Учебно-методическое пособие: IoT и туманные вычисления (Fog Computing) – часть 1

Раздел 1: Теория (IoT и Fog)

Что такое IoT? Интернет вещей (Internet of Things, IoT) – это сеть взаимосвязанных «умных» устройств, которые собирают, обмениваются и обрабатывают данные через интернет[1]. Такие устройства оснащены датчиками, актуаторами, микроконтроллерами и сетевыми модулями, что позволяет им взаимодействовать друг с другом и с облачными сервисами без участия человека. IoT открывает возможности для умных систем в разных сферах – от умного дома до промышленности, здравоохранения и умных городов[2]. Благодаря IoT становится возможным создание новых сервисов и улучшение качества жизни: например, мониторинг инфраструктуры, дистанционное управление устройствами, автоматизация рутины и др.

Архитектура IoT: Edge – Fog – Cloud. Для эффективного функционирования IoT-системы организуют многоуровневую архитектуру, включающую краевые устройства, промежуточные узлы туманных вычислений и облако. Основные уровни и их роли:

- Edge (краевой уровень) – все конечные IoT-устройства и сенсоры, которые непосредственно генерируют данные на «периферии» сети. Часто оснащены микроконтроллерами, могут выполнять простейшую локальную обработку данных или сразу передавать их дальше[3]. Примеры: датчики температуры, умные розетки, носимые гаджеты.

- Fog (туманный уровень) – узлы промежуточной обработки, расположенные ближе к источникам данных (например, локальные серверы, шлюзы или мощные роутеры). Fog-узлы собирают данные от групп Edge-устройств, выполняют предварительную обработку (агрегацию, фильтрацию) и

решают, какие данные отправить в облако[4][3]. Таким образом, Fog является *«мини-облаком на периферии»*, сокращая расстояние до источника данных.

- Cloud (облачный уровень) – централизованные дата-центры или облачные платформы, где выполняется глубинная обработка, долговременное хранение данных и сложная аналитика. Облако получает уже отфильтрованные/агрегированные данные от Fog-уровня или Edge, и предоставляет глобальный доступ к результатам через приложения и сервисы[3].

Зачем нужны Fog-узлы? В чем отличие от облака? Fog computing был предложен для устранения ограничений сугубо облачной IoT-архитектуры. В традиционной схеме все данные с устройств отправляются в облако, что создаёт большую нагрузку на сеть и приводит к задержкам[5]. Это неприемлемо для приложений, требующих мгновенного отклика (например, управление беспилотным транспортом или аварийный мониторинг)[6]. Туманные вычисления позволяют разместить часть ресурсов обработки и хранения *ближе к данным*, на промежуточном уровне. В отличие от облака, которое централизовано, Fog-узлы децентрализованы: они находятся «на полпути» между устройствами и облаком, зачастую в локальной сети или географически близко. За счёт этого: (1) Сокращается задержка – путь данных короче, часть запросов обслуживается локально[4]. (2) Снижается трафик в облако – фильтруются избыточные данные, в центр отправляется только нужная информация[4]. (3) Повышается надёжность и автономность – при потере связи с интернетом локальный Fog-узел может временно обслуживать систему. Таким образом, Fog дополняет облако, обеспечивая распределённую обработку. *Принципиальная разница*: облако – “core” (ядро сети), fog – “edge” (периферия), ближе к источникам, иногда рассматривается как расширение облака на край сети.

Архитектуры Fog. Для стандартизации подходов к туманным вычислениям промышленность разработала эталонные модели. В частности, консорциум

OpenFog (ныне Industrial IoT Consortium) опубликовал OpenFog Reference Architecture (утверждён как стандарт IEEE 1934-2018)[7]. Этот документ описывает универсальные принципы построения Fog-систем: иерархию уровней, сетевую инфраструктуру, требования к узлам, безопасность и др. Существование таких референсных архитектур помогает проектировать совместимые решения – от умных городов до автономных автомобилей – где задействованы Edge, Fog и Cloud.

Протоколы связи в IoT. Для обмена данными между узлами IoT применяются специальные легковесные протоколы. Один из ключевых – MQTT (Message Queuing Telemetry Transport). MQTT – это простой и эффективный протокол обмена сообщениями по модели “publish/subscribe” (издатель/подписчик), разработанный для устройств с ограниченными ресурсами и сетей с высоким латентностью или низкой пропускной способностью[8]. В MQTT устройства не общаются напрямую друг с другом, вместо этого они отправляют сообщения определённым “топикам” на брокер-сервер, который рассылает их всем подписчикам этого топика. Преимущества MQTT для IoT: минимальный объём служебных данных (заголовок сообщения всего 2 байта), поддержка тысяч одновременно подключённых клиентов, три уровня качества доставки сообщений (QoS 0, 1, 2) для надёжности, поддержка шифрования и авторизации[9][10]. Благодаря этому MQTT стал стандартом де-факто для обмена телеметрией между датчиками, контроллерами и облаком. Помимо MQTT часто используются: HTTP/HTTPS (для передачи данных в веб-сервисы, но более “тяжёлый” протокол), иногда CoAP (UDP-протокол для IoT), AMQP, WebSockets и др. Выбор протокола зависит от требований по энергоэффективности, надёжности и скорости в конкретном проекте.

