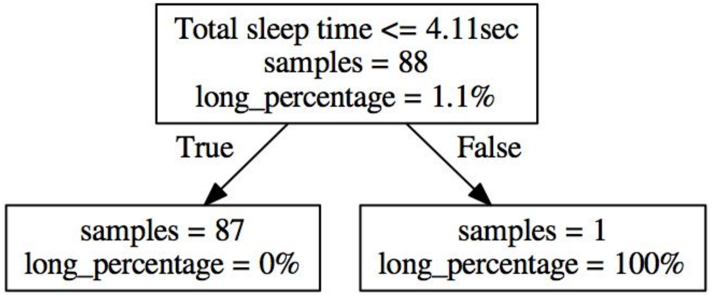
Relevant resource factors for a critical function

- Resource usage for a critical function
  - **Relevant interval** I^m_t: time intervals when a critical function *m* is invoked by thread *t*
  - Compute resource usage under all I^m_t for function *m*
- Decision tree based resource factor selection
  - **Input features**: usage on various types of resource
  - **Input label**: indicator of critical function slowdown
  - **Output tree nodes** => relevant resource factors

Relevant resource factors



Posix.recvfromBytes



NativeCrypto.SSL_read

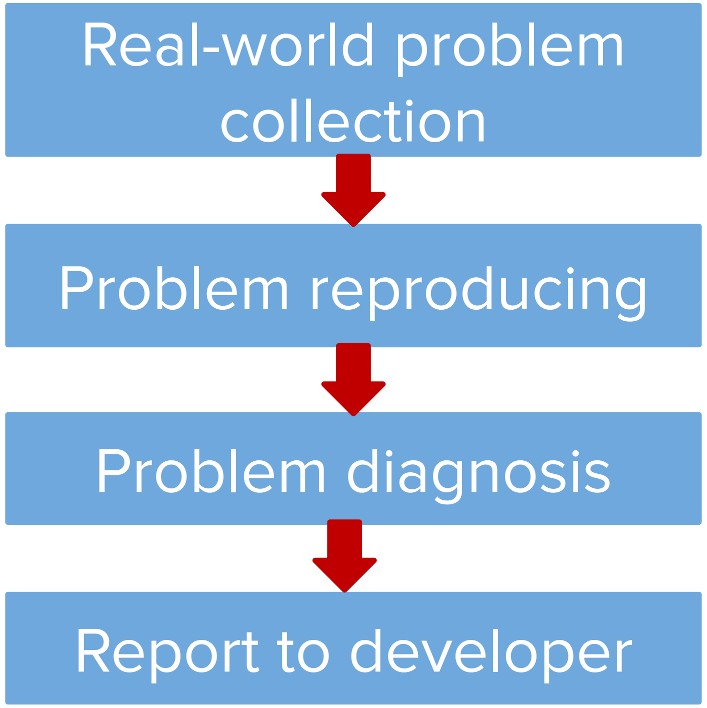
Longer time blocking for network I/O -> network factor

Longer time in interruptible sleep -> I/O event delay

Experiment results summary

- Cross-layer profiling incurs < 3.5% increase of delay
  - Traceview's sampling profiling incurs up to 22% increase
- Performed diagnosis on 22 popular Android apps
  - **Relevant resource factors**: network/server, CPU, disk I/O
  - **Cross-layer vs. pure resource profiling**: pinpointed true relevant resource factors in 8 apps
- Android app developer study
  - iNaturalist app developer acknowledged our diagnosis and **adopted our problem fixing direction (10x speedup)**

