Санкт-Петербургский
Государственный Университет

Математико-Механический факультет
Кафедра Системного Программирования

Кузнецов Илья Александрович

Исследование констант энергопотребления модулей Wi-Fi и Bluetooth и реализация модуля их определения в Navitas Framework

Отчёт по учебной практике

Научный руководитель:
ст. преп. С. Ю. Сартасов

Санкт-Петербург
2021

Содержание

Введение

Наше время невозможно представить без мобильных устройств. Ввиду высокой портативности и универсальности, смартфоны, планшеты и умные часы всё активнее распространяются в мире. В 2020 г. уже насчитывается более 3.5 млрд. пользователей смартфонов, а в 2021 г. ожидается, что эта цифра превысит 3.8 млрд. человек [1]. На сегодняшний день наиболее распространенной операционной системой для мобильных устройств является Android [2].

Около 2 млрд. пользователей смартфонов, что составляет половину от общего числа, ежедневно используют Wi-Fi для выхода в сеть Интернет, скачивания и выгрузки данных [3], а также многие пользователи мобильных устройств активно используют Bluetooth для беспроводного прослушивания аудио, передачи данных между устройствами [4]. В связи с этим вопрос об оценке и оптимизации энергопотребления данных модулей устройств является актуальным, ведь от этого показателя напрямую зависит время их работы. Многие компании, такие как Google, Bluetooth SIG и другие, работают над снижением энергопотребления данных модулей. Так, к 2025 г. ожидается, что 90% всех устройств, поддерживающих Bluetooth, будут использовать технологию Bluetooth Low Energy [4].

Одним из существующих проектов по оценке энергопотребления различных модулей устройств на базе Android является Navitas Framework, разработанный студентами кафедры Системного Программирования СПбГУ [5]. На данный момент Navitas Framework поддерживает оценку энергопотребления CPU и дисплея, проводит анализ собранных данных и визуализирует результаты энергопрофилирования.

Помимо уже имеющейся функциональности, предлагается расширить возможности профилирования путем сбора и анализа информации об энергопотреблении модулей Wi-Fi и Bluetooth, а также их интеграции в Navitas Framework. Также особый интерес вызывает задача определения констант энергопотребления и построения соответствующего энергетического профиля устройства. В данной работе рассматриваются различные методы определения констант энергопотребления, сравнение с уже существующим, а также предлагается реализация метода получения информации об энергопотреблении для профиля питания устройства в рамках проекта Navitas Framework.

1. Цели и задачи

Целью данной работы является исследование различных методов определения констант энергопотребления, разработка собственных подходов, и создание на основе этого модуля их автоматического определения для компонентов Wi-Fi и Bluetooth в энергопрофилировщике-плагине Navitas Framework. Для достижения цели были поставлены следующие задачи.

1. Провести обзор существующих энергопрофилировщиков.
2. Исследовать константы энергопотребления модулей Wi-Fi и Bluetooth в файле профиля питания в различных режимах работы и методы их получения.
3. Разработать свои методы определения констант энергопотребления и сравнить их с уже существующими, а также с Power Profile устройства.
4. Разработать и внедрить систему автоматического определения констант Power Profile для модулей Wi-Fi и Bluetooth в Navitas Framework, а также реализовать генерацию Power Profile с собственными константами.
5. Провести работу над улучшением существующей функциональности Navitas Framework, повышением стабильности и увеличением числа поддерживаемых устройств.

2. Термины и определения

- **Power Profile** (**power_profile.xml**) — XML-файл с константными значениями энергопотребления компонентов конкретного смартфона, предоставляющиеся производителями в определенной директории устройства. В противном случае, содержимое файла инициализируется значениями по умолчанию от Google [6]. В профиле мощность указывается в миллиамперах (mA) потребляемого тока при номинальном напряжении.

- **Android Debug Bridge** (**adb**) — универсальный инструмент командной строки, позволяющий взаимодействовать с мобильным ус-тройством и предоставляющий доступ к оболочке UNIX, которую можно использовать для запуска различных команд на устройстве [7].

- **UID приложения** — уникальный идентификатор, который платформа Android назначает каждому приложению. Android использует UID для создания изолированной программной среды приложения на уровне ядра [8].

- **Права суперпользователя (root-права)** — специальный аккаунт и группа пользователей в UNIX-подобных системах, владелец которого имеет право на выполнение всех без исключения операций.

3. Обзор предметной области

3.1. Обзор энергопрофилировщиков

3.1.1. PowerTutor

PowerTutor — отдельное мобильное приложение, показывающее информацию о текущем потреблении энергии смартфоном [9]. Для работы с модулем Wi-Fi оно использует библиотеку `android.net.wifi.WifiManager` [10]. Однако информации об энергопотреблении модуля Bluetooth данное приложение не получает.

PowerTutor для каждого компонента сопоставляет переменную статуса, которая характеризует возможные состояния модулей устройств. Если сложить их и умножить на определенные коэффициенты, полученные в результате запуска тестовых сценариев на данном устройстве, можно получить приблизительный расход энергии.

Технические ограничения: для устройств под управлением Android 7.0 (Android API Level 24) и выше доступ к файлам, откуда достается информация о потреблении энергии модулем Wi-Fi [11], невозможен без получения прав суперпользователя, что значительно усложняет такой способ получения значений энергопотребления и делает его пригодным для применения только на заранее подготовленных устройствах.

HTC Dream Power Model

Model	$(\beta_{uh} \times freq_h + \beta_{ul} \times freq_l) \times util + \beta_{CPU} \times CPU\_on + \beta_{br} \times brightness + \beta_{Gon} \times GPS\_on + \beta_{Gsl} \times GPS\_sl + \beta_{Wi\text{-}Fi\_l} \times Wi\text{-}Fi_l + \beta_{Wi\text{-}Fi\_h} \times Wi\text{-}Fi_h + \beta_{3G\_idle} \times 3G_{idle} + \beta_{3G\_FACH} \times 3G_{FACH} + \beta_{3G\_DCH} \times 3G_{DCH}$						
Category	System variable	Range	Power coefficient	Category	System variable	Range	Power coefficient
CPU	util	1–100	β_{uh}: 4.34 β_{ul}: 3.42	LCD	brightness	0–255	β_{br}: 2.40
	$freq_l$,$freq_h$	0,1	n.a.	GPS	GPS_on	0,1	β_{Gon}: 429.55
	CPU_on	0,1	β_{CPU}: 121.46		GPS_sl	0,1	β_{Gsl}: 173.55
Wi-Fi	npackets, R_{data}	0–∞	n.a.	Cellular	data_rate	0–∞	n.a.
	$R_{channel}$	1–54	β_{cr}		downlink_queue	0–∞	n.a.
	$Wi\text{-}Fi_l$	0,1	$\beta_{Wi\text{-}Fi\_l}$: 20		uplink_queue	0–∞	n.a.
	$Wi\text{-}Fi_h$	0,1	$\beta_{Wi\text{-}Fi\_h}$: Equation 1		$3G_{idle}$	0,1	β_{3G_idle}: 10
Audio	Audio_on	0,1	β_{audio}: 384.62		$3G_{FACH}$	0,1	β_{3G_FACH}: 401
					$3G_{DCH}$	0,1	β_{3G_DCH}: 570

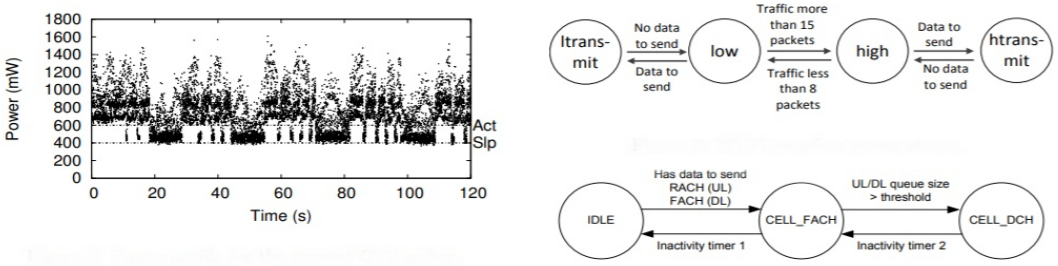

Рис. 1: Модель энергопотребления устройства в PowerTutor [12]

3.1.2. Energy Profiler

Energy Profiler — встроенный в Android Studio плагин-профилировщик.

Energy Profiler контролирует использование CPU, мобильной сети, Wi-Fi, Bluetooth, датчика GPS и отображает визуализацию того, сколько энергии потребляет каждый из этих компонентов [13]. Energy Profiler также показывает наличие системных событий (wakelocks, alarms, jobs и location requests), которые могут повлиять на потребление энергии.

Energy Profiler не измеряет потребление энергии напрямую, а использует модель, которая его оценивает для каждого ресурса на устройстве. Информация о потреблении тока в реальном времени сопоставляется с программным счётчиком.

