



Заряжаемся метриками

Как измерить батарею



Немного обо мне

- 3 года в android разработке
- преподаю в ИТМО

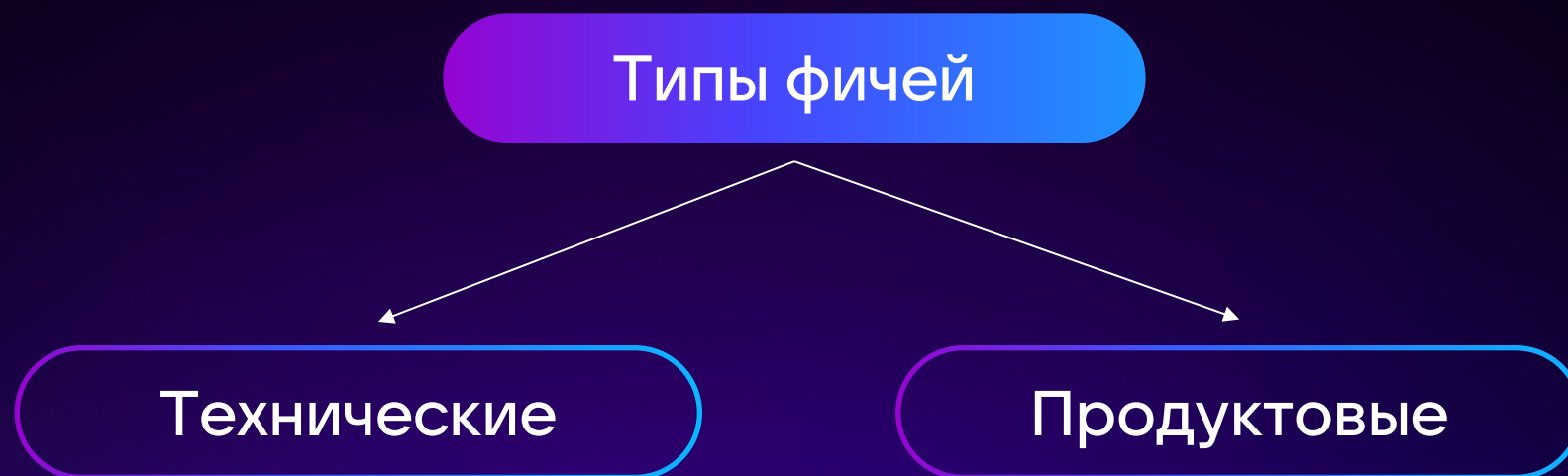
02



Зачем нужно измерять энергопотребление?

Универсальность

04



Сегодня в программе

05

01 Подготовим
ингредиенты

02 Научимся
с ними
работать

03 Набиваем
шишки
на первом
приседании

04 Финализируем
и проводим АБ

На что не будет ответов в этом докладе

Никаких best
practices
по оптимизациям

Только
программный
подход

Доклад
не про «золотой
молоток»

А сколько наше приложение потратило батареи?

Когда метрика покажет себя лучше всего

09

Это статистика, поэтому:



Чем больше замеров,
тем лучше



Чем дольше каждый
замер, тем лучше

Давайте просто посчитаем проценты

Случайные шумы метрики — то, на что мы никак не можем повлиять:

Вендоры

Внешние
факторы

Операционная
система

Почему важен объем данных и их «качество»?

Может, всё-таки проценты?

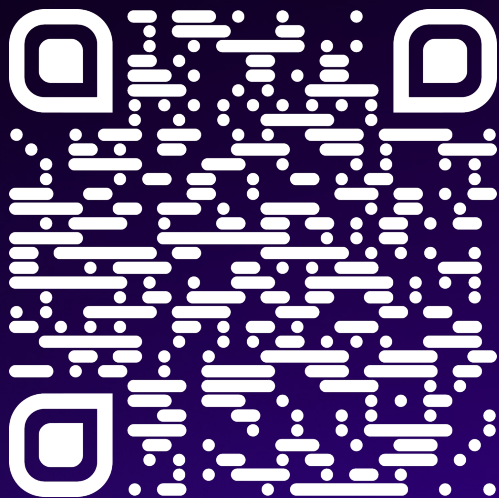
```
batteryManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY)
```

- Малый набор возможных значений
- Нелинейная функция разряда от реальной энергии

Подбираем другие ингредиенты

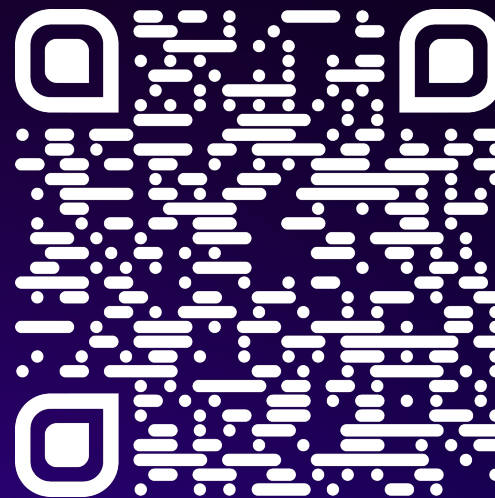
Отличные доклады про энергопотребление

15



Никита Васильченко
«Анализ энергопотребления»

[ссылка](#)

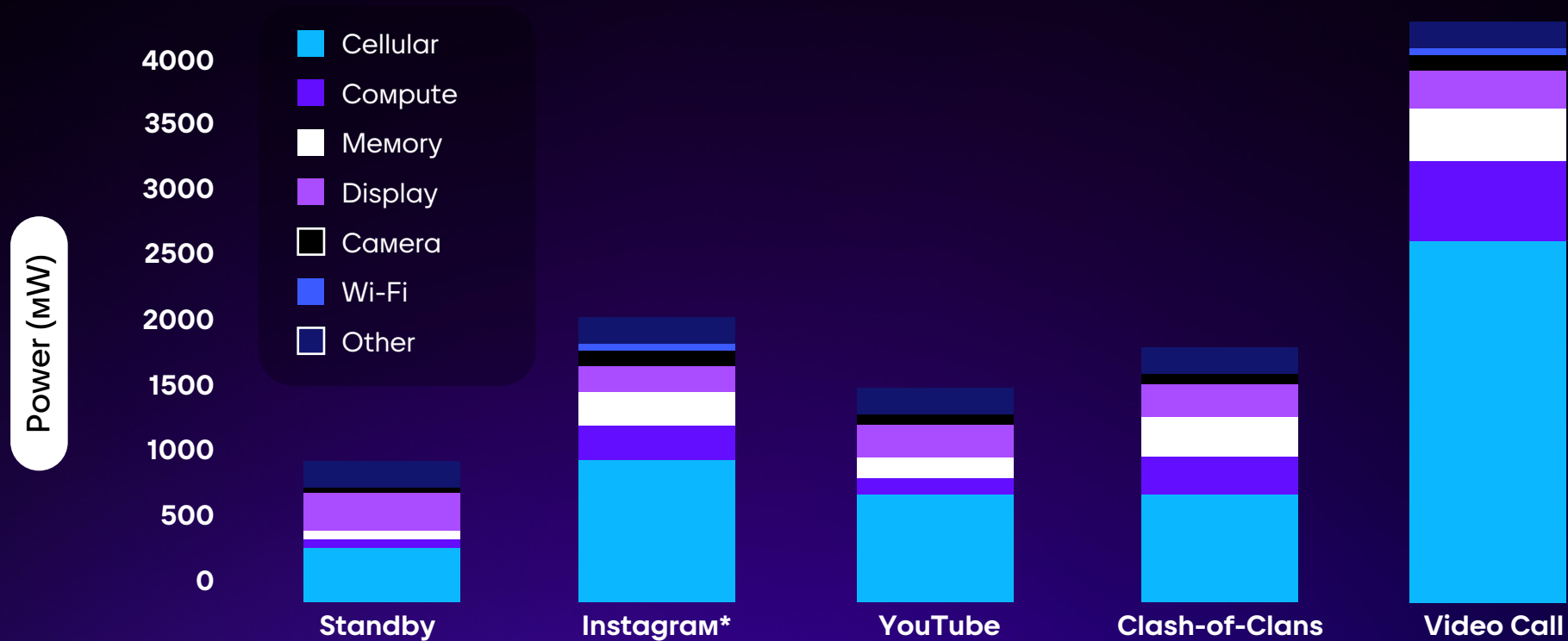


Дмитрий Мельников
«Энергопотребление»

[ссылка](#)

"3 W's of smartphone power consumption: Who, Where and How much is draining my battery?"