Real-world app developer study



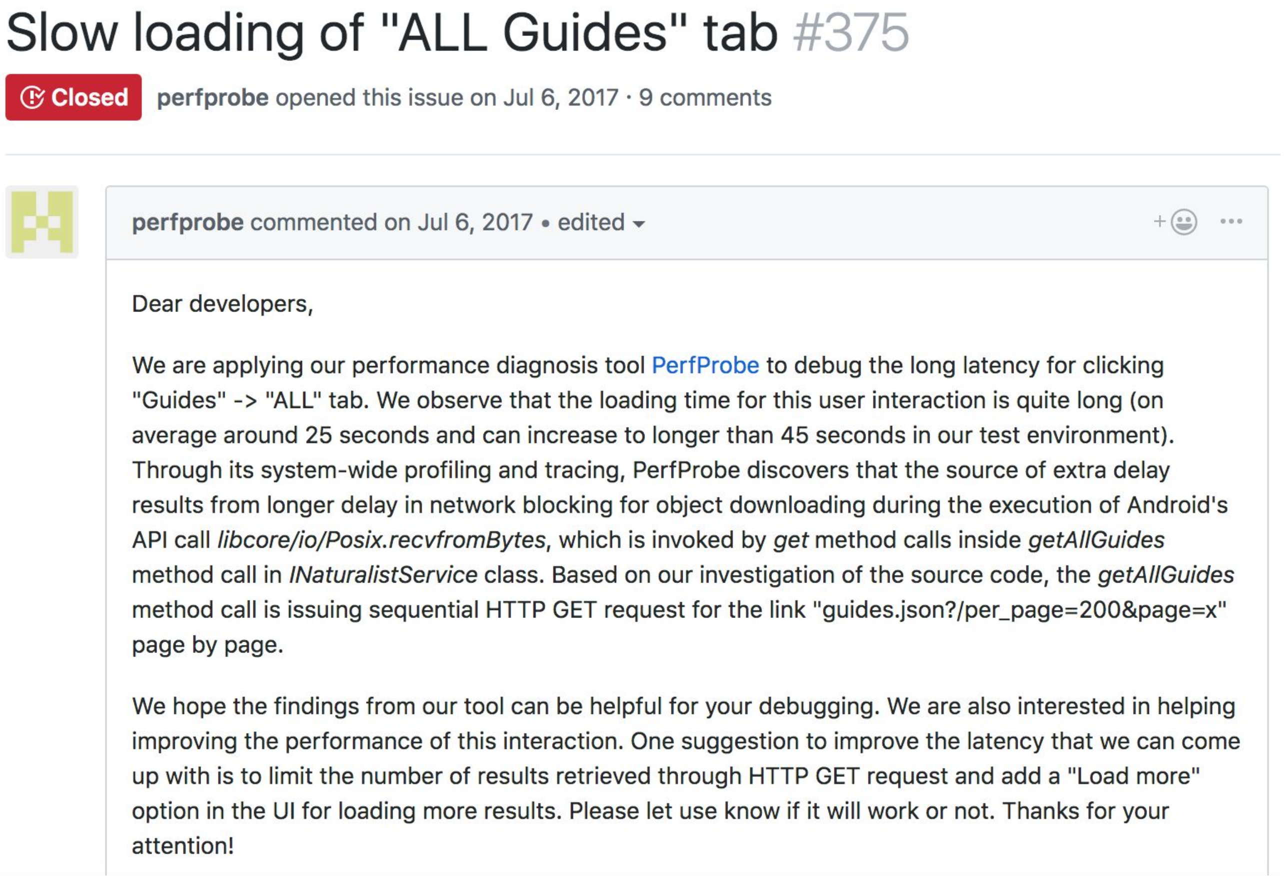
Crawl user-reported performance problems from issue trackers