Роль симуляции в разработке IoT-систем. Симуляция позволяет предварительно проверить и настроить IoT-архитектуру в виртуальной среде до её реального развёртывания. Настоящая IoT-инфраструктура с сотнями

устройств может быть дорогостоящей и сложной, а эксперименты на ней трудно контролируемыми. Поэтому исследователи и инженеры используют программные инструменты (например, iFogSim, собственные программы и др.) для моделирования поведения IoT-систем [11]. В симуляции можно задать необходимое число устройств, топологию сети, задержки передачи, объёмы данных, стратегии обработки – и получить оценки производительности: end-to-end задержку, загрузку каналов, объём трафика, энергопотребление, нагрузку на узлы, стоимость обработки и пр.[12]. Симулятор позволяет варьировать параметры (например, расположение fog-узлов, политику распределения задач) и *в повторяемых условиях* замерять метрики эффективности. Это позволяет показать, например, узкие места архитектуры, влияние масштабирования. В нашем курсе мы уделяем особое внимание симуляции: она используется в лабораторных работах для оценки задержек и других показателей IoT-системы. Полученные через модель результаты помогают обосновать архитектурные решения и оптимизации, прежде чем реализовывать проект на практике.

Инструменты симуляции в области туманных и облачных систем и при разработке IoT-систем.

Типовые шаблоны Edge–Fog–Cloud стоит брать из OpenFog/IEEE 1934, ETSI MEC, oneM2M+MEC и облачных референс-архитектур.

Для моделирования “без самописных проектов” лучше всего подходят iFogSim/iFogSim2 и YAFS (академические симуляторы), а для стандартной телеком-архитектуры – изучить ETSI MEC Sandbox и белые книги MEC.

Для учебных проектов и демонстраций удобно использовать облачные IoT-платформы и лёгкие стенды на ThingsBoard/Blynk/AWS/Azure.

Раздел 2: Обзор лабораторных работ (3.1 – 3.3)

Лабораторная работа 3.1: Моделирование задержек в распределённой системе «умный дом» (robots-fog-mini-sim)

Цель и задачи: Цель первой работы – изучить принципы построения схем распределённых IoT-систем в соответствии со стандартами (ГОСТ 2.701/2.702), освоить инструменты Google Colab, PlantUML Editor и draw.io для визуализации архитектуры, а также приобрести навык создания структурных и функциональных диаграмм (Deployment- и State-диаграмм) для IoT-системы. Проект основан на примере системы *«умный дом»*, и студентам предлагается научиться единообразно отображать её архитектуру и процессы, следуя стандартам (чтобы разные специалисты «говорили на одном языке» при проектировании сложных систем).

Структура модели/архитектуры: В качестве объекта моделирования рассматривается упрощённая распределённая IoT-система умного дома, состоящая из пяти основных компонентов, распределённых по трём уровням вычислительной архитектуры:

- Краевой уровень (Edge): *Робот-датчик* – устройство сбора сырых данных (сенсор на краю сети).
- Туманный уровень (Fog): *Fog-обработчик* – узел локальной обработки данных (промежуточный сервер-шлюз).
- Облачный уровень (Cloud): *Ноутбук-сервер* – облачный сервер для аналитики и хранения данных.
- Связующие компоненты: *Робот-курьер* (подвижное устройство для транспортировки данных между узлами) и *Смартфон* (локальный шлюз, выполняющий буферизацию сообщений и пересылку данных дальше). В совокупности эти элементы образуют цепочку передачи данных от датчика

через Fog-уровень в облако (Edge → Fog → Cloud). Диаграмма компонентов системы отражает роли каждого узла и направление потоков данных между ними.

Основные параметры (входные данные): Данная работа носит схемотехнический характер, поэтому явных числовых входных данных не предусматривает – моделируется фиксированный сценарий «умного дома» с заданными компонентами. Студенты работают с *логической моделью* системы (определённый набор устройств и связей) и следуют стандартам оформления схем. Индивидуального набора числовых параметров нет, однако каждый студент может использовать собственные обозначения компонентов и типов соединений на диаграммах (например, переименовать узлы, указать разные протоколы связи – Wi-Fi, ZigBee, Ethernet и т.п.). Это обеспечивает уникальность схем, сохраняя при этом общую архитектуру системы.