16



*Признан экстремистской организацией
и запрещён на территории РФ

CPU ТИКИ

```
adb shell cat /proc/{pid}/stat
```

17

- (14) `utime %lu`
Amount of time that this process has been scheduled in user mode, measured in clock ticks (divide by `sysconf(_SC_CLK_TCK)`). This includes guest time, `guest_time` (time spent running a virtual CPU, see below), so that applications that are not aware of the guest time field do not lose that time from their calculations.

- (15) `stime %lu`
Amount of time that this process has been scheduled in kernel mode, measured in clock ticks (divide by `sysconf(_SC_CLK_TCK)`).

- (16) `cutime %ld`
Amount of time that this process's waited-for children have been scheduled in user mode, measured in clock ticks (divide by `sysconf(_SC_CLK_TCK)`). (See also `times(2)`.) This includes guest time, `cguest_time` (time spent running a virtual CPU, see below).

- (17) `cstime %ld`
Amount of time that this process's waited-for children have been scheduled in kernel mode, measured in clock ticks (divide by `sysconf(_SC_CLK_TCK)`).

CPU тики

18

Итого отсылаем:

Тики

`Os.sysconf(OsConstants._SC_CLK_TCK)`

Количество ядер

Ценность CPU тиков

19

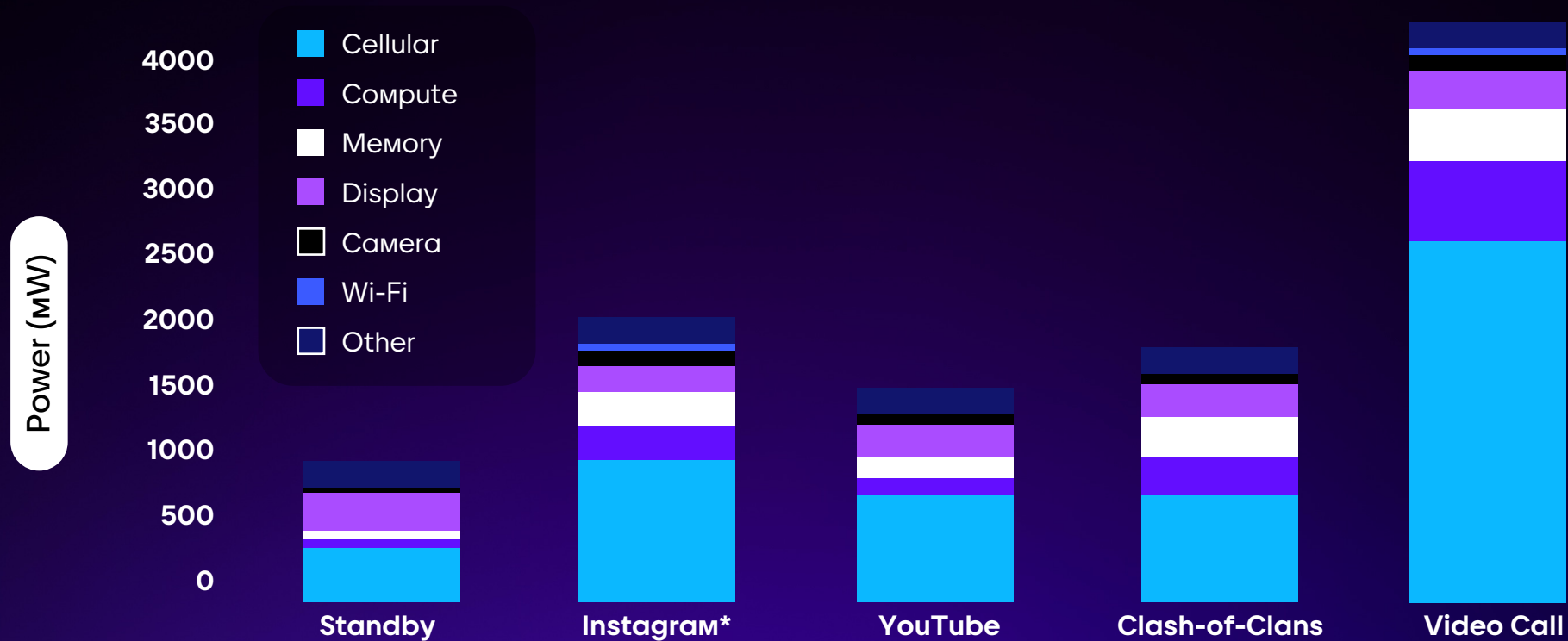
Анализ:

Технические/
продуктовые фичи

Kernel/user стороны

"3 W's of smartphone power consumption: Who, Where and How much is draining my battery?"

20



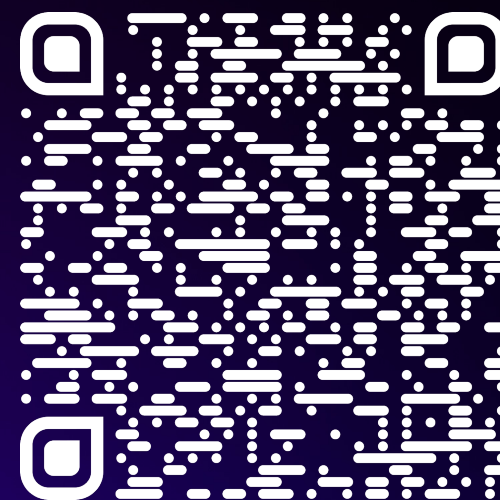
*Признан экстремистской организацией
и запрещён на территории РФ

Сравнение радиомодулей

21

	α_u (mW/Mbps)	α_d (mW/Mbps)	β (mW)	α_u/α_d
LTE	438.39	51.97	1288.04	8.44
3G	868.98	122.12	817.88	7.12
Wi-Fi	283.17	137.01	132.86	2.07