Технические ограничения: Android 8.0 (Android API Level 26) и выше.

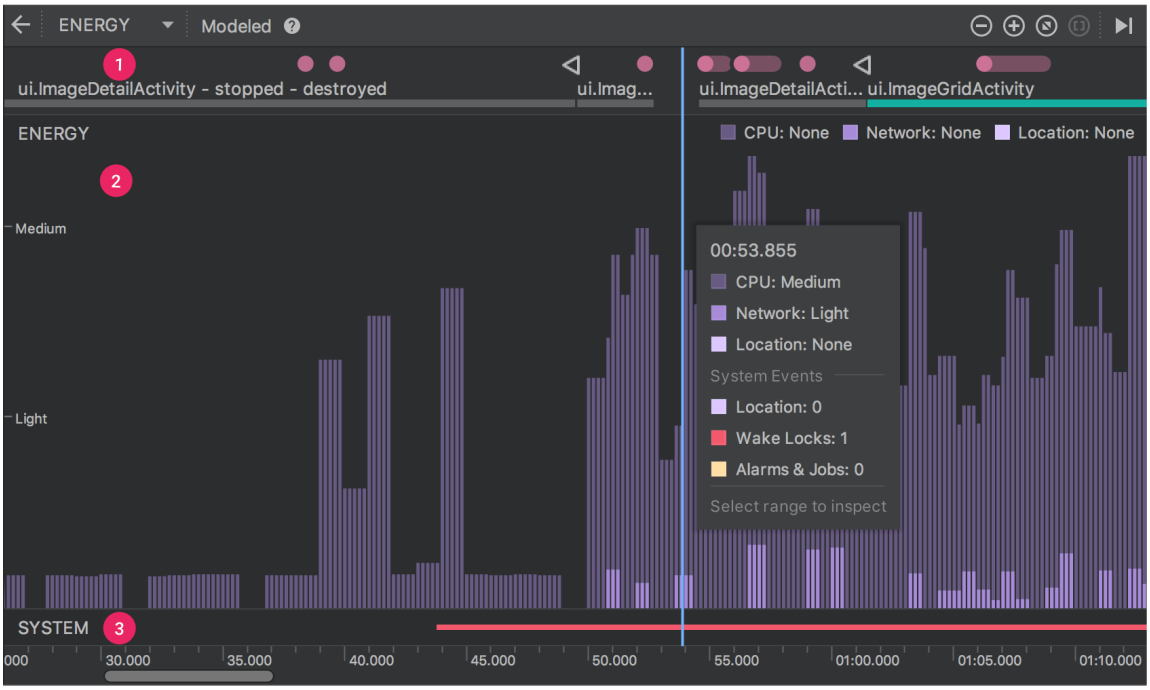

Рис. 2: Визуализация энергопотребления в Energy Profiler

3.1.3. Battery Historian

Battery Historian — инструмент компании Google, разработанный для получения информации о разряде батареи устройства с течением времени [14].

На общесистемном уровне инструмент визуализирует связанные с питанием события из системных журналов в формате HTML. На уровне конкретного приложения инструмент предоставляет различные данные, которые могут помочь определить поведение приложения, расходующего заряд батареи.

Стоит отметить, что профилировщик собирает данные, непосредственно используя команду `adb shell dumpsys batterystats`, при этом используя для запуска Docker и .zip-отчёт, который получается командой `adb bugreport [path/] bugreport.zip`.

Диаграмма, отображающая выводимую Battery Historian информацию, разделена по категориям, в которых отображается информация об активности соответствующего модуля в течение определенного времени, как показано на абсциссе диаграммы. На ординате диаграммы каждая строка показывает сегмент цветной полосы во времени, когда компонент системы активен и, таким образом, потреблял ток от батареи [15].

Однако диаграмма не показывает, сколько энергии было использовано компонентом, а только то, в течение какого времени приложение было активным.

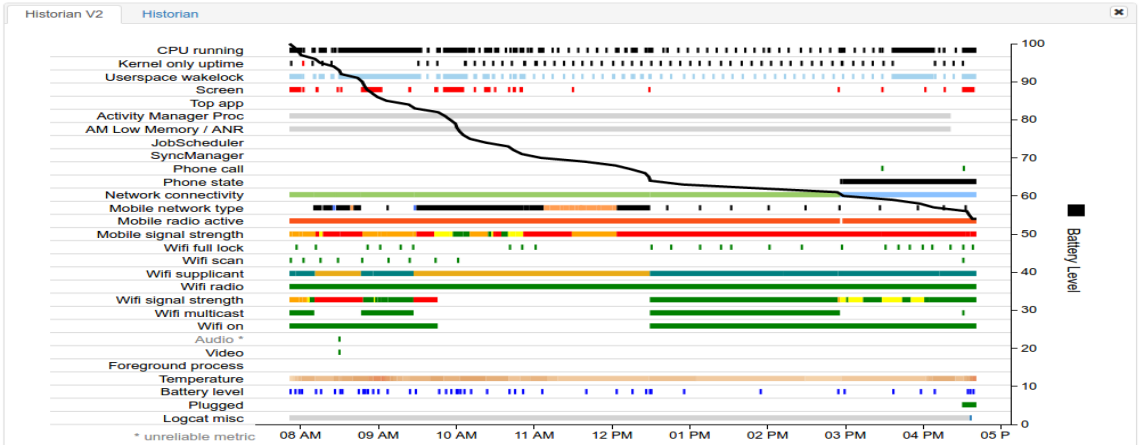

Рис. 3: Отображение в Battery Historian общесистемных событий, влияющих на энергопотребление

Можно также просмотреть дополнительную информацию из соответствующего файла `batterystats.txt` в разделе статистики **System Stats** под диаграммой, показанной в Battery Historian. Именно из этого файла, который был

получен командой `adb shell dumpsys batterystats`, Battery Historian берёт и анализирует всю информацию.

Стоит заметить, что данные в нём отсчитываются с последнего сброса статистики энергопотребления в данном файле.

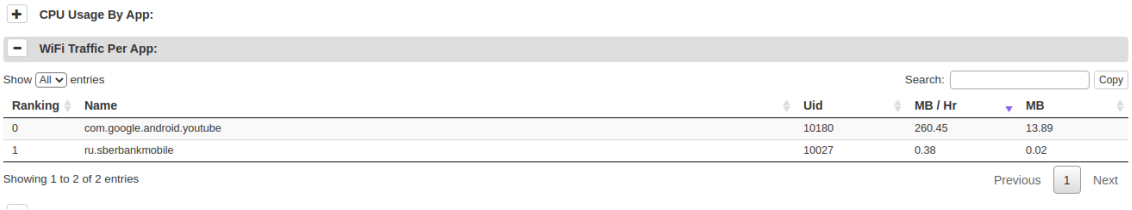

Рис. 4: Информация об использовании трафика приложениями

Таким образом, здесь выводится более подробная информация о модуле Wi-Fi, в которой имеется:

- UID приложения, использующего или когда-либо использовавшего данный модуль;
- скорость потребления в мегабайт/час для каждого приложения;
- количество мегабайт, потребленных тем или иным приложением.

Однако информации, напрямую отражающей потребление энергии каким-либо модулем или приложением, взаимодействующим с этим модулем, здесь нет.

Необходимо отметить, что, несмотря на наличие информации в Battery Historian о времени работы модуля Bluetooth в различных состояниях, какой-либо информации о данном модуле в разделе **System Stats** нет.

Технические ограничения: Android 5.0 (Android API Level 21) и выше.

3.2. Профили питания Android

В соответствии с документацией платформы Android [16], информация об использовании батареи получается из статистики использования батареи и значений профиля мощности [17].

Платформа Android автоматически определяет статистику использования батареи, отслеживая, как долго компоненты устройства находятся в разных состояниях. Когда компоненты (набор микросхем Wi-Fi, мобильная связь, Bluetooth, GPS, дисплей, CPU) меняют состояния (вкл. / выкл., режим ожидания / полная мощность, низкая / высокая яркость и другие), управляющая служба сообщает об этом службе фреймворка BatteryStats. BatteryStats собирает информацию с течением времени и сохраняет её для использования при перезагрузках. Служба не отслеживает потребление тока батареей напрямую, а вместо этого собирает информацию о времени, которую можно использовать для приблизительного расчета потребления батареи различными компонентами.

Во избежание потери статистики использования в случае выключения устройства (то есть батарея достигла нулевой оставшейся емкости), фреймворк отображает статистику примерно каждые 30 минут.

В соответствующем Power Profile файле устройства находятся следующие параметры Wi-Fi:

- `wifi.on` — данное значение показывает мощность, потребляемую, когда модуль Wi-Fi включен, но не осуществляет передачу или прием данных. По рекомендациям Google, данное значение можно измерить разницей между текущим потреблением энергии в состоянии приостановки (сна) системы с включенным и отключенным Wi-Fi;
- `wifi.scan` — данное значение показывает мощность, потребляемую во время сканирования точек доступа при помощи Wi-Fi;
- `wifi.active` — данное значение показывает мощность, используемую при передаче или приеме данных по Wi-Fi.