Ключевые метрики: *(В этой лабораторной работе количественные метрики не рассчитываются; акцент делается на визуальном представлении структуры системы, а не на численных показателях производительности.)* Тем не менее, на диаграммах отражается сквозной путь данных от датчика к облаку (end-to-end), что закладывает понимание будущих метрик задержки. Например, на схемах показано, какие компоненты последовательно обрабатывают данные и где могут возникать задержки, хотя численно они здесь не измеряются.

Описание экспериментов (практических заданий): Студенты выполняют серию заданий по построению диаграмм архитектуры распределённой системы:

- *Диаграмма архитектуры через Python:* С помощью Google Colab и библиотеки Diagrams на Python студенты генерируют программно deployment-диаграмму системы «умный дом». Пример кода создаёт узлы (робот-датчик, Fog-шлюз, облачный сервер и др.) и связи между ними; затем предлагается модифицировать этот код – например, добавить второго *робота-датчика* – чтобы обновить схему и отразить параллельные источники данных. Такой подход демонстрирует, как можно автоматически визуализировать архитектуру.

- *Построение схем в PlantUML:* Далее, используя онлайн-редактор PlantUML, студенты вручную создают две диаграммы. Deployment-диаграмма должна отображать как минимум 3 краевых устройства, 2 Fog-узла и 1 облачный сервер, причём рекомендуется задать собственные названия компонентов и указать типы соединений (например, радиоканал ZigBee между датчиками и шлюзом, проводное Ethernet-соединение с облаком и т.п.). Кроме того, строится State-диаграмма для одного из узлов – в данном случае для *Робота-курьера*, с изображением состояний этого устройства (например: Зарядка, Ожидание_задания, Навигация_к_смартфону, Передача_данных и возможное состояние Перегрузка при переполнении буфера). Эти диаграммы отражают соответственно структуру системы и логику состояний узла.
- *Знакомство с Draw.io:* Дополнительно в методических указаниях приведён пример создания схемы во внешнем графическом инструменте Draw.io (diagrams.net). Студентам предлагается альтернативный путь – вручную нарисовать схему в визуальном редакторе, перетаскивая на холст условные обозначения устройств и соединений. Это не обязательный шаг, но он показывает другой способ визуализации для тех, кто предпочитает графический интерфейс.

Формат сдачи: Результатом выполнения работы является краткий отчёт, в который студент включает созданные диаграммы и необходимые пояснения. В частности, Deployment-диаграмма и State-диаграмма должны быть сохранены в форматах изображений (например, PNG) и приложены к отчёту. В отчёте следует описать кратко состав системы (какие узлы на схеме присутствуют) и пояснить, что отображают диаграммы. Требуется убедиться, что диаграммы читаемы и соответствуют требованиям (есть легенда или подписи компонентов, указаны все связи). Отчёт сдаётся в электронном виде с вложенными картинками схем.

Индивидуальные параметры и используемые метрики: В ЛР 3.1 отсутствует деление задания на индивидуальные варианты с уникальными параметрами – все работают с одной и той же архитектурой умного дома.

Индивидуализация происходит лишь за счёт творческого оформления схем: каждый студент сам выбирает названия узлов и детали обозначений соединений. Показатели эффективности системы (задержки, нагрузки и пр.) на данном этапе не рассчитываются, поэтому никаких числовых метрик студент не вычисляет. Критериями оценки являются правильность и наглядность построенных схем, соответствие стандарту (ГОСТ/UML) и полнота отображения компонентов и связей.

Лабораторная работа 3.2: Моделирование эталонной архитектуры Edge–Fog–Cloud с задержками

Цель и задачи: Во второй работе студенты переходят от статической схемы к динамическому моделированию. Цель – понять принципы работы распределённых систем и научиться моделировать и анализировать задержки передачи данных в IoT-системе на примере «умного дома». В рамках имитации ставятся следующие задачи:

- проследить путь данных от датчика до облачного сервера (каким образом информация проходит через все узлы),
- измерить сквозную задержку доставки данных (end-to-end delay) от краевого устройства до облака,
- проанализировать работу буфера смартфона (очереди сообщений) на промежуточном узле,
- поэкспериментировать с параметрами системы (изменяя настройки задержек и режим отправки данных) и оценить влияние этих изменений на показатели системы.

Структура модели/архитектуры: Моделируемая архитектура соответствует схеме из ЛР 3.1, то есть включает компоненты умного дома на уровнях Edge, Fog и Cloud. В симуляции задействованы: краевой датчик (*робот-датчик*),Fog-узел локальной обработки (*Fog-обработчик*), транспортный узел (*робот-курьер*),

шлюз с буферизацией (*смартфон*) и облачный сервер (*ноутбук-сервер*). Таким образом, формируется цепочка передачи задач: датчик → Fog → курьер → смартфон → облако, повторяющая логическую структуру из первой работы. На каждом участке пути данные могут задерживаться, что в сумме даёт сквозную задержку. Данная архитектура считается *эталонной (базовой)* для проекта, от которой затем будут отталкиваться в последующих экспериментах.