Repeated testing of related interactions

PerfProbe’s cross-layer diagnosis finding

Collect developer’s feedback for tool evaluation

iNaturalist case study



Slow loading of "ALL Guides" tab #375

Closed perfprobe opened this issue on Jul 6, 2017 · 9 comments

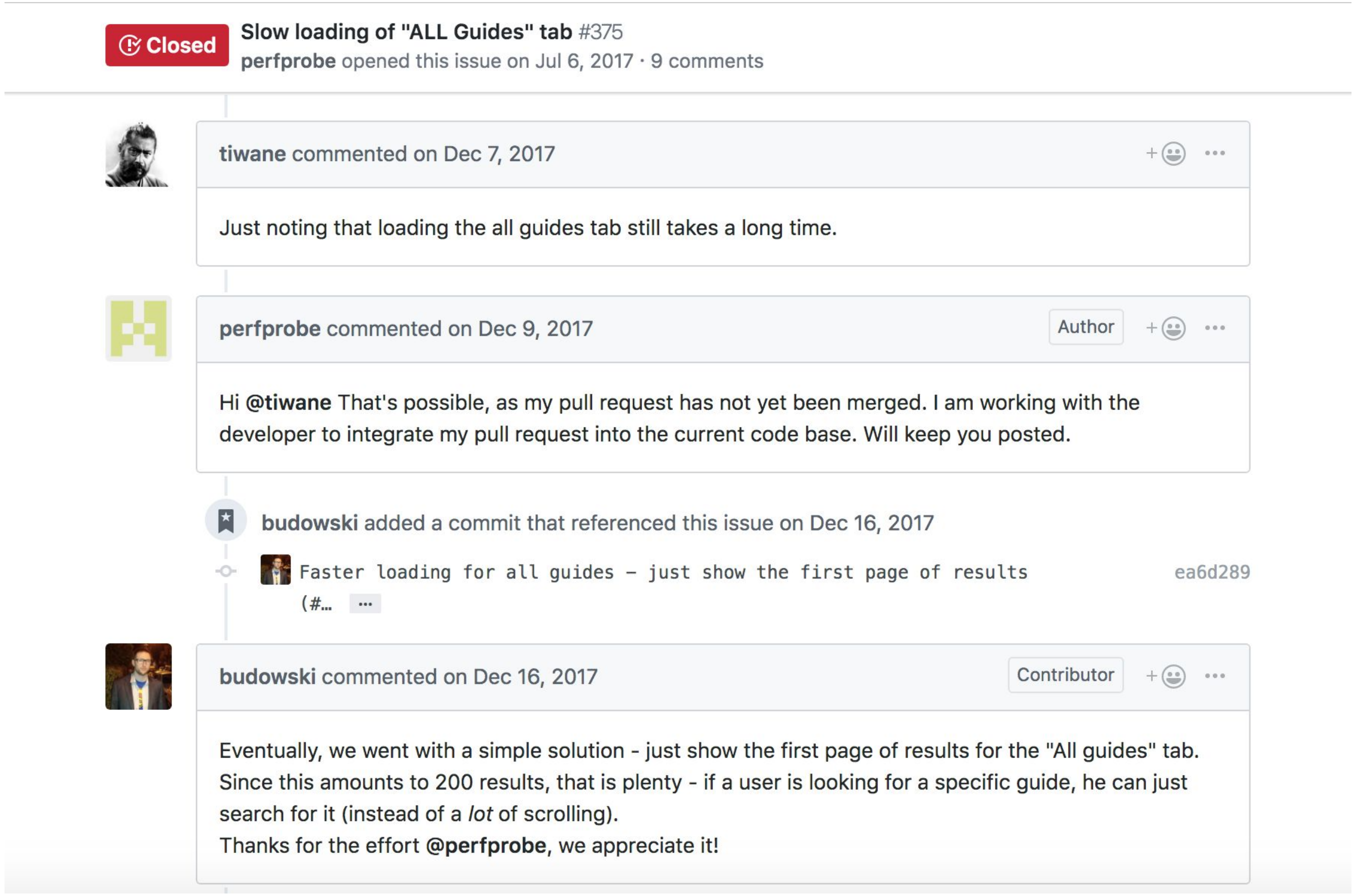
perfprobe commented on Jul 6, 2017 • edited

Dear developers,

We are applying our performance diagnosis tool PerfProbe to debug the long latency for clicking "Guides" -> "ALL" tab. We observe that the loading time for this user interaction is quite long (on average around 25 seconds and can increase to longer than 45 seconds in our test environment). Through its system-wide profiling and tracing, PerfProbe discovers that the source of extra delay results from longer delay in network blocking for object downloading during the execution of Android's API call *libcore/io/Posix.recvfromBytes*, which is invoked by *get* method calls inside *getAllGuides* method call in *INaturalistService* class. Based on our investigation of the source code, the *getAllGuides* method call is issuing sequential HTTP GET request for the link "guides.json?/per_page=200&page=x" page by page.