Также в этом файле есть информация о параметрах модуля Bluetooth:

- `bluetooth.on` — данное значение показывает мощность, когда модуль Bluetooth включен и находится в режиме поиска доступных устройств, но при этом не подключен к какому-либо устройству;
- `bluetooth.scan` — данное значение показывает мощность, потребляемую во время сканирования доступных для подключения устройств при помощи Bluetooth;

- `bluetooth.active` — данное значение показывает мощность, когда происходит передача данных по Bluetooth.

4. Исследование констант энергопотребления

Известно, что многие производители игнорируют требование Google о предоставлении актуального для конкретного устройства Power Profile. Как было сказано в разделе 2., данные Power Profile в таком случае инициализируются константами в mA по умолчанию от Google, которые не имеют никакого отношения к реальным данным устройств.

В таком случае ничего не остаётся, кроме проведения собственного экспериментального уточнения значений констант, на основе которого можно реализовать систему автоматического уточнения этих данных для модулей Wi-Fi и Bluetooth для внедрения в Navitas Framework через запуск тестового приложения. Данный раздел посвящён методологиям получения уточнённых констант для модулей Wi-Fi и Bluetooth в Power Profile для конкретного устройства.

4.1. Получение констант энергопотребления для Power Profile методом А. Саксонова

Получение информация об энергопотреблении модулей Wi-Fi и Bluetooth происходило при помощи инструмента **adb**, подробнее о котором написано в работе В. И. Мирошникова «Исследование энергопотребления модулей Wi-Fi и Bluetooth и их интеграция в Navitas Framework». Данные для анализа получены тестированием при помощи написанного bash-скрипта [18] в следующем тестовом окружении:

1. Перевод всех приложений, кроме непосредственно задействованных в эксперименте, в режим (глубокого) сна, запрещающего любую фоновую деятельность и отправку уведомлений. Ввиду отсутствия команд, позволяющих осуществить это без прав суперпользователя, подготовка выполнялась вручную.

2. Включение авиарежима, который отключает мобильную связь. Данное действие производится вручную, так как команду, не требующую прав суперпользователя для текущего уровня API, найти не удалось; а также до начала начала тестирования, так как его включение приводит к отключению тестируемых модулей Wi-Fi и Bluetooth.

3. Отключение зарядки устройства с помощью команды `adb shell dumpsys battery unplug` либо команд `adb shell dumpsys battery set ac 0`, `adb`

`shell dumpsys battery set usb 0`, `adb shell dumpsys battery set wireless 0`.

4. Включение только одного модуля, тестируемого в текущий момент, для снижения погрешности в замерах энергопотребления, связанной с влиянием компонентов друг на друга. Данное действие совершается при помощи команды `adb shell svc [module] enable`.

5. Отключение экрана устройства с помощью команды `adb shell input keyevent 26`.

На устройстве **Samsung Galaxy S9+ (Android API Level 28)**, было проведено 5 тестовых запусков, каждый из которых продолжался 30 секунд. Все эксперименты проходили с частотой дискретизации, равной 2 Гц. Таким образом, было проведено по 300 замеров для каждого модуля в двух состояниях.

Для определения соответствующей константы энергопотребления был выбран метод, использующийся в работе Андрея Саксонова «Method to Derive Energy Profiles for Android Platform» [19].

1. По полученным данным вычисляется математическое ожидание **a** и стандартное отклонение **s**.

2. Все значения, не входящие в отрезок **[a - s; a + s]** более не рассматриваются.

3. Вычисляется математическое ожидание точек **c**, находящихся в части отрезка **[a - s, a]**. Полученное значение — константа энергопотребления в **mAh**, далее это число переводится в mA, чтобы соответствовать стандартным для Power Profile единицам измерения.

1. **Константа энергопотребления модуля Wi-Fi в режиме** `wifi.on`:

   По полученным данным был построен следующий график:

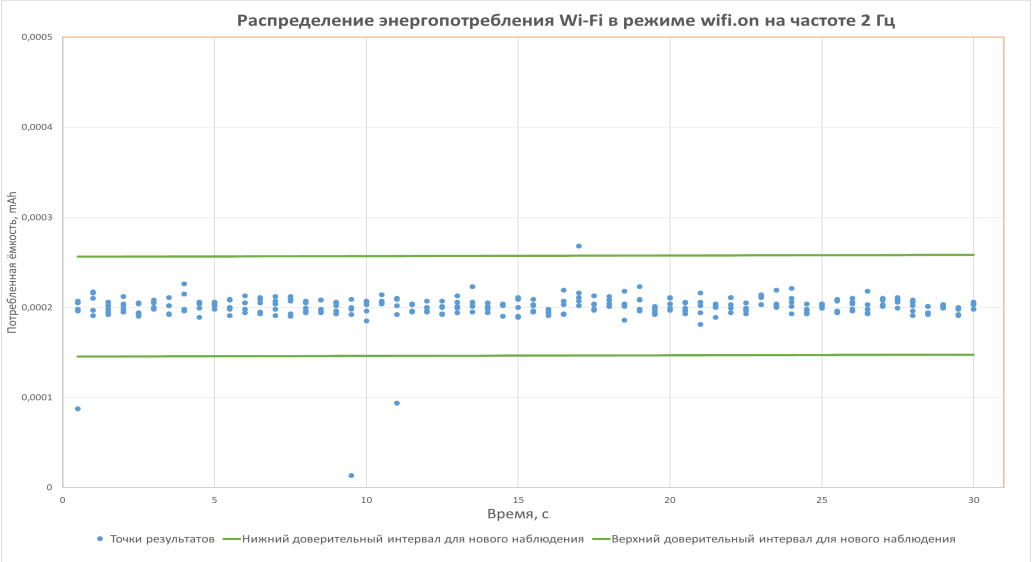

Рис. 5: Распределение энергопотребления Wi-Fi в режиме wifi.on на частоте 2 Гц

По построенному графику с помощью метода описанного выше, была получена константа энергопотребления, равная **1.41 mA**.

2. **Константа энергопотребления модуля Wi-Fi в режиме** `wifi.scan`:

   Для данного тестового сценария была найдена команда, запускающая сканирование сетей Wi-Fi, `adb shell su 0 service call wifi 11`, однако для запуска она требует прав суперпользователя, которых нет на тестируемом устройстве. Поэтому тестирование данного режима не проводилось.

3. **Константа энергопотребления модуля Wi-Fi в режиме** `wifi.active`:

   По полученным данным был построен следующий график:

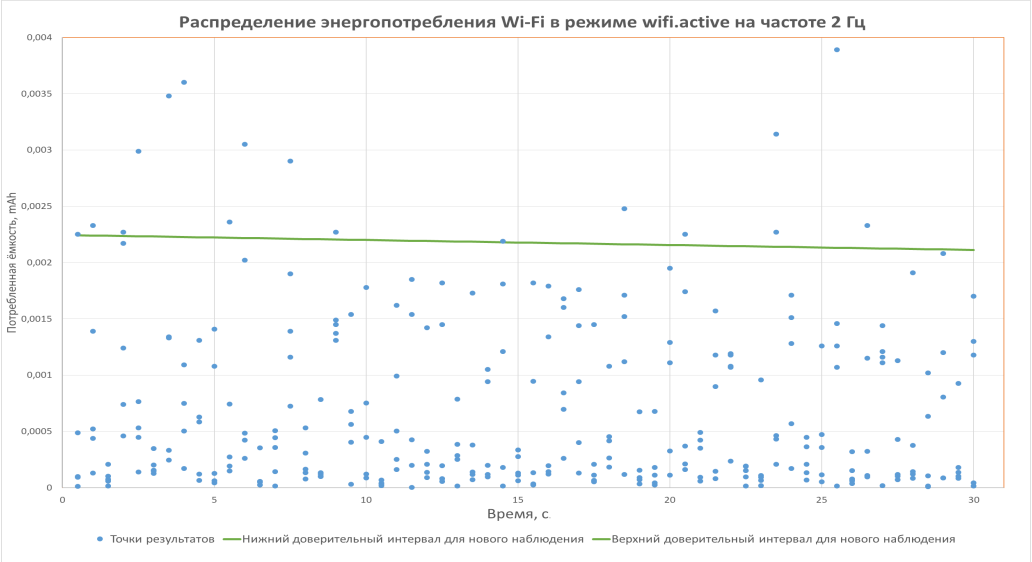

Рис. 6: Распределение энергопотребления Wi-Fi в режиме wifi.active на частоте 2 Гц

По построенному графику с помощью метода, описанного выше, была получена константа энергопотребления, равная **4.67 mA**.