Основные параметры (входные данные): Для имитационного моделирования задаются исходные параметры, определяющие поведение системы:

- Число задач (сообщений), генерируемых в симуляции. Например, моделируется выполнение *n* задач (итераций), отправляемых датчиком в облако.
- Диапазоны задержек для каждого компонента: в коде симуляции время задержки на каждом этапе генерируется случайным образом в заданных пределах. По умолчанию используются диапазоны: датчик – 20–60 мс, Fog-обработчик – 30–80 мс, робот-курьер – 10–40 мс. Эти величины имитируют вариативность времени обработки/передачи на соответствующих узлах.
- Интервал чтения сообщений смартфоном (параметр read_interval_ms) – периодичность, с которой смартфон пересылает накопленные данные на сервер. В исходной конфигурации задан некоторый стандартный интервал (например, 100 мс); впоследствии в экспериментах его значение изменяется (ускоряется или замедляется). Этот параметр влияет на размер очереди смартфона: чем реже устройство отправляет данные, тем больше сообщений скапливается в буфере.

Ключевые метрики: Основной оцениваемый показатель – сквозная задержка (end-to-end delay) выполнения задачи, т.е. время от генерации данных на датчике до получения их облачным сервером. В ходе симуляции вычисляется задержка для каждой задачи, и результаты отображаются на графике сквозной задержки по номерам задач. По этому графику можно определить среднее время

доставки и разброс значений. Второй важный показатель – состояние очереди смартфона (буфера) во времени. На выходе симуляции строится график динамики размера буфера смартфона, показывающий, как в процессе обработки задач растёт или уменьшается число ожидающих сообщений. Этот график характеризует способность промежуточного узла справляться с входящим потоком данных. *(Примечание: Метрики распределения задержек высокого порядка – например, 95-й перцентиль – в данной работе ещё не рассчитываются, фокус сделан на среднем времени и визуальном анализе графиков задержки и очереди.)*

Описание экспериментов: В лабораторной работе предусмотрено два эксперимента, позволяющих исследовать влияние отдельных компонентов на задержку и найти пути оптимизации:

- Эксперимент 1: Оптимизация Fog-узла. Цель – определить, как время обработки на уровне Fog влияет на общую производительность системы. Студент находит в коде участок, задающий задержку Fog-обработчика (по умолчанию random.randint(30, 80) мс) и фиксирует исходное значение диапазона в отчёте. Затем выполняется запуск симуляции с этими параметрами и собираются результаты – в частности, средняя сквозная задержка системы (по результатам всех задач) и график задержек. Далее студент уменьшѐт задержку Fog-узла, подставляя новый оптимизированный диапазон, например 10–40 мс, и повторно запускает моделирование. Ожидается, что средняя сквозная задержка заметно снизится; методические указания ориентируют на снижение порядка 25–30%. Полученные значения и графики (до и после оптимизации) фиксируются, и делается вывод, оправдались ли ожидания ускорения обработки при улучшении Fog-узла.
- Эксперимент 2: Настройка буфера смартфона. Цель – исследовать влияние частоты отправки данных с промежуточного узла на размер очереди и задержки. Студент изменяет значение параметра read_interval_ms в функции

симуляции, отвечающего за периодичность передачи сообщений смартфоном. Предлагается проверить два случая: ускоренная отправка (например, интервал уменьшен до 60 мс) и замедленная отправка (интервал увеличен до 200 мс). Для каждого случая проводится отдельный прогон симуляции, и строятся графики заполнения буфера смартфона во времени. Сравнивая графики, студент наблюдает, что при небольшом интервале чтения буфер либо не успевает значительно заполниться (сообщения отправляются чаще и очередь быстро освобождается), тогда как при увеличенном интервале буфер растёт значительно больше (смартфон копит данные дольше). Эксперимент позволяет наглядно увидеть компромисс между задержкой отправки и размером очереди. В отчёте необходимо указать, соответствует ли поведение системы ожиданиям (например, что при более частой отправке очередь в среднем меньше).

Формат сдачи: По завершении работы студент оформляет отчёт, включающий описание проведённых экспериментов, использованные значения параметров и полученные результаты. В отчёте должны присутствовать графики: сквозной задержки (до и после оптимизации Fog-узла) и графики изменения буфера смартфона при разных интервалах отправки. К каждому графику добавляется пояснение – что он показывает и какие выводы из него следуют. Также приводятся рассчитанные числовые показатели, такие как средняя сквозная задержка в исходном и оптимизированном вариантах, и формулируются выводы по каждому эксперименту. В конце отчёта студент делает общий вывод о том, как изменения параметров (ускорение Fog-узла, изменение интервала отправки) повлияли на работу системы. Отчёт с графиками и выводами сдаётся преподавателю для проверки.