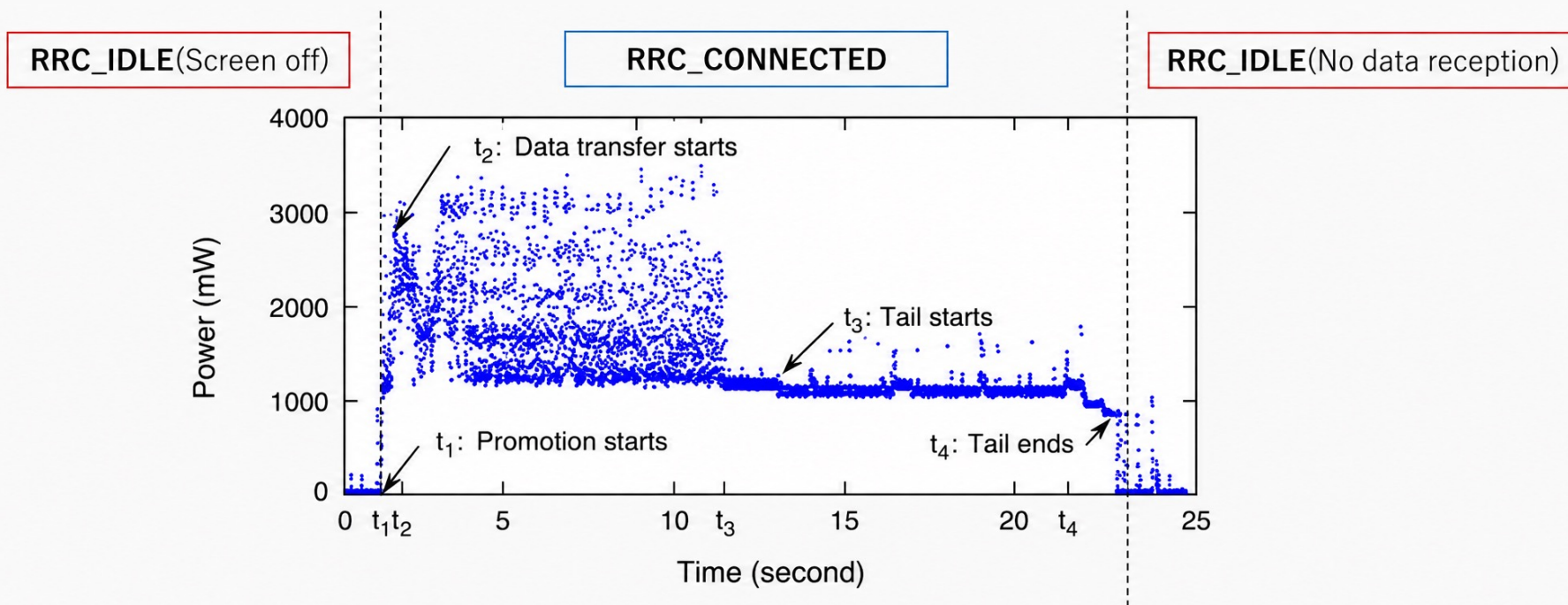
Table 4: Data transfer power model



[ССЫЛКА](#)

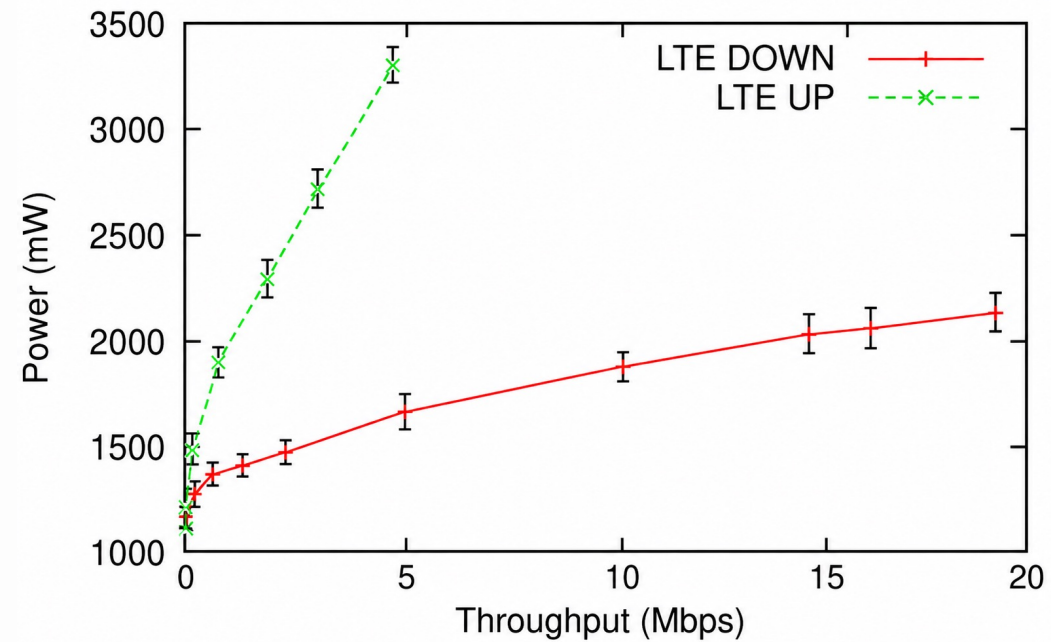
Влияние сетевого IO

22



Влияние сетевого IO

25



Что в итоге собираем?

24



Отдельно
wifi/lte трафик



Сколько
получили,
отправили байт



Сколько модули
были в idle режиме

Как получать все эти значения?

HealthStats

```
systemHealthManager.takeMyUidSnapshot()
```

```
rxBytes = healthStats.readMeasurement(UidHealthStats.MEASUREMENT_MOBILE_RX_BYTES)
```

```
txBytes = healthStats.readMeasurement(UidHealthStats.MEASUREMENT_MOBILE_TX_BYTES)
```

```
idleMs = healthStats.readMeasurement(UidHealthStats.MEASUREMENT_MOBILE_IDLE_MS)
```

TrafficStats

```
rxBytes = TrafficStats.getUidRxBytes(uid).coerceAtLeast(0L)  
txBytes = TrafficStats.getUidTxBytes(uid).coerceAtLeast(0L)  
idleMs = 0L
```

Ценность ингредиента сетевого трафика

28



Rx/Tx —
неоптимальный кеш,
проблемное API



Idle — отсутствие
батчинга, неудачный
паттерн запросов



Wifi/Lte — инсайтов
не дает

Давайте проще

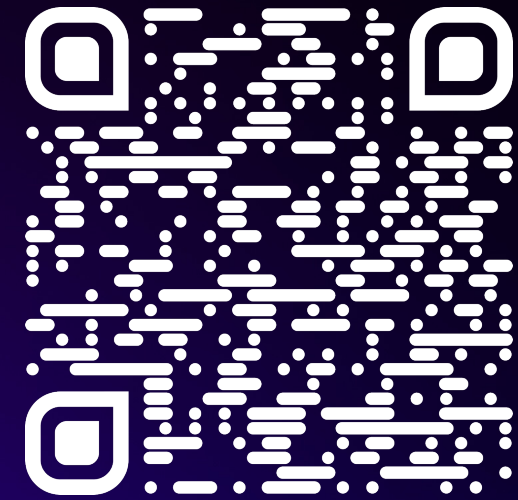
Итоговый скоринг

30

```
score = a * cpu / time + b * network / time
```

Measure Power Values

wifi.on	Additional power used when Wi-Fi is turned on but not receiving, transmitting, or scanning	2 mA
wifi.active	Additional power used when transmitting or receiving over Wi-Fi	31 mA
wifi.scan	Additional power used when Wi-Fi is scanning for access point	100 mA



[ССЫЛКА](#)

Итоговый скоринг

```
score = cpuLoad·1.0  
        + wifiTxLoad·0.35 + wifiRxLoad·0.25 + wifiIdleLoad·0.03  
        + mobileTxLoad·1.20 + mobileRxLoad·0.85 + mobileIdleLoad·0.08
```