4. **Константа энергопотребления модуля Bluetooth в режиме** `bluetooth.on`:

   По полученным данным был построен следующий график:

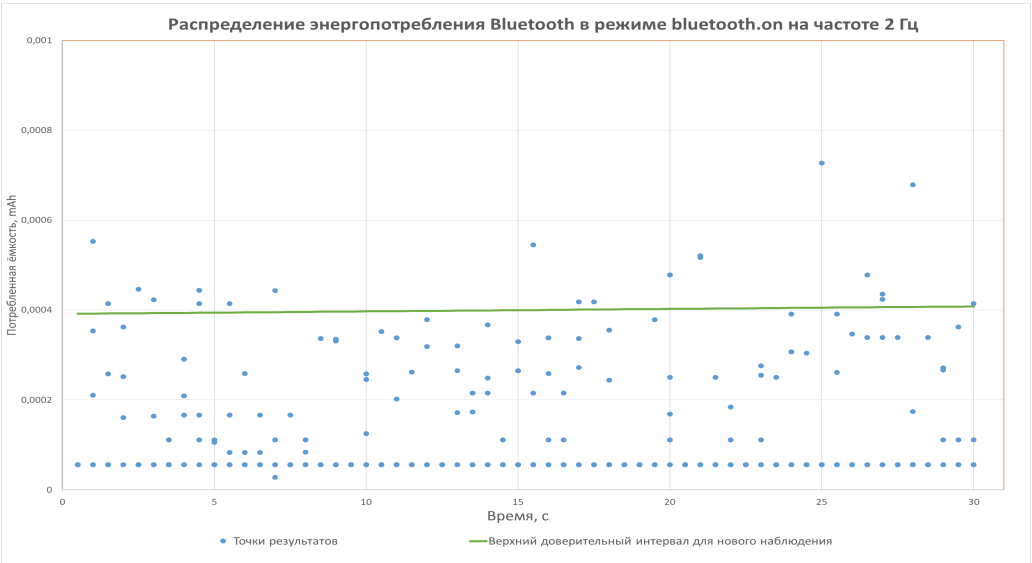

Рис. 7: Распределение энергопотребления Bluetooth в режиме bluetooth.on на частоте 2 Гц

По построенному графику с помощью алгоритма, описанного выше, была получена константа энергопотребления, равная **0.44 mA**.

5. **Константа энергопотребления модуля Bluetooth в режиме** `bluetooth.scan`:

   Для данного тестового сценария не была найдена команда, запускающая сканирование сети Bluetooth, не требующая для запуска прав суперпользователя, которые отсутствуют на тестируемом устройстве. Таким образом, тестирование данного режима не проводилось.

6. **Константа энергопотребления модуля Bluetooth в режиме** `bluetooth.active`:

   По полученным данным был построен следующий график:

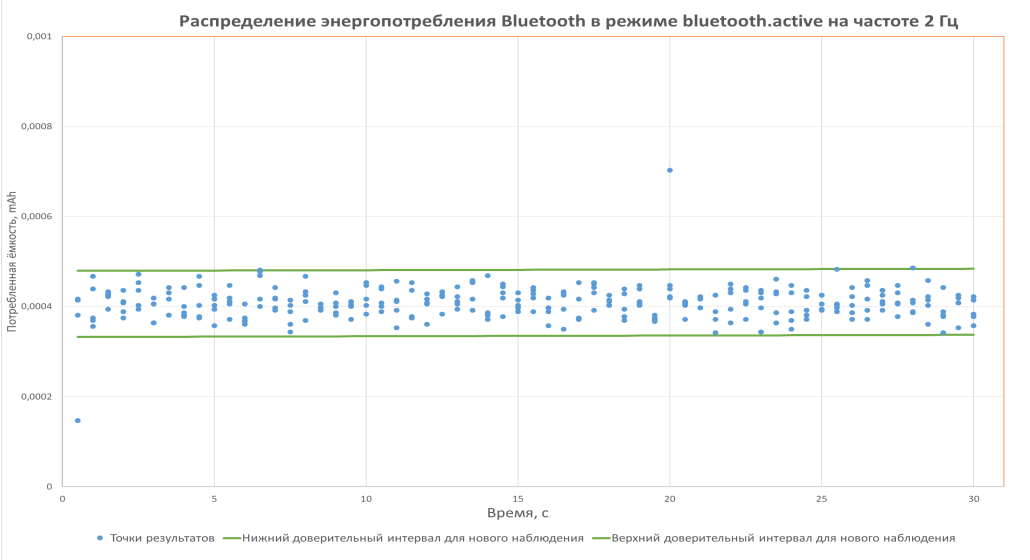

Рис. 8: Распределение энергопотребления Bluetooth в режиме bluetooth.active на частоте 2 Гц

По построенному графику с помощью алгоритма, описанного выше, была получена константа энергопотребления, равная **2.82 mA**.

Представим итоги данного эксперимента в виде таблицы.

Таблица 1: Таблица констант энергопотребления

Режим работы	**Константа энергопотребления (mA)**
wifi.on	1.41
wifi.active	4.67
bluetooth.on	0.44
bluetooth.active	2.82

Таким образом, были получены собственные значения констант Power Profile для модулей Wi-Fi и Bluetooth с использованием приведённой методологии. К сожалению, сравнить их с оригинальным файлом Power Profile устройства **Samsung Galaxy S9+ (Android API Level 28)** не представляется возможным, так как производитель, как и многие другие, не предоставил актуальных значений в этом файле. Соответственно, там указаны значения по умолчанию, не представляющие интереса для исследования.

4.2. Исследование зависимости данных об энергопотреблении устройства от частоты дискретизации эксперимента

При изучении инструмента **adb** в работе В. И. Мирошникова «Исследование энергопотребления модулей Wi-Fi и Bluetooth и их интеграция в Navitas Framework» была выдвинута гипотеза, что получение информации об энергопотреблении средствами **adb** также затрачивает некоторую энергию, а именно: с увеличением частоты дискретизации измеренная в ходе эксперимента энергия также возрастает. В данном разделе работы будет проведено изучение связи между частотой дискретизации и изменением энергопотребления устройства.

Эксперимент проводился на двух тестовых устройствах: **Samsung Galaxy S21 Ultra (Android API Level 30)** и **Sony ZL (Android API Level 22)**. Для проведения данного исследования был также написан соответствующий bash-скрипт [18] и обеспечена такая же изоляция, что и в экспериментах раздела 4.1.

Для проведения исследования была выбрана следующая методология.

1. Для каждого из тестовых режимов `wifi.on`, `wifi.active`, `bluetooth.on`, `bluetooth.active` было проведено тестирование на частотах, приведённых в таблице ниже. Эксперименты в режимах `wifi.scan` и `bluetooth.scan` не проводились ввиду отсутствия подходящих команд для указанных уровней

API Android. При этом, для улучшения точности эксперимента для каждой частоты было выполнено по 5 тестовых запусков.

Таблица 2: Частоты дискретизации и количество замеров за 30 секунд для одного тестового запуска

Частота дискретизации (Гц)	**Количество замеров (шт.)**
10	300
5	150
3.33	100
2.5	75
2	60
1.67	50
1.43	42
1.25	37
1.11	33
1	30
0.91	27
0.83	25
0.77	23
0.71	21
0.67	20
0.625	18
0.59	17
0.56	16
0.52	15
0.5	15

2. Затем для всех 5-ти тестовых запусков на каждой частоте дискретизации по-отдельности вычислялся коэффициент наклона соответствующей регрессионной прямой. Таким образом, результатом данного этапа для каждого тестового сценария стала таблица 5x20, где столбцы представляют собой частоты дискретизаций в Гц, а строки — коэффициенты наклона регрессионных прямых.

3. Далее для каждой частоты по 5-ти коэффициентам наклона подсчитывалось математическое ожидание и медиана эксперимента. Выбор данной мо-

дели вычисления обусловлен следующим: 5 полученных значений для каждой частоты позволяют усреднить её коэффициент наклона, ведь за время тестирования, несмотря на обеспеченность надёжной изоляцией, всё ещё могут возникать внешние воздействия, влияющие на энергопотребление. Вычисление и математического ожидания, и медианы для тестирования позволяют оценить разницу итоговых коэффициентов наклона регрессионных прямых.

4. Для полученных данных были построены соответствующие графики с применением линейного и экспоненциального регрессионного анализа, а также произведена их экстраполяция в точке пересечения с осью ординат.