Индивидуальные параметры и используемые метрики: В ЛР 3.2 все студенты исследуют одну и ту же базовую конфигурацию умного дома, поэтому индивидуальных вариантов с разными исходными данными не предусмотрено – задания и сценарий симуляции едины для всех. Основные параметры (диапазоны

задержек, интервалы) заданы в методических указаниях, и все обучающиеся используют их по умолчанию. Метрики, которые должны быть отражены в анализе, тоже общие: это средняя сквозная задержка системы (до/после изменений) и характеристики очереди смартфона (например, максимальный размер или динамика заполнения во времени). Каждый студент, выполнив эксперименты, рассчитывает эти показатели для своей симуляции и включает в отчёт. Таким образом, сравнение между работами разных студентов сводится не к разным исходным данным, а к корректности проведённых экспериментов и анализу графиков. *Примечание:* на этом этапе не требуются сложные статистические метрики (типа перцентилей) – достаточно средних значений и качественного анализа поведения системы.

Лабораторная работа 3.3: Моделирование дополнительных параметров IoT-системы (тип устройства, размер очереди) и оптимизация

Цель и задачи: Третья лабораторная работа посвящена масштабированию и оптимизации эталонной модели Edge–Fog–Cloud. Студентам предстоит исследовать, как различные параметры системы влияют на производительность, и научиться подбирать оптимальные настройки. В числе основных целей:

- исследовать влияние количества устройств на каждом уровне архитектуры (Edge, Fog, Cloud) на сквозную задержку в системе (как рост числа датчиков, шлюзов или серверов сказывается на времени доставки данных),
- сравнить производительность системы при разных типах краевых устройств: стационарных vs мобильных (например, статичные датчики против подвижных роботов) и оценить, как мобильность устройств влияет на задержки и стабильность передачи,
- проанализировать, как ограничение размера очереди на Fog-узлах влияет на общую производительность (что происходит, если у промежуточного узла небольшая емкость буфера, либо наоборот очень большая),

- научиться оптимизировать параметры системы с целью снижения задержек и устранения «узких мест» (т.е. подобрать такое соотношение количества устройств и настроек, при котором система работает наиболее эффективно). Эти задачи направлены на углубление понимания того, как масштаб и конфигурация компонентов определяют поведение распределённой IoT-системы.

Структура модели/архитектуры: Рассматривается эталонная архитектура «Край–Туман–Облако», расширенная до большого числа узлов. По сути, это та же многоуровневая модель, что и в предыдущих работах, но допускающая произвольное масштабирование количества устройств на каждом уровне. В симуляции можно задать, к примеру, “малую” конфигурацию (несколько десятков датчиков, десяток Fog-узлов, пару облачных серверов) или “большую” (сотни и тысячи узлов). Методические указания предлагают три ориентировочных масштаба для начала:

- Малая система: 100 краевых устройств, 10 Fog-узлов, 3 облачных сервера – имитирует небольшой проект (например, умный дом с большим числом датчиков и одним локальным шлюзом).
- Средняя система: 500 Edge-устройств, 50 Fog-узлов, 10 облачных серверов – более масштабная архитектура, распределённая по нескольким локальным узлам и облачному центру.
- Крупная система: 2000 Edge-устройств, 200 Fog-узлов, 20 Cloud-серверов – модель промышленного масштаба, с большим количеством устройств на всех уровнях.

Эти варианты используются, чтобы проследить тенденцию изменения задержек при росте системы (ожидается, что с увеличением числа устройств средняя задержка возрастает из-за ростущей нагрузки на сеть и узлы обработки). Архитектурно, система по-прежнему состоит из уровней Edge, Fog и Cloud, но теперь на каждом уровне может быть *множество* устройств,

взаимодействующих по схеме «многие-ко-многим» (множество датчиков отправляют данные через множество Fog-узлов в облачный кластер). Также вводятся различия в типах узлов (некоторые Edge-устройства могут двигаться) и в конфигурации очередей Fog-уровня.

Основные параметры (входные данные): В данной работе внимание уделяется *параметрам конфигурации* системы, которые студент будет изменять:

- Количество устройств на каждом уровне – задаётся тройкой чисел: [число Edge-устройств, число Fog-узлов, число Cloud-серверов]. Именно эти величины определяют масштаб моделируемой системы и могут варьироваться в экспериментах.
- Тип краевых устройств – в модели предусмотрено два типа: *стационарные* (фиксированные датчики, постоянно подключенные к сети) и *мобильные* (устройства, которые могут перемещаться, временно терять связь, иметь дополнительные задержки обмена при движении). Параметр типа влияет на характер генерируемых задержек: мобильные узлы могут, например, вносить более высокую вариативность или паузы в передаче данных.
- Емкость очереди Fog-узла (queue_capacity) – максимальное количество задач, которое каждый Fog-узел может удерживать в ожидании обработки. Если поступает больше сообщений, чем вместимость, лишние могут отбрасываться или вызывать задержку до освобождения очереди. Этот параметр важен для оценки устойчивости системы к перегрузкам.