Преимущества скоринга

Сильно режется
случайный шум

В разы чувствительнее
к изменениям

Результаты
сравнения в 1 числе

Легко интерпретировать

Начинаем собирать

Требования

35

Девайсов у нас очень много

Собирать необходимо всю сессию

Необходимо минимизировать систематические шумы

Архитектура сбора

Интервальный сбор («слайс»), пока процесс жив

Сохраняем персистентно в базу

Восстанавливаем замеры, агрегируем, отсылаем в аналитику

Систематические шумы

37

Разные сценарии
использования

Разные девайсы
по железу

Разные батареи
по состоянию

Самый главный разрез — видимость приложения

38

Если мы в форграунде

у нас самый высокий
приоритет с точки
зрения ОС

Если мы в бекграунде

ОС начинает нас
пессимизировать

Архитектура сбора V2

Интервальный сбор («слайс»), пока процесс жив

Сохраняем персистентно в базу

Восстанавливаем замеры, агрегируем, отправляем в аналитику

Добавляем слайсы при переходе из FG в BG, и обратно

Останавливаем сбор при обнаружении зарядки

Из чего состоит слайс?

```
public data class BatterySnapshot(  
    val sliceTime: Long,  
    val utime: Long,  
    val stime: Long,  
    val batteryPercent: Int,  
    val healthStats: HealthStats,  
    val trafficStats: TrafficStats,  
    val isBatteryOptimizationsEnabled: Boolean,  
    val isBackgroundActivityDisabled: Boolean,  
    val processes: ProcessMask,  
)
```

Из чего состоит слайс — CPU

```
public data class BatterySnapshot(  
    val sliceTime: Long,  
    val utime: Long,  
    val stime: Long,  
    val batteryPercent: Int,  
    val healthStats: HealthStats,  
    val trafficStats: TrafficStats,  
    val isBatteryOptimizationsEnabled: Boolean,  
    val isBackgroundActivityDisabled: Boolean,  
    val processes: ProcessMask,  
)
```

Из чего состоит слайс — процент батареи

```
public data class BatterySnapshot(  
    val sliceTime: Long,  
    val utime: Long,  
    val stime: Long,  
    val batteryPercent: Int,  
    val healthStats: HealthStats,  
    val trafficStats: TrafficStats,  
    val isBatteryOptimizationsEnabled: Boolean,  
    val isBackgroundActivityDisabled: Boolean,  
    val processes: ProcessMask,  
)
```

Боль 1: Шальные 300 процентов потраченного заряда

Процент заряда как лакмусовая бумага вскрыл особенности сбора:

Иногда вендор мог не прислать бродкаст о начале зарядки

Случались перекалибровки контроллеров батареи

Из чего состоит слайс — сетевые параметры

```
public data class BatterySnapshot(  
    val sliceTime: Long,  
    val utime: Long,  
    val stime: Long,  
    val batteryPercent: Int,  
    val healthStats: HealthStats,  
    val trafficStats: TrafficStats,  
    val isBatteryOptimizationsEnabled: Boolean,  
    val isBackgroundActivityDisabled: Boolean,  
    val processes: ProcessMask,  
)
```

Боль 2: Фолбэк на TrafficStats

45

HealthStats
не обновляет свои
счетчики, даже
при опросе раз
в минуту

Теряем дорогую
информацию
про трафик

Снимаем через
оба подхода.
Если HealthStats
не изменился,
пытаемся нарастить
дифф через TrafficStats

Из чего состоит слайс — разрез по настройкам в ОС

```
public data class BatterySnapshot(  
    val sliceTime: Long,  
    val utime: Long,  
    val stime: Long,  
    val batteryPercent: Int,  
    val healthStats: HealthStats,  
    val trafficStats: TrafficStats,  
    val isBatteryOptimizationsEnabled: Boolean,  
    val isBackgroundActivityDisabled: Boolean,  
    val processes: ProcessMask,  
)
```

Из чего состоит слайс — разрез по процессам

```
public data class BatterySnapshot(  
    val sliceTime: Long,  
    val utime: Long,  
    val stime: Long,  
    val batteryPercent: Int,  
    val healthStats: HealthStats,  
    val trafficStats: TrafficStats,  
    val isBatteryOptimizationsEnabled: Boolean,  
    val isBackgroundActivityDisabled: Boolean,  
    val processes: ProcessMask,  
)
```

ProcessMask — 64 возможных процесса

```
@JvmInline  
public value class ProcessMask(public val rawValue: Long) {  
    public val hasUpload: Boolean get() = hasFlag(1L shl 0)  
    public val hasDownload: Boolean get() = hasFlag(1L shl 1)  
}
```

ProcessTracker — реестр процессов

```
private val counters = AtomicIntegerArray(64)
```

Все операции атомарны, задизайнен для многопоточности

Явный start/stop для каждого процесса

Процессы — самый вариабельный разрез

50

Standalone процесс:

00010

01000

...

Комбинации
процессов:

01010

01001

...

Эталон:

00000

Процессы — необходима калибровка

Эталон может «прятать» проблемные процессы

Валидируется через минимальный скоринг среди всех процессов

Отправка события

52

Валидация

отбрасываем
скачки
batteryPercent

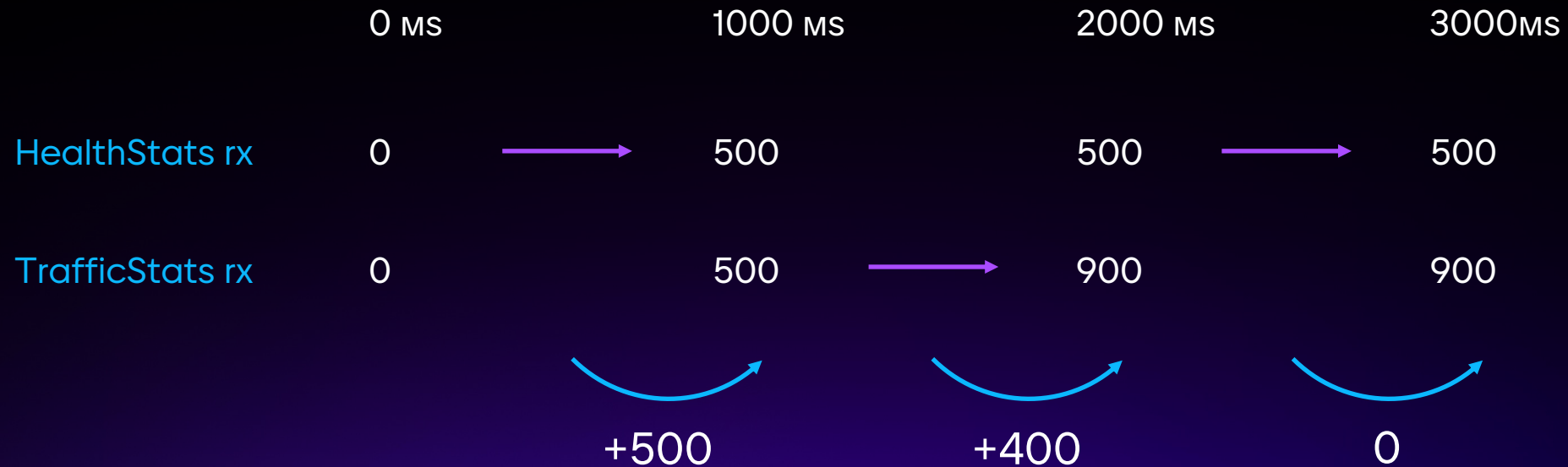
Аккумуляция

накапливаем
счетчики

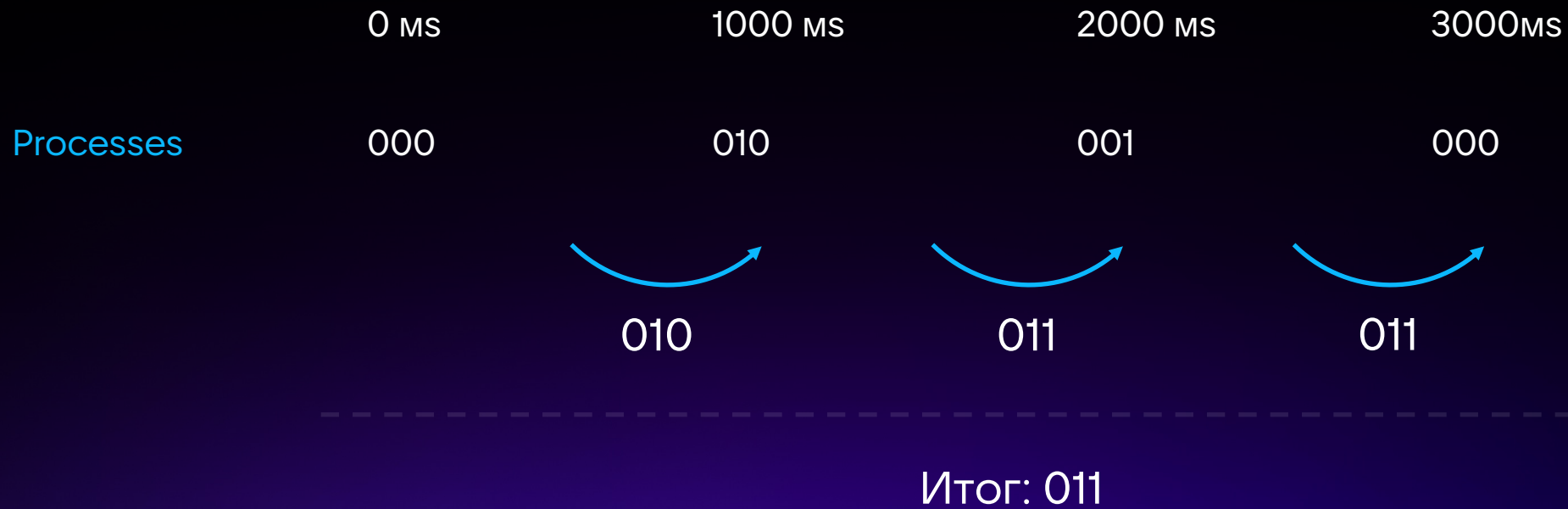
Отправка