Рассмотрим результаты проведённых экспериментов.
Тестирование модуля Wi-Fi в режиме `wifi.on`:
Samsung Galaxy S21 Ultra (Android API Level 30)

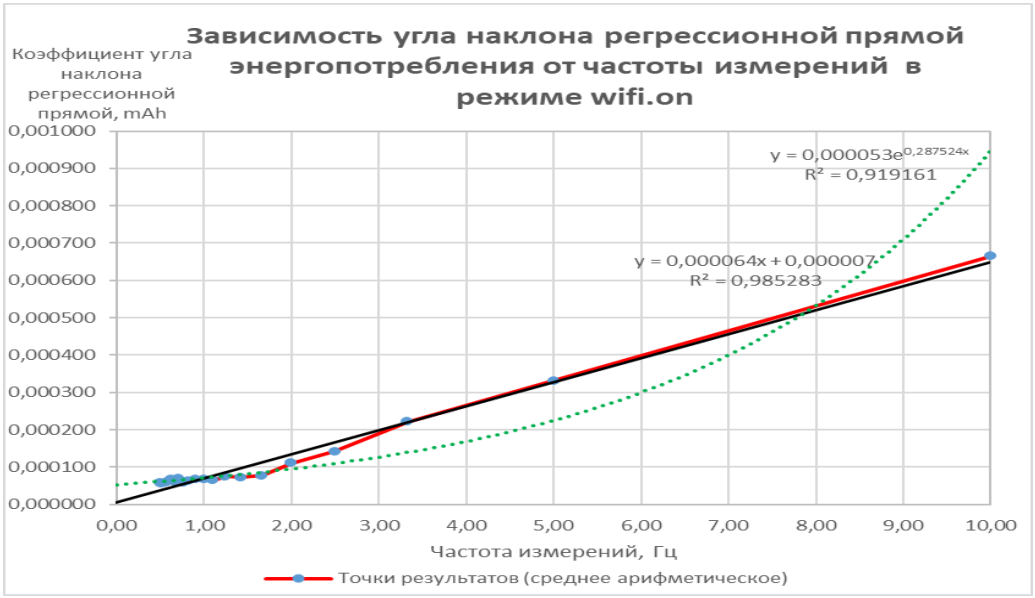

Рис. 9: Зависимость угла наклона регрессионной прямой энергопотребления от частоты измерений в режиме `wifi.on`

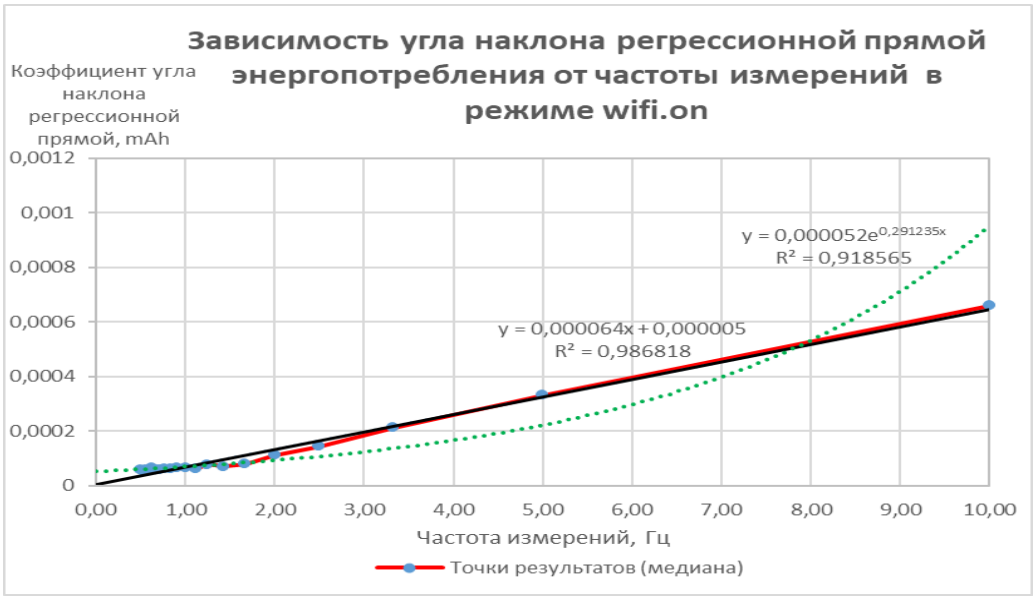

Рис. 10: Зависимость угла наклона регрессионной прямой энергопотребления от частоты измерений в режиме `wifi.on`

Sony ZL (Android API Level 22)

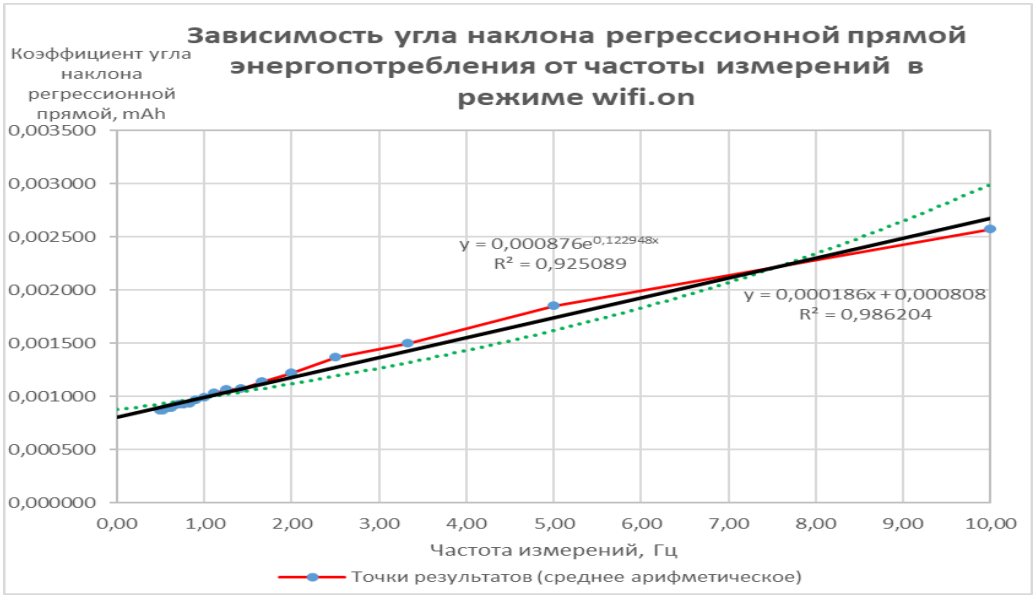

Рис. 11: Зависимость угла наклона регрессионной прямой энергопотребления от частоты измерений в режиме `wifi.on`

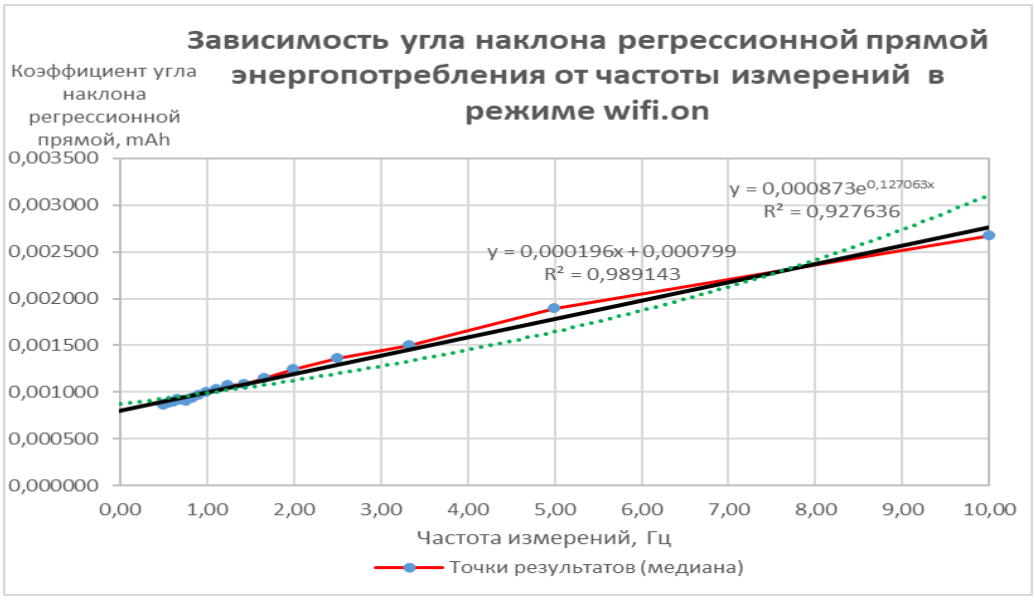

Рис. 12: Зависимость угла наклона регрессионной прямой энергопотребления от частоты измерений в режиме `wifi.on`

Примечательно, что, начиная с частоты 2 Гц, на обоих устройствах линейная регрессия дает наиболее близкое приближение к точкам результатов, нежели экспоненциальная, которая, в свою очередь, дает лучшее приближение на малых частотах. Это же подтверждается коэффициентами детерминации, указанными на графиках. Этот факт нужно также учитывать при выборе вида регрессии, поэтому для в качестве основной была выбрана линейная регрессия, так как она обладает максимальным правдоподобием.

По построенным графикам видно, что **с увеличением частоты дискретизации растет и коэффициент наклона соответствующей регрессионной прямой**. Из наблюдаемого роста коэффициента наклона регрессионной прямой с увеличением частоты дискретизации следует, что и получаемые значения энергопотребления при увеличении частоты — растут. Это связано с тем, что коэффициент наклона регрессионной прямой при определенной частоте дискретизации и отражает величину энергопотребления на этой частоте.