Помимо перечисленных, модель использует и прежние настройки (задержки на передачу и обработку, частоту отправки и пр.), но именно перечисленные три являются новыми *переменными*, с которыми экспериментируют студенты. В исходном коде симуляции предусмотрена специальная функция simulate_custom_config(), позволяющая быстро настроить конфигурацию системы перед запуском, установив значения количества узлов, типа устройств и ёмкости очередей согласно замыслу эксперимента.

Ключевые метрики: Для оценки эффективности разных конфигураций вводится расширенный набор метрик производительности:

- Средняя сквозная задержка – среднее время прохождения данных от краевого устройства до облака (усредняется по всем задачам в ходе симуляции); основной показатель, характеризующий общую задержку в системе.
- 95-й перцентиль задержки – значение задержки, ниже которого укладываются 95% всех задач. Этот показатель отражает *близкий к худшему* для большинства задач случай (то есть, 5% самых медленных передач могут быть длиннее этого времени) и служит для оценки гарантированности обслуживания.
- Максимальная задержка – наибольшее зарегистрированное время доставки данных среди всех задач. Этот экстремальный показатель указывает на возможные проблемные ситуации или узкие места, которые приводят к особо длительным задержкам.
- Средняя загрузка Fog-узлов – в контексте методики под этим понимается *среднее время ожидания задач в очередях Fog-узлов* (или альтернативах, средняя длина очереди). Фактически, эта метрика показывает, насколько заняты промежуточные узлы: большое время ожидания в Fog означает, что узлы перегружены и задачи там копятся.

Все указанные метрики вычисляются в ходе симуляции для каждой протестированной конфигурации. Например, строится сводный график или таблица, где указываются эти показатели для разных вариантов системы (малая, средняя, большая и т.д.). Это позволяет сравнить производительность разных конфигураций не только по среднему времени, но и по надёжности (percentile, max) и степени загрузки ресурсов (очереди Fog).

Описание экспериментов: Лабораторная работа включает три эксперимента, каждый из которых направлен на исследование отдельного аспекта конфигурации эталонной модели:

1. Эксперимент 1: Масштабирование системы. Студент изучает, как изменение количества устройств на разных уровнях влияет на задержку. Для начала выполняется *базовый анализ масштабируемости*: проводится серия запусков симуляции на трёх базовых конфигурациях – малой, средней и крупной – чтобы зафиксировать тенденцию роста задержек при росте системы. Далее переходит к *своему индивидуальному варианту* (см. ниже) и проводит углублённый анализ: последовательно изменяет один из параметров конфигурации, удерживая два других постоянными. Например, при фиксированных Fog- и Cloud-узлах увеличивает число Edge-устройств на +25%, +50%, +75%, +100% и смотрит, как это скажется на средней задержке; затем возвращается к исходному количеству Edge и пробует увеличивать число Fog-узлов на +10%, +20%, ... +50% (при фиксированных Edge и Cloud); аналогично проверяет влияние роста числа облачных серверов (например, в 2x, 3x раза) при фиксированных Edge/Fog. Каждое такое изменение конфигурации – по сути отдельный прогон симуляции – сопровождается фиксацией ключевых метрик и последующим анализом: студент выясняет, какое звено (Edge, Fog или Cloud) сильнее влияет на задержку и где наступает насыщение производительности. Например, может оказаться, что добавление облачных серверов практически не снижает задержку, если узким местом остаётся перегруженный Fog-уровень. Итоги эксперимента 1 – понимание, при каком соотношении устройств система сбалансирована, а при каком – один из уровней становитсяBottle-neck.

2. Эксперимент 2: Анализ типов устройств. В этом опыте моделируется сценарий, когда в системе используются разные типы Edge-устройств. Студент настраивает симуляцию на две ситуации: вариант A – все краевые устройства стационарные, вариант B – все краевые устройства мобильные (или значительная их часть мобильна). Оба варианта запускаются (желательно на той же самой конфигурации чисел устройств для чистоты сравнения), и собираются метрики производительности. Задача – сравнить результаты: например, вычислить

разницу в средней сквозной задержке и 95-м перцентиле между сценариями со стационарными vs мобильными узлами. Ожидается, что мобильность может добавлять непредвиденные задержки (из-за потери пакетов, переподключений или ограничений беспроводной связи), что проявится в большем разбросе задержек и, возможно, росте процентовилей/максимумов. Студент делает вывод, насколько существенно мобильные устройства влияют на качество обслуживания по сравнению с фиксированными сенсорами.