We hope the findings from our tool can be helpful for your debugging. We are also interested in helping improving the performance of this interaction. One suggestion to improve the latency that we can come up with is to limit the number of results retrieved through HTTP GET request and add a "Load more" option in the UI for loading more results. Please let use know if it will work or not. Thanks for your attention!

iNaturalist case study



Closed **Slow loading of "ALL Guides" tab** #375

perfprobe opened this issue on Jul 6, 2017 · 9 comments

tiwane commented on Dec 7, 2017

Just noting that loading the all guides tab still takes a long time.

perfprobe commented on Dec 9, 2017 Author

Hi **@tiwane** That's possible, as my pull request has not yet been merged. I am working with the developer to integrate my pull request into the current code base. Will keep you posted.

budowski added a commit that referenced this issue on Dec 16, 2017

```
Faster loading for all guides - just show the first page of results      ea6d289
(#… …
```

budowski commented on Dec 16, 2017 Contributor

Eventually, we went with a simple solution - just show the first page of results for the "All guides" tab. Since this amounts to 200 results, that is plenty - if a user is looking for a specific guide, he can just search for it (instead of a *lot* of scrolling).
Thanks for the effort **@perfprobe**, we appreciate it!

Conclusion

- PerfProbe as a mobile diagnosis framework for ***unpredictable performance problems***
- PerfProbe performs ***low-overhead, cross-layer*** monitoring and trace collection
- PerfProbe perfomrs ***cross-layer trace analysis*** for performance problem diagnosis

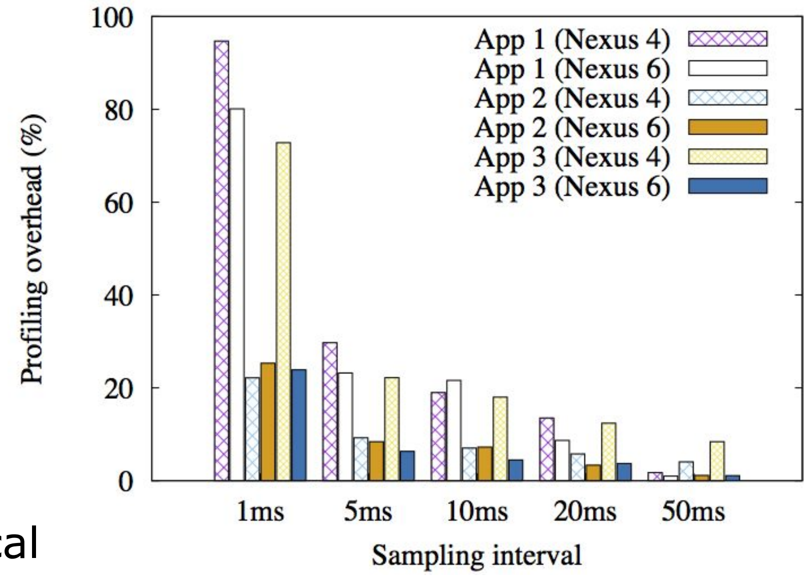
Q & A

Thank You

High overhead with app profiling

- Traceview with sampling of call stack

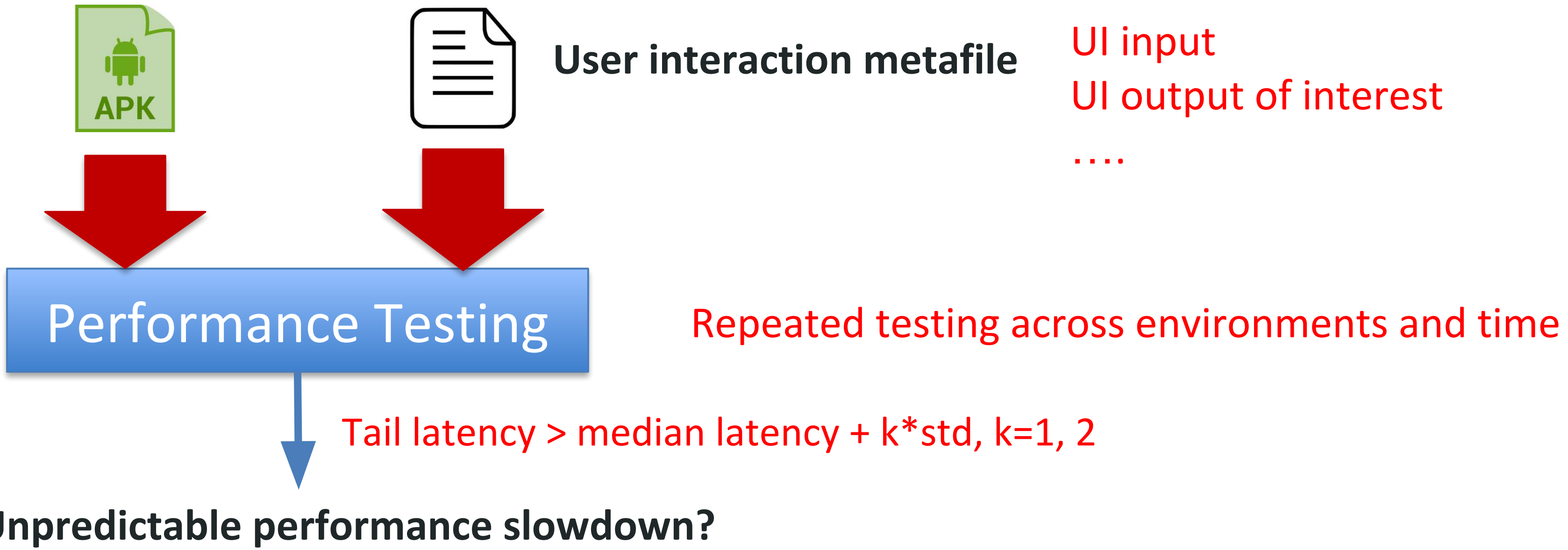
Setting a proper sampling frequency requires app and device-specific profiling