В других тестовых сценариях, таких как `wifi.active`, `bluetooth.on` и `bluetooth.active`, выявлена схожая тенденция: при увеличении частоты дискретизации растет и коэффициент угла наклона регрессионной прямой, и, соответственно, само энергопотребление. Для краткости изложения графики этих тестовых сценариев приведены не будут.

Установленный факт подтверждает гипотезу, сформулированную в начале

раздела 4.2., и дает понять, что использование метода **adb** в качестве источника получения информации об энергопотреблении тоже имеет свой недостаток: **с увеличением частоты дискретизации растет и общее энергопотребление**. Это должно быть учтёно при разработке методов определения констант.

4.3. Собственные методы определения констант энергопотребления

Во время работы над исследовательской частью учебной практики было принято решение разработать свои методы определения констант энергопотребления.

Выделим два метода определения констант энергопотребления, которые в дальнейшем будут именоваться **«средние по средним»** и **«медиана по медиане»**.

Рассмотрим общую методологию получения констант энергопотребления с помощью данных алгоритмов, схожую с той, что была рассмотрена в разделе 4.2.

1. Для каждого тестового режима из `wifi.on`, `wifi.active`, `bluetooth.on` и `bluetooth.active` было проведено тестирование на тех же частотах дискретизации, что и в разделе 4.2., при этом для лучшей точности эксперимента для каждой частоты было сделано по 5 тестовых запусков.

2. Затем для каждой частоты и всех 5-ти тестовых запусков в каждый момент времени значение энергопотребления в **mAh** переводилось в силу тока в **mA**, чтобы соответствовать стандартным единицам измерения для Power Profile устройства. По полученному столбцу сил тока вычислялось математическое ожидание и медиана. Полученные величины являются усредненными по определенному правилу значениями силы тока для определённой частоты. Таким образом, результатом данного этапа для каждого эксперимента стала таблица 5x20, где строки представляют собой значения силы тока в **mA**, а столбцы - частоты дискретизаций в Гц.

3. Далее, по аналогии с этапом эксперимента, описанным в разделе 4.2., для каждой частоты по 5-ти значениям силы тока высчитывалось математическое ожидание и медиана данного эксперимента.

4. Для полученных данных были построены соответствующие графики с применением линейного и экспоненциального регрессионного анализа, а также произведена их экстраполяция в точке пересечения с осью ординат: она и

является константой энергопотребления, ведь фактически отражает значение силы тока при частоте дискретизации, равной 0 Гц, — так, если бы это происходило в отсутствие влияния **adb** на эксперимент, описанном в разделе 4.2.

Перейдем к результатам экспериментов, которые проходили на устройстве **Sony ZL (Android API Level 22)**, поскольку он обладает проинициализированным производителем Power Profile.

Тестирование модуля Wi-Fi в режиме `wifi.on`:

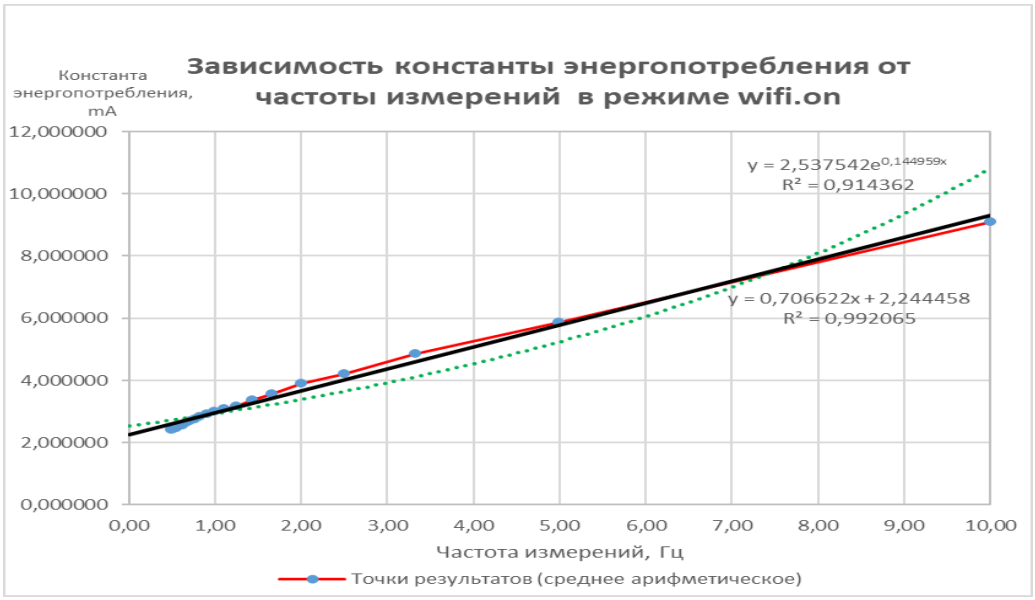

Рис. 13: Зависимость константы энергопотребления от частоты измерений в режиме `wifi.on`

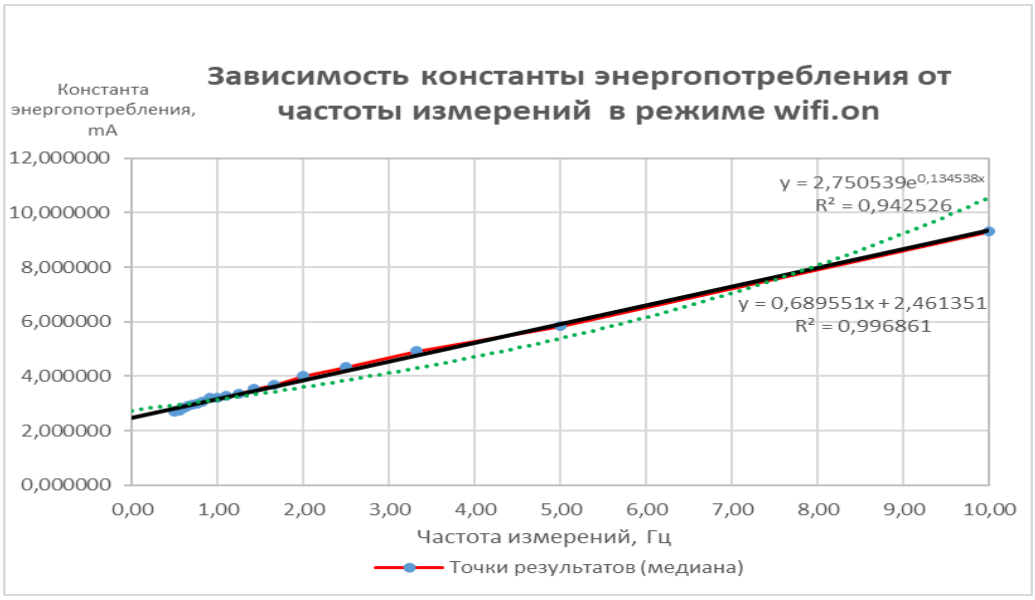

Рис. 14: Зависимость константы энергопотребления от частоты измерений в режиме `wifi.on`

На полученных графиках определены соответствующие константы энергопотребления, причем видно, что значения при подходах вычисления через математическое ожидание и медиану отличаются несущественно. Также по значению коэффициента детерминации видно, что линейная регрессия дает лучшее приближение, нежели экспоненциальная.

Для других тестовых сценариев аналогичным образом были определены константы энергопотребления. Исключение составляет тестирование модуля Bluetooth в режиме `bluetooth.on`, так как на тестируемом устройстве **adb** всегда выдавал нулевое значение энергопотребления, так называемый «чистый ноль».

Для подтверждения достоверности констант, получаемых данными методами, в следующем разделе будет проведено их сравнение методов с алгоритмом А. Саксонова, описанным в разделе 4.1.

4.4. Сравнение собственных методов определения констант энергопотребления с методом А. Саксонова

Сравним методы, описанные в разделе 4.3., с методом А. Саксонова [19], а также с Power Profile устройства **Sony ZL (Android API Level 22)**, данные в котором приведены в **mA**. Данный телефон был выбран, поскольку производитель предоставил актуальные значения для данного устройства.

Стоит отметить, что вычисление по методу А. Саксонова поделилось ещё на 2 случая: высчитывание константы в **mAh** и дальнейший перевод в **mA**; изначальный перевод всех данных в **mA** и последующее вычисление константы энергопотребления. Эксперимент проходил на двух частотах: 1 и 5 Гц.