обогащаем
постоянными
разрезами
и отправляем

Аккумуляция на примере процессов



Аккумуляция на примере процессов



Разрезы, константные между сессиями

55

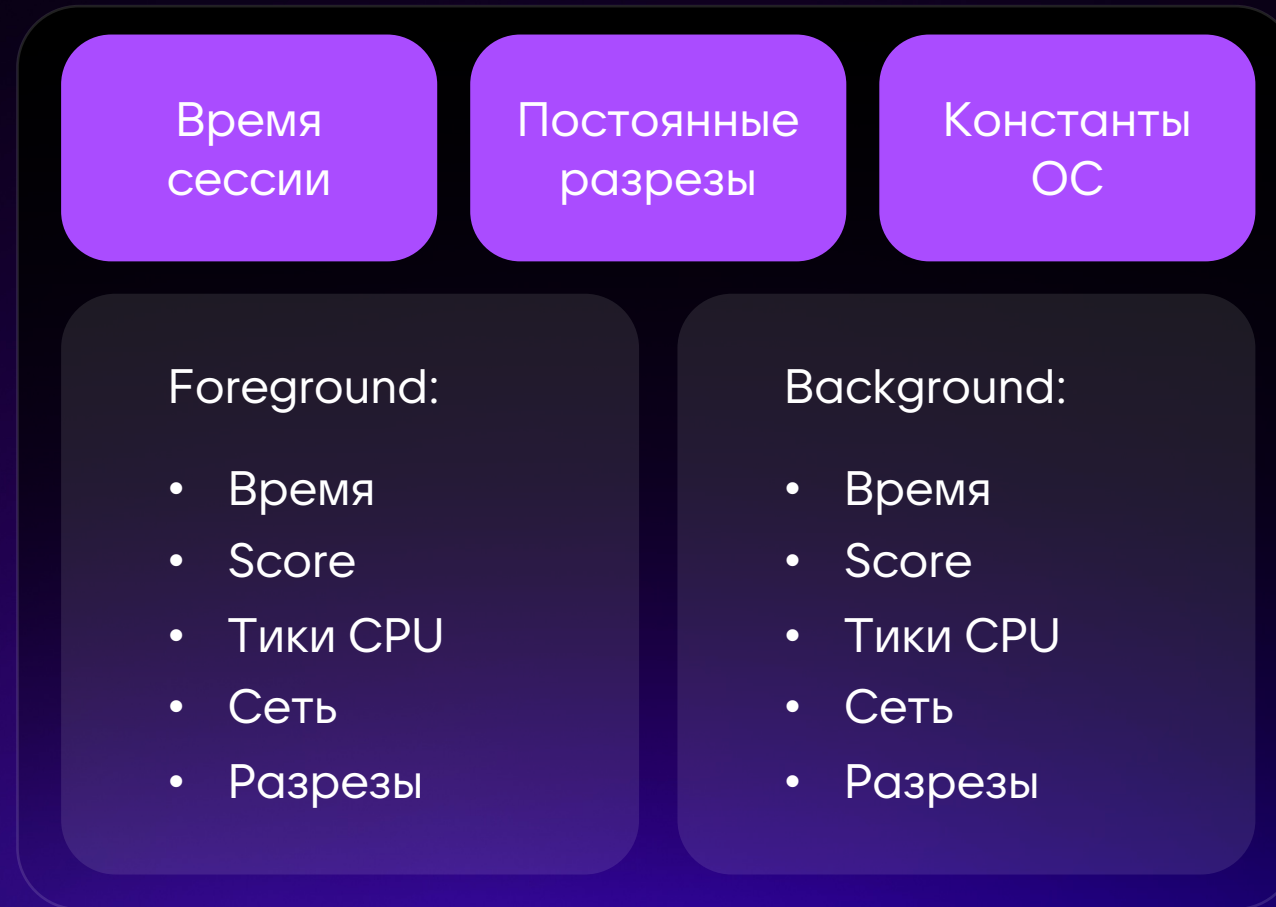
DevicePerformanceClass

BatteryManager.EXTRA_HEALTH

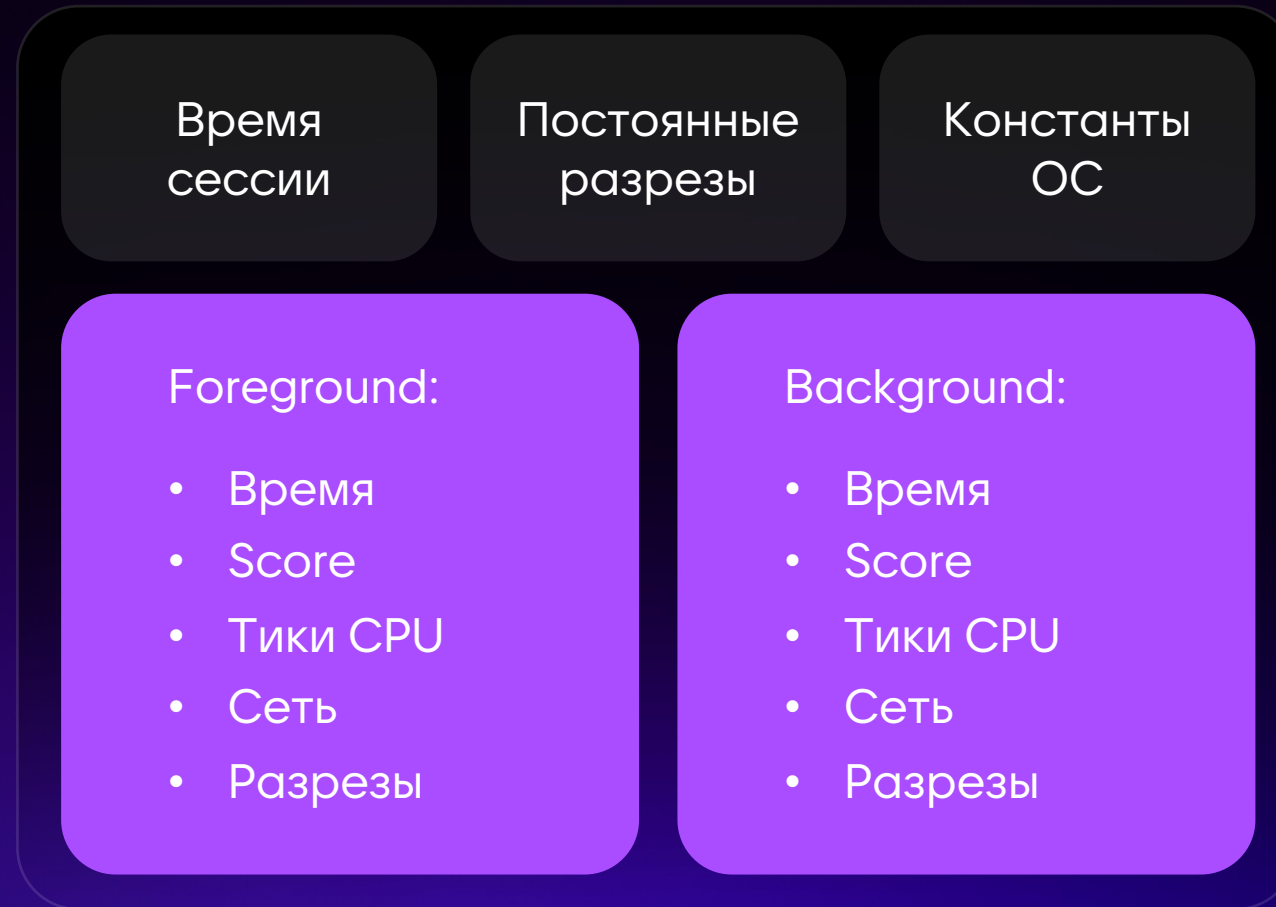
VERSION_CODES.BAKLAVA:
BatteryManager.EXTRA_CYCLE_COUNT