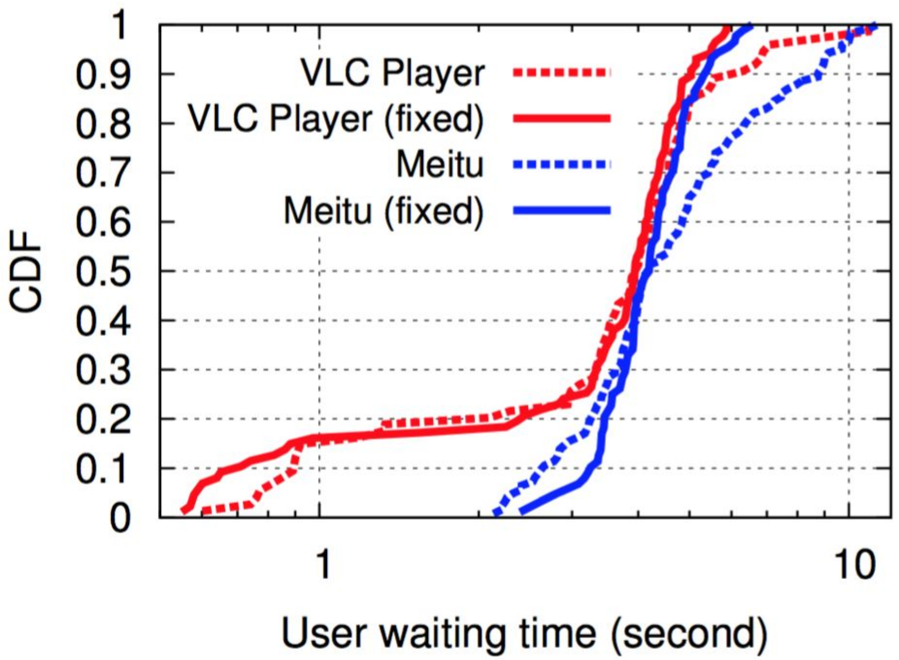
App 1, 2, 3 performing similar optical character recognition workload

Usage-triggered monitoring

- Configuration interface


APK
User interaction metafile
UI input
UI output of interest
….
Performance Testing
Repeated testing across environments and time
Tail latency > median latency + k*std, k=1, 2
Unpredictable performance slowdown?

Relevant resource factors on disk I/O



Diagnosis findings: slowdown due to disk I/O on Nexus 4

Fixing: increasing the size of read-ahead buffer

Tail user waiting time reduced:
by 45% (to < 6sec) for VLC Player
by 42% (to < 7sec) for Meitu

Diagnosing user-reported problems

App	Interaction	Root cause findings
K9 mail	Sync mailbox	IMAP connection loss
iNaturalist	Click All Guides	Too many web requests
Riot	Load a directory	Computation bound for large bitmap loading
cgeo	Search nearby cache	Sequential network requests
GeoHashDroid	Launch app	GPS signal handling
TomaHawk	Search songs	Dependency on web requests

Developers invites us to implement proposed improvement (iNaturalist and Riot app)