Таблица 3: Таблица констант энергопотребления

Устройство	**Алгоритм тестирования /источник данных**	**wifi.on (mA)**	**wifi.active (mA)**	**bluetooth.on (mA)**	**bluetooth.active (mA)**
Sony Zl (Android API level 22)	Saksonov 5 hz (mAh->mA)	1.15	0.8	-	9.9
	Saksonov 5 hz (mA)	1.15	0.79	-	9.9
	Saksonov 1 hz (mAh ->mA)	0.34	0.17	-	2.97
	Saksonov 1 hz (mA)	0.32	0.17	-	2.97
	Средние по средним (линия)	2.24	2.2	-	48.13
	Средние по средним (экспонента)	2.54	2.35	-	51.7
	Медиана по медиане (линия)	2.46	2.41	-	30.1
	Медиана по медиане (экспонента)	2.75	2.56	-	35.8
	Power Profile	2.7	26	1	27.7

Рассмотрим выводы, которые можно сделать на основании данной таблицы.

1. Независимо от способа вычисления методом А. Саксонова (вычисление в **mAh** и дальнейший перевод в **mA** или же изначальный перевод данных в **mA**), разницы в величине константы практически нет, поэтому можно считать оба подхода эквивалентными.

2. При вычислении с помощью метода А. Саксонова при помощи **adb** при частоте 5 Гц получаются большие константы, чем при частоте 1 Гц. Это обусловлено обоснованным в разделе 4.2. фактом, что при использовании **adb** с увеличением частоты дискретизации энергопотребление также увеличивается.

3. Для режима `bluetooth.on` получить константу не предоставляется возможным, поскольку при данном режиме работы **adb** мобильное устройство показывает нулевое энергопотребление.

4. Значения констант, полученные собственными методами, описанными в разделе 4.3., существенно ближе к величинам, указанным в Power Profile тестового устройства, чем получаемые методом А. Саксонова.

Итого, стало понятно, что использование метода А. Саксонова в качестве метода получения констант энергопотребления для модуля определения констант в Navitas Framework не является целесообразным. Это обусловлено тем, что разрабатываемое приложение использует для получения информации об энергопотреблении **adb**, однако, как было описано выше, метод А. Саксонова сильно зависит от частоты дискретизации, на которой происходит тестирование. Следовательно, получаемые им константы не подходят из соображений точности либо требуют предварительного отбора частоты для каждого конкретного устройства.

Собственные методы нахождения констант энергопотребления показали свою независимость от конкретной частоты тестирования, а также дали приближение лучше, нежели в методе А. Саксонова. Таким образом, при реализации модуля определения констант энергопотребления планируется использование методов из раздела 4.3., поправленных на необходимую продолжительность тестирования.

5. Реализация модуля определения собственных констант энергопотребления Wi-Fi и Bluetooth в Navitas Framework

В рамках задачи получения собственных констант энерпотребления Wi-Fi и Bluetooth для Power Profile произвольного устройства с учетом архитектуры Navitas Framework были выделены основные подзадачи.

1. Добавить в профилировщик NaviProf функциональность, позволяющую проведение многопоточного и многомодульного непрерывного тестирования энергопотребления модулей устройства с различной частотой дискретизации. Добавить поддержку нового режима тестирования — константы.

2. Внедрить в NaviPlugin поддержку нового режима работы — константы, а также функциональность для регрессионного анализа и оценки получаемых от NaviProf тестовых данных с целью выведения собственных констант энергопотребления для Power Profile произвольного устройства. Создать генератор XML для получения собственных Power Profile файлов.

3. Создать тестовое приложение NaviConstants в проекте NaviTests и реализовать экспериментальные классы для определения энерпотребления модулей Wi-Fi и Bluetooth на произвольном устройстве в следующих режимах: `wifi.on`, `wifi.scan`, `wifi.active`, а также `bluetooth.on`, `bluetooth.scan` и `bluetooth.active`.

5.1. NaviProf

NaviProf — это gradle-плагин [20], отвечающий за профилирование, то есть получение информации об энергопотреблении устройства, а также инструментовку тестов и преобразование полученных данных в JSON файл.

В рамках модуля NaviProf была реализована описанная ниже логика, являющаяся расширением и улучшением существовавшего подхода к тестированию устройств.

1. В связи с появлением нового режима тестирования — константы, изменены соответсвующие gradle-задачи, добавлено их конфигурирование выбранным видом тестирования.

2. Полностью изменен подход к получению тестовых данных: каждому исследуемому компоненту сопоставляется один или более логирующих потоков с возможностью совершать замеры на разных частотах дискретизации, либо, в отсутствие такой необходимости (например, во время режима профилирования), с максимальной возможной для **adb** частотой. Таким образом, была создана гибкая система тестирования, в которую легко добавить как новые компоненты устройства, так и виды тестирования.

3. С помощью инструмента **adb** реализовано получение информации об энергопотреблении модулей Wi-Fi и Bluetooth, а также запись полученных данных в промежуточные текстовые файлы, дополненных информацией о частоте и времени каждого замера.

4. С учетом новой информации об энергопотреблении, частоте и времени был переработан парсер текстовых файлов с возможностью последующего расширения другими компонентами и, соответсвенно, структура генерируемого JSON файла.

5.2. NaviPlugin

NaviPlugin — плагин к Android Studio [21], отвечающий за конфигурирование тестов, запускающихся на устройстве, профилирование устройства с помощью NaviProf, парсинг полученных данных, представленных в формате JSON, анализ собранных данных и визуализацию результатов профилирования, а также генерацию файлов с результатами оценки энергопотребления.

В рамках работы над NaviPlugin была реализована приведённая ниже функциональность, расширившая существовавшие возможности поддержкой нового режима работы — константы, основанного на собственном методе из раздела 4.3.

1. В соответствии с изменениями структуры JSON файла был изменен его парсинг, а также добавлены новые классы для хранения данных.

2. Реализован класс линейной регрессии.

3. С использованием линейной регрессии разработан модуль для анализа, являющийся реализацией описанного в разделе 4.3. собственного метода получения констант энергопотребления для Power Profile произвольного устройства.

4. Frond-end часть плагина, реализованная при помощи библиотеки **Swing** была расширена новым режимом работы, позволяющим выбрать специфические для него тестовые классы из NaviConstants. Получаемые данные о константах энергопотребления Wi-Fi и Bluetooth были визуализированны в виде таблицы.

5. Создан генератор XML файлов, позволяющий создавать файл Power Profile идентичный по структуре настоящему, проинициализированный полученными в ходе анализа константами энергопотребления модулей устройства в соответсвующих режимах работы.

В ходе работы над данным компонентом приложения была соблюдена существующая в NaviPlugin архитектура, в основе которой лежит паттерн MVVM.

5.3. NaviTests

Корректность работы плагина проверяется в специально созданном Android-проекте NaviTests [22], в рамках которого для специфического тестирования модулей Wi-Fi и Bluetooth на константы энергопотребления было создано приложение NaviConstants, содержащее тесты длительностью 2-е минуты данных компонентов устройства в перечисленных ниже режимах работы.

1. Тест модуля Wi-Fi в режиме `wifi.on`
   Происходит ожидание в течение всего теста с включённым пользователем заранее модулем Wi-Fi, так как методы, позволяющие сделать это автоматически, вырезаны из SDK Android и не являются безопасными.

2. Тест модуля Wi-Fi в режиме `wifi.scan`
   На данный момент все методы, позволяющие запустить сканирование модуля Wi-Fi, являются удалёнными из SDK Android, поэтому реализация данного теста невозможна по техническим причинам и является задачей будущих лет.

3. Тест модуля Wi-Fi в режиме `wifi.active`
   С включённым пользователем заранее модулем Wi-Fi при помощи API YouTube в созданном приложении происходит запуск трансляции и её проигрывание в течение всей работы теста.

4. Тест модуля Bluetooth в режиме `bluetooth.on`
   Модуль Bluetooth включается вручную пользователем либо автоматически в начале тестирования, а затем происходит ожидание в течение всего теста.

5. Тест модуля Bluetooth в режиме `bluetooth.scan`
   Модуль Bluetooth включается вручную пользователем либо автоматически в начале тестирования, а затем происходит сканирование в течение всего теста.

6. Тест модуля Bluetooth в режиме `bluetooth.active`
   Модуль Bluetooth включается вручную пользователем либо автоматически в начале тестирования, затем пользователь подключает Bluetooth-аудиоустройство и в течение всего теста на него транслируется заранее загруженная в приложение аудиозапись.