VERSION_CODES.BAKLAVA:
BatteryManager.EXTRA_CAPACITY_LEVEL

Итоговый набор данных



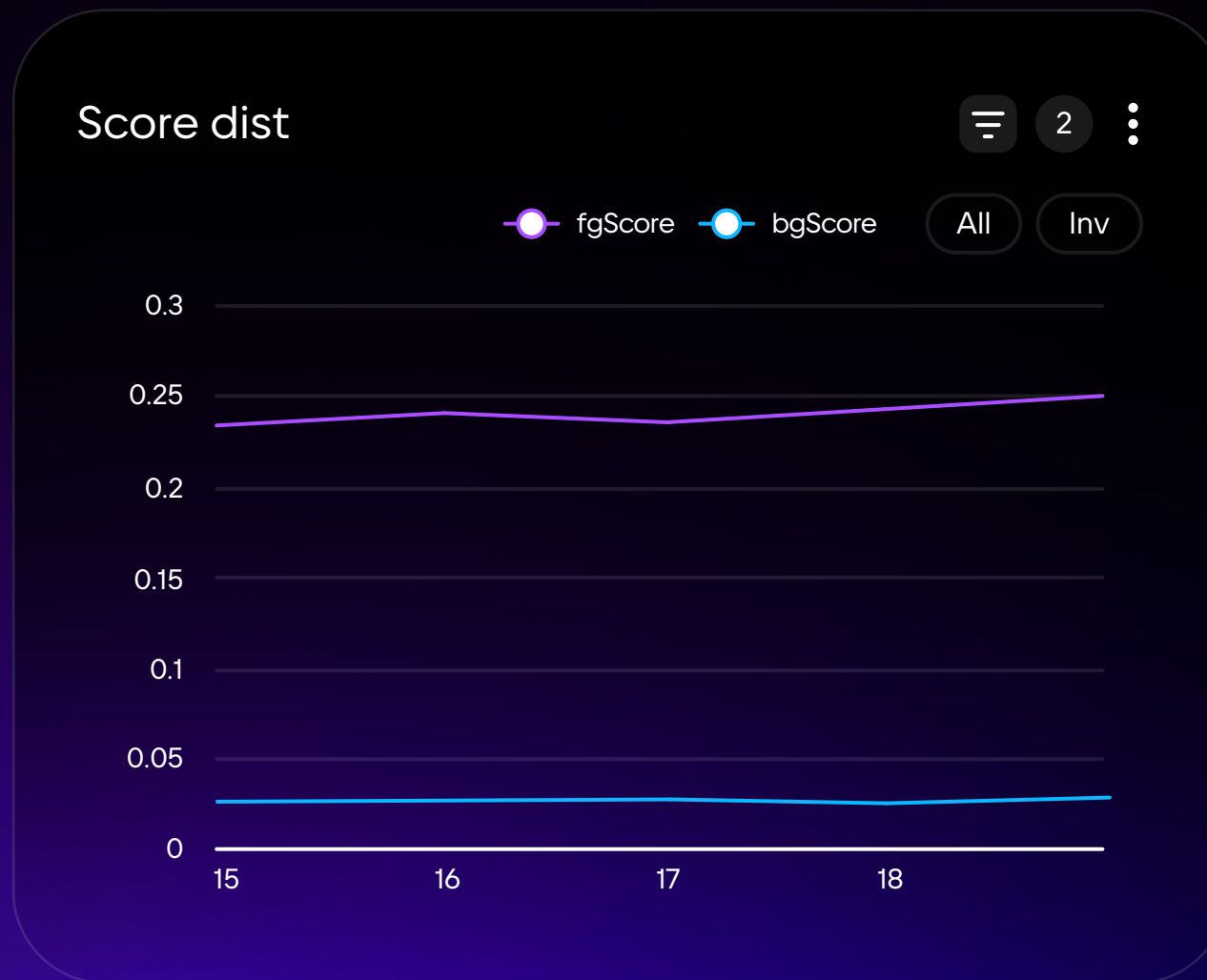
Итоговый набор данных



Дегустируем

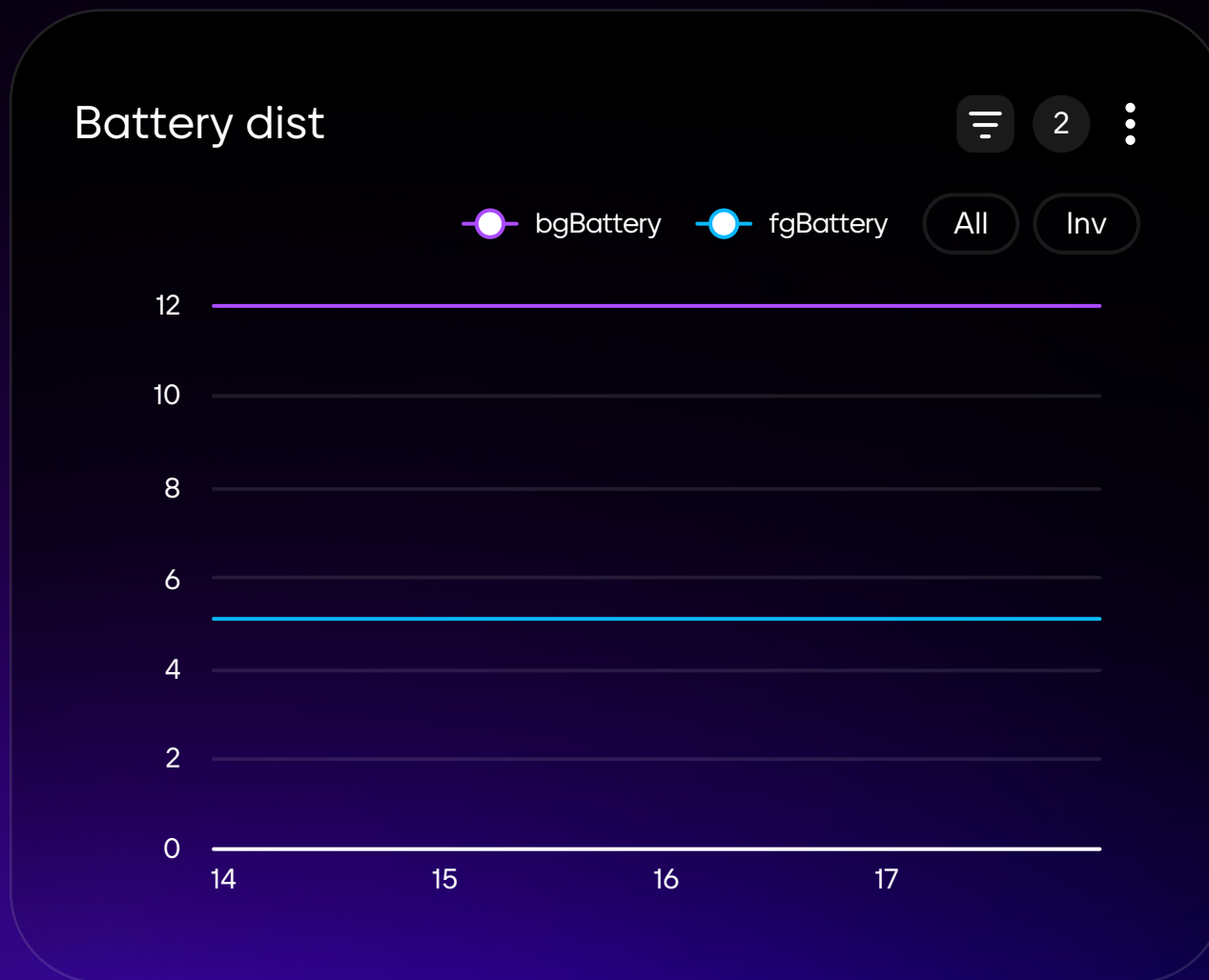
Скоринг в FG & BG

59



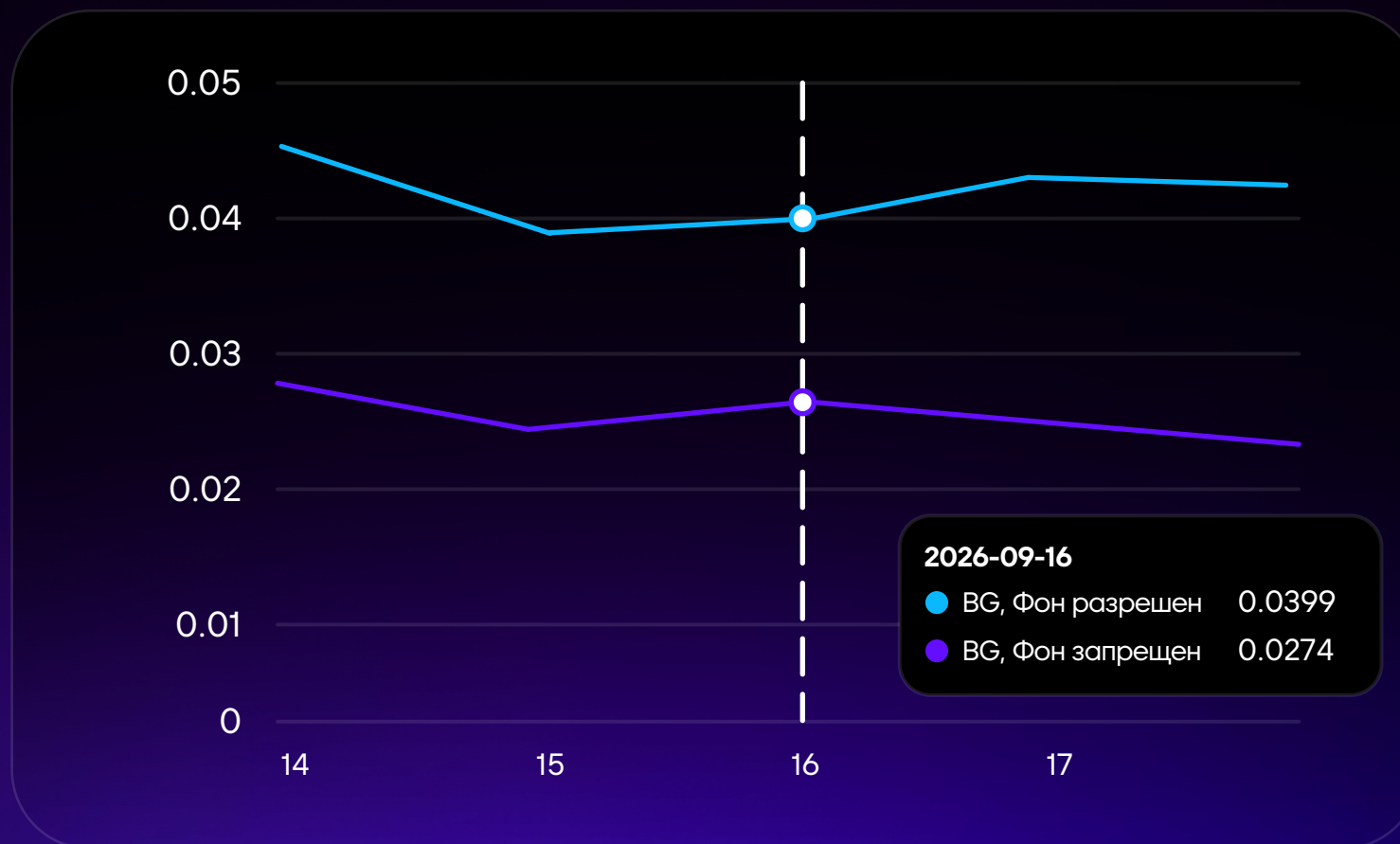
Проценты батареи в FG & BG

60



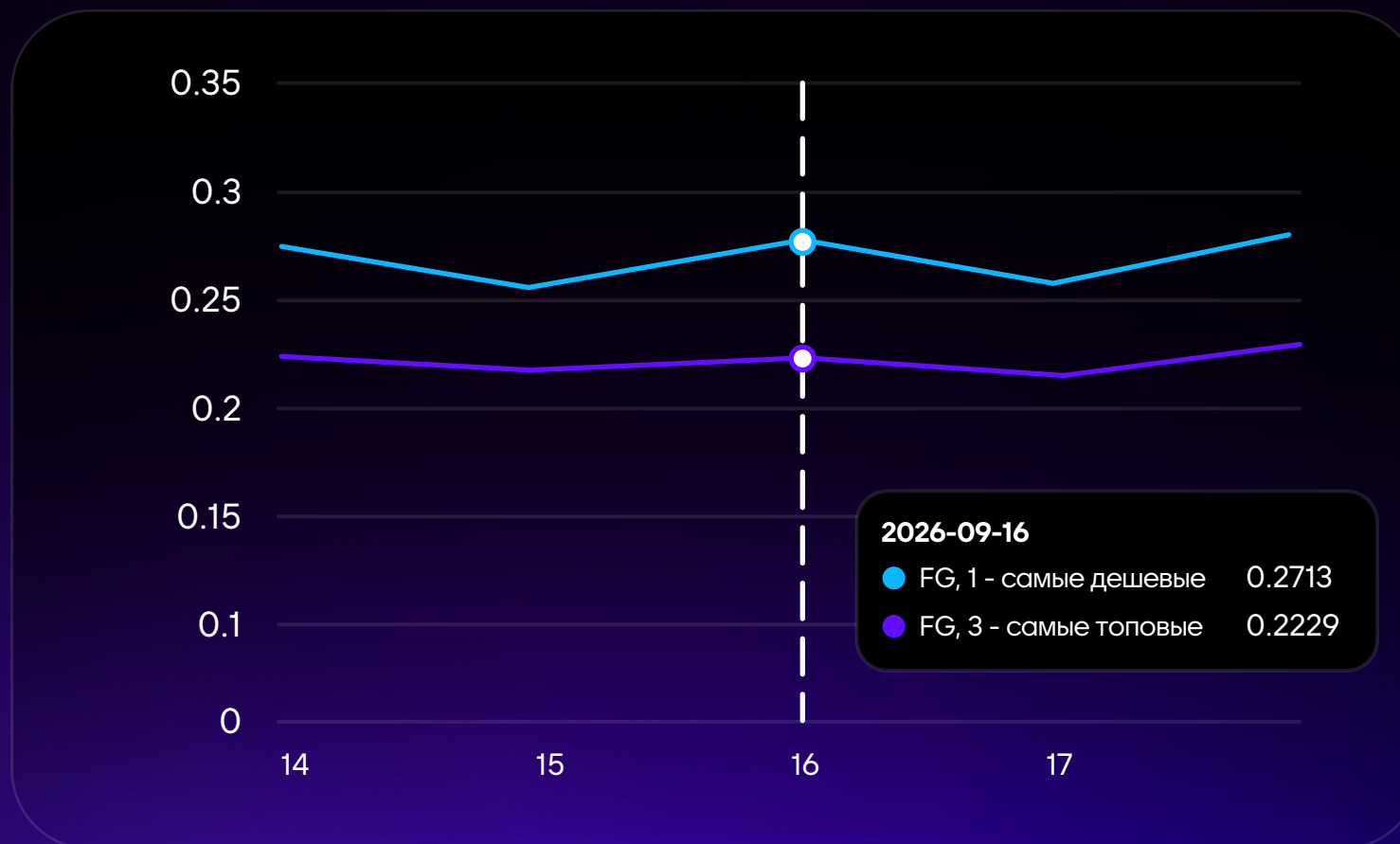
Влияние настройки о фоновой активности

61



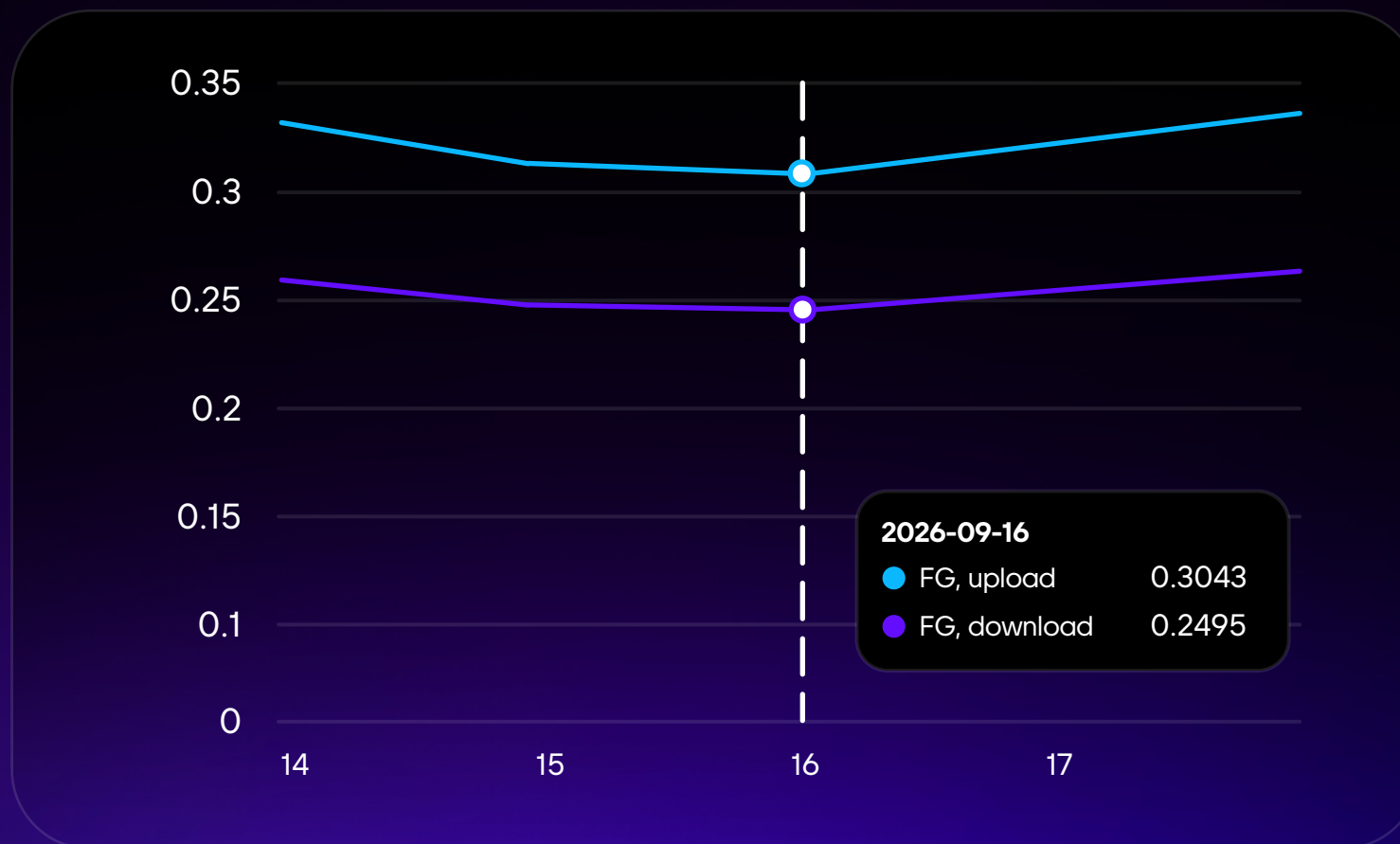
Влияние device performance class

62



Влияние device performance class

63



Сравнение скоринга с эталоном

64



Показательный АБ

65

Мы в релизе
не выключаем
пользователям логи

Логирование
приводит
к увеличению
исполнения кода
на процессоре

Гипотеза —
выключение
логирования
уменьшит
энергопотребление

Сравнение по процентам батареи

	FG p95	BG p95
А (логи есть)	5	13
В (логов нет)	5	13

Сравнение по скорингу

	FG p95	BG p95
А (логи есть)	0,222	0,025
В (логов нет)	0,217	0,023
	-3%	-8%

Сравнение ингредиентов

	FG p95	BG p95
А (логи есть)	91948	19280
В (логов нет)	90670	17908

Получили стабильный
инструмент
для сравнительного
анализа
энергопотребления

Написали множество
переиспользуемых
механизмов, например,
для метрики памяти

Даже при первых
калибровках метрики
находили проблемные
места нашего
приложения



**ГОТОВ ОТВЕТИТЬ
НА ВОПРОСЫ**