3. Эксперимент 3: Оптимизация очередей Fog-узлов. Здесь исследуется параметр ёмкости очереди на промежуточных узлах. Студент варьирует queue_capacity Fog-узлов и наблюдает, как меняются метрики системы. Например, можно испытать очень маленькую очередь (способную хранить лишь 1–2 задачи), очередь среднего размера и практически неограниченную. Для каждого случая запускается симуляция, и особое внимание уделяется максимальной задержке и загрузке Fog-узлов. Маленькая очередь может приводить к тому, что при наплыве задач некоторые из них будут отвергнуты или ждать повторной отправки, что увеличит максимум задержки или приведёт к потерям. Слишком большая очередь, напротив, не отбрасывает задачи, но может увеличивать время ожидания (среднюю задержку), если задачи долго стоят в очереди. Цель эксперимента – найти баланс: достаточную ёмкость, чтобы почти все задачи обрабатывались без потерь, но при этом не создавать избыточных задержек. Студент определяет, при каком значении capacity система достигает приемлемого компромисса, и фиксирует это как *оптимальное*. В отчёте описывается, как изменение размера очереди повлияло на 95-й перцентиль и максимальную задержку (например, уменьшилось ли число экстремально долгих задач при увеличении очереди, или напротив средняя задержка выросла).

Формат сдачи: По итогам работы студент готовит подробный отчёт, структурированный по экспериментам. В начале отчёта указывается исходная конфигурация (вариант) системы, принятая для экспериментов, и расчетные

значения основных метрик для этой конфигурации (средняя задержка, 95-й перцентиль и др.) – как отправная точка. Далее для каждого эксперимента приводятся: описание проведения (какие изменения параметров делались), таблицы или графики результатов, и анализ этих результатов. Например, в эксперименте 1 целесообразно оформить таблицу сравнения метрик при разных комбинациях устройств (или график, показывающий зависимость задержки от количества устройств на том или ином уровне). В эксперименте 2 – возможно, график сравнения задержек для мобильных vs стационарных устройств. В эксперименте 3 – график, показывающий, как меняется максимальная/средняя задержка в зависимости от размера очереди. В конце отчёта (раздел «Выводы») студент суммирует ключевые инсайты: как масштабирование влияет на систему, какой тип устройств предпочтительнее в контексте задержек, и какой баланс очереди и ресурсов рекомендуется для эффективной работы. Отчёт должен продемонстрировать понимание, какие факторы ограничивают производительность и как их можно смягчить настройками. Форма сдачи – электронный отчёт с включёнными графиками, пояснениями и выводами.

Индивидуальные параметры и используемые метрики: В ЛР 3.3 каждому студенту выдаётся индивидуальный вариант конфигурации системы для эксперимента 1. Варианты представлены в таблице методических указаний. Например, вариант 1: 100 Edge-устройств, 5 Fog-узлов, 2 Cloud-сервера – ситуация, где на один Fog приходится очень много датчиков (потенциальная перегрузка Fog); вариант 4: 500 Edge, 10 Fog, 3 Cloud – относительно мало Fog-узлов на большое число датчиков (узкое место на Fog-уровне); вариант 8: 2000 Edge, 100 Fog, 15 Cloud – весьма масштабируемая система с большим числом промежуточных узлов. Студент берет *свой* вариант как базу и проводит все эксперименты именно с ним, что добавляет уникальности результатам каждого. Тем не менее, метрики оценки для всех общие и заранее определённые: каждый должен вычислить для своего варианта среднюю сквозную задержку, 95-й

перцентиль задержки, максимальную задержку и среднюю загрузку Fog-узлов (время ожидания в очереди) – как в исходной конфигурации, так и при всех изменениях. Эти метрики используются последовательно во всех трёх экспериментах. Индивидуальный подход таким образом заключается в разных входных параметрах (соотношении количества устройств), а сравнение результатов между студентами покажет, в каких конфигурациях система работает лучше или хуже. Каждый студент в своём отчёте оперирует именно своими числовыми результатами, но интерпретация ведётся на языке единых метрик, понятных преподавателю и другим студентам (задержки, процентили, нагрузки), что облегчает совместную работу над проектом, защиту и обсуждение выводов.

Таким образом, каждая лабораторная работа помогает понять свою часть проекта: от фундаментальных принципов (ЛР 3.1) через архитектурное планирование (ЛР 3.2) к оптимизационным настройкам (ЛР 3.3).

Лабораторные работы 3.1-4 дают комплект навыков для самостоятельного выполнения итогового проекта по моделированию IoT-системы.

На итоговой лекции в конце семестра поясняется, как перейти от лабораторных работ к собственному проекту.

Раздел 3: Как собрать свою систему (пошаговый план)

При выполнении собственного проекта по разработке и исследованию модели IoT системы, как и при разработке любого проекта, рекомендуется придерживаться заданной последовательности шагов. Ниже представлены этапы разработки – от идеи до итогового отчёта – с указанием их сути и связи с проделанными лабораторными работами:

1. Формулировка цели и постановка задачи.

Определите, что вы хотите смоделировать и зачем. Иными словами, сформулируйте прикладной сценарий IoT-системы и критерии её работы. Например: *«Умный дом для студента должен автоматически регулировать климат и освещение в комнате для комфорта и энергосбережения»* или *«IoT-мониторинг парковки должен отслеживать занятость мест в реальном времени»*. Чётко пропишите: какие данные собираются (температура, движение, и т.д.), какие устройства участвуют, какие показатели производительности важны (например, задержка оповещения не более 1 секунды, ограничение на потери данных и т.д.).

Постановка задачи – это фундамент проекта. Она задаёт требования, от которых будут зависеть решения на следующих шагах. *(Связь с лабораторными: ЛР 3.1 начиналась с описания сценария "умный дом", ЛР 3.2 знакомила с типами параметров системы, например, видами задержек и способами оценки параметров системы, ЛР 3.3 показывала, как оптимизировать систему по ее параметрам, то есть, за счет чего можно снизить задержки в реагировании системы и обеспечить ее устойчивую работу. В своём проекте вы делаете то же для собственной темы.)*

2. Выбор архитектуры и дизайн системы.

Спроектируйте архитектуру IoT-решения, исходя из поставленной цели. Решите, какие уровни нужны: будут ли использоваться только Edge-устройства

и Cloud, или требуется промежуточный Fog-слой? Сколько и каких устройств (вычислительных узлов) будет на каждом уровне? Как эти узлы будут связаны между собой?

3. Изобразите схему: датчики/источники данных, каналы связи, узлы обработки, конечные потребители данных. Определите, где будет происходить какая обработка: что делается на краю (например, предварительная фильтрация), что на туманных узлах (агрегирование, локальные расчёты), что в облаке (глобальная аналитика, хранение).

Выбор архитектуры включает и решение по протоколам: например, использовать MQTT для передачи от датчиков до Fog, HTTP(S) для отправки агрегированных данных в облачный сервер, и т.д.

Также спланируйте сеть: локальная сеть для Edge↔Fog (Wi-Fi, BLE?), интернет-соединение Fog↔Cloud (через 4G/5G или Ethernet) – от этого зависят задержки и надёжность.

Рекомендуется опираться на референсные архитектуры и примеры: шаблон “Edge–Fog–Cloud” подходит для большинства случаев (мы применяли его в ЛР 3.2 и 3.3).

На этом этапе полезно нарисовать две диаграммы: логическую (функциональную) – кто что делает и как данные текут; и техническую – какие устройства и протоколы задействованы[28][29]. *(Связь с лабораторными: в ЛР 3.1 вы составляли такую схему для “умного дома”, разделив роли по уровням. Теперь проделайте то же для своей задачи.)*

4. Настройка симуляции (подготовка модели).

После того как структура системы определена, переходите к её математической, а затем к программной модели.

Для программной модели выберите способ симуляции: вы можете использовать готовые инструменты (например, iFogSim, YAFS) или писать свой скрипт/программу, как это делалось в лабораторных работах (на Python, Java или

любом удобном языке). Второй подход часто проще для небольшой учебной задачи: можно адаптировать предоставленные в курсе файлы. Например, взять код из ЛР 3.1–3.3 (проект *STUDY_FOG*) и постепенно модифицировать под свою архитектуру.

Определите параметры модели: задержки в передачи сигнала на каждом типе связи (пинг между устройствами, задержка обработки на каждом узле – можно использовать разумные случайные диапазоны, как в лабораторных работх: датчик 20–60 мс, Fog 30–80 мс, сеть интернет 50–100 мс и т.п.), частоты генерации данных (например, датчик шлет показания раз в секунду), объём данных (байты, но можно абстрагировать как “задачи”). Если у вас разные типы устройств, заложите их особенности – как в лабораторных работах было сделано для мобильных (больший разброс задержек). Не забудьте про очереди: если узел может получать задачи быстрее, чем обрабатывать, введите параметр ёмкости буфера и логику отказа или ожидания при переполнении.

На этом этапе вы фактически программируете свою схему системы, то есть, вы создаёте классы/функции для имитации устройств и их взаимодействия.

Убедитесь, что ваша модель корректно отражает архитектуру системы, которую вы разработали ранее. *(Связь с лабораторными: ЛР 3.1 дала вам структурную схему системы, ЛР 3.2, ЛР 3.3 – шаблон масштабируемой модели, которые вы можете использовать как заготовку для ваше проекта).*

5. Запуск экспериментов.

С подготовленной моделью проведите серию экспериментов, чтобы получить результаты для анализа.

Во-первых, выполните