Пример вывода констант энергопотребления в Navitas Framework:

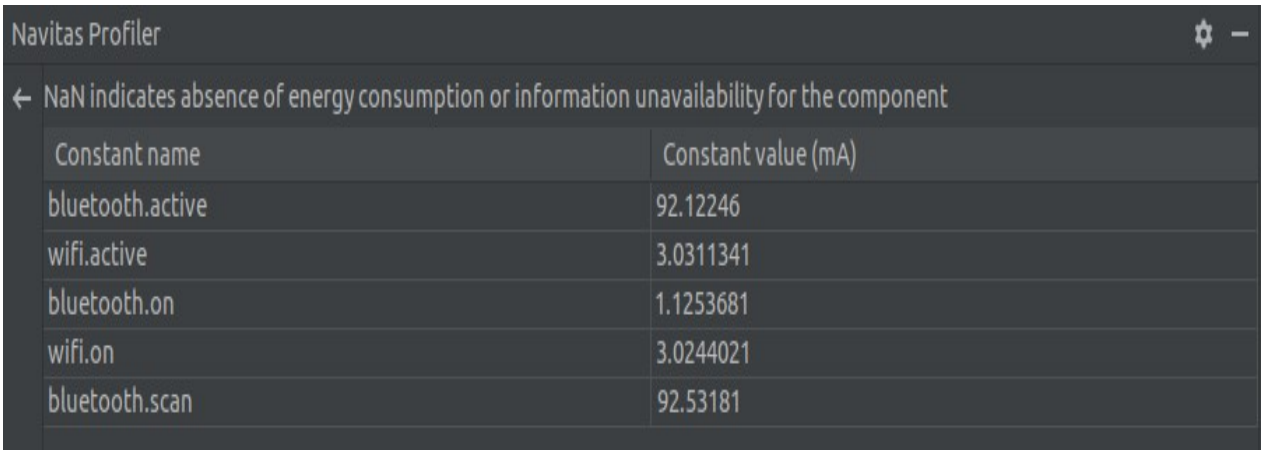

Рис. 20: Визуализация результатов определения констант энергопотребления

В режиме визуализации отображаются названия констант энергопотребления, соответствующие режимам работы компонентов устройства, а также результаты анализа — константы в миллиамперах (**mA**).

Пример вывода констант энергопотребления в виде Power Profile:

```xml
<?xml version="1.0" encoding="utf-8"?>
<device name="Android">
  <item name="bluetooth.active">92.12246</item>
  <item name="wifi.active">3.0311341</item>
  <item name="bluetooth.on">1.1253681</item>
  <item name="wifi.on">3.0244021</item>
  <item name="bluetooth.scan">92.53181</item>
</device>
```

Рис. 21: Power Profile, сгенерированный по результатам анализа

По итогам выведения констант энергопотребления модулей Wi-Fi и Bluetooth генерируется XML файл, являющийся полностью идентичным по структуре с Power Profile устройства и содержащий полученные значения.

5.4. Другие улучшения и исправления

- В рамках работы над Navitas Framework и в качестве помощи с защитой статьи по данному плагину на конференции `SEIM-2021` был реализован набор тестовых сценариев.

    1. Написан тест для оценки вариантов кода, в котором на предмет энергопотребления сравниваются два типа: отдельно реализованный класс с созданием объекта этого класса и анонимный класс, повторяющий реализацию первого. Navitas Framework показал, что первый сценарий использует больше энергии, нежели второй. Это связано с тем, что при использовании анонимного класса происходят оптимизации `ahead-of-time` компилятора `ART`, которые положительно сказываются на энергопотреблении.
    2. Также был создан тест, показывающий, сколько энергии потребляет код, работающий в отдельном потоке и создающий массив, заполняя его случайными числами, и затем записывающий его в файл. Для сравнения существует и второй вариант данного эксперимента: после заполнения массив ещё сортируется. Navitas Framework показал, что

вариант с сортировкой использует больше энергии, что соответствует действительности: при сортировке большого массива чисел растет процессорное время, число инструкций и, как следствие, общее энергопотребление.

- Увеличино количество поддерживаемых устройств за счёт рассмотрения большего числа вариаций вывода информации об энергопотребления средствами **adb**, так как основные ошибки возникали именно из-за их несоответсвия требуемой в Navitas Framework структуре. Таким образом, минимальная поддерживаемая версия Android в Navitas Framework расширилась до 5.1 (Android API Level 22). Однако рекомендуемая версия осталась без изменений — 7.0 и выше (Android API Level 24+).

- Было произведено улучшение распределения процессоров по кластерам, вызванное тем, что на некоторых Power Profile устройств предоставляется некорректная информация о нём. Такое несоответствие ранее приводило к аварийному завершению программы. Теперь присутствует возможность использовать больше процессоров, чем обозначено в кластерах, тогда как обратная функциональность присутствовала и раньше. Также это открывает возможности для профилирования устройств с использованием заведомо не принадлежащих им Power Profile. Таким образом, выбор верного Power Profile для тестирования и достижения наилучшей точности — ответственность пользователя, однако планируется поиск и реализация более продвинутого решения.

- Увеличина стабильность и потокобезопасность парсеров и генераторов текстовых, JSON и XML файлов.

- Текстовые файлы и JSON, содержащие логи тестирования устройства, сделаны доступными пользователю и не удаляются после окончания экспериментов.

6. Результаты

Благодаря работе над учебной практикой были достигнуты следующие результаты.

1. Проведен обзор энергопрофилировщиков и профилей питания Android.
2. Средствами метода **adb** найден способ, позволяющий получить собственные константы энергопотребления для файла Power Profile конкретного устройства.
3. Исследована связь между частотой дискретизации и изменением энергопотребления в экспериментах.
4. Было найдено устройство с актуальными значениями в файле Power Profile и проведено сравнение получаемых констант методом А. Саксонова с ними, а также с константами, которые вычисляются собственными методами.
5. Разработана и внедрёна система автоматического получения констант Power Profile для модулей Wi-Fi и Bluetooth в Navitas Framework и их вывод, как и Power Profile устройства, в виде XML-файла.
6. Проведена работа над улучшением существующей функциональности Navitas Framework, повышением стабильности и увеличением числа поддерживаемых устройств.

Благодаря полученным результатам, виден дальнейший вектор работы над проектом Navitas Framework со следующей задачей:

1. Добавить валидацию используемого для тестирования Power Profile по CPU и получение числа кластеров и ядер другими способами.

Список литературы

[1] Number of smartphone users worldwide from 2016 to 2021:
https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] Mobile Operating System Market Share Worldwide Oct 2019 - Oct 2020:
https://gs.statcounter.com/os-market-share/mobile/worldwide

[3] Estimating the number of Wi-Fi users via smartphones Jan 2019:
https://www.bluetooth.com/wp-content/uploads/2018/04/2019-Bluetooth-Market-Update.pdf

[4] Total Annual Bluetooth Device Shipments in 2019:
https://www.bluetooth.com/wp-content/uploads/2018/04/2019-Bluetooth-Market-Update.pdf

[5] Estimating the number of Wi-Fi users via smartphones Jan 2019:
https://www.bluetooth.com/wp-content/uploads/2018/04/2019-Bluetooth-Market-Update.pdf

[6] Default Power Profile by Google:
https://android.googlesource.com/platform/frameworks/base/+/master/core/res/res/xml/power_profile.xml

[7] Android Debug Bridge Developers Documentation:
https://developer.android.com/studio/command-line/adb

[8] Application Sandbox in Android:
https://source.android.com/security/app-sandbox?hl=en

[9] PowerTutor - A Power Monitor for Android-Based Mobile Platforms:
http://ziyang.eecs.umich.edu/projects/powertutor/index.html

[10] Android Developers Documentation about Wi-Fi Manager:
https://developer.android.com/reference/android/net/wifi/WifiManager

[11] Wi-Fi module implementation in PowerTutor:
https://github.com/msg555/PowerTutor/blob/master/src/edu/umich/PowerTutor/components/Wifi.java

[12] Robert P. Dick, Lide Zhang, Birjodh Tiwana and Zhiyun Qian. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 105-114, october 2010.
https://dl.acm.org/doi/abs/10.1145/1878961.1878982

[13] Android Developers Documentation about Energy Profiler:
https://developer.android.com/studio/profile/energy-profiler

[14] Profile battery usage with Batterystats and Battery Historian:
https://developer.android.com/topic/performance/power/setup-battery-historian

[15] Analyze power use with Battery Historian:
https://developer.android.com/topic/performance/power/battery-historian

[16] Power Profiles for Android:
https://source.android.com/devices/tech/power

[17] Power value settings:
https://source.android.com/devices/tech/power/values#values

[18] Bash-scripts used for Wi-Fi and Bluetooth power consumption research:
https://github.com/Stanislav-Sartasov/Navitas-Framework/tree/wifi_bluetooth_research

[19] Andrey Saksonov. Method to Derive Energy Profiles for Android Platform, Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany, 2014.
http://www.se.uni-oldenburg.de/documents/saksonov-MA2014.pdf

[20] NaviProf plugin implementation:
https://github.com/Stanislav-Sartasov/Navitas-Framework/tree/master/NaviProf

[21] NaviPlugin implementation:
https://github.com/Stanislav-Sartasov/Navitas-Framework/tree/master/Navitas-Plugin

[22] NaviTests implementation:
https://github.com/Stanislav-Sartasov/Navitas-Framework/tree/master/UI-Testing-Samples

[23] Useful links for System Literature Review:
https://docs.google.com/spreadsheets/d/17SRvTyuGqcMrsQe856b6pDOe-9Ynszg8KhhAeGrcvvM/edit#gid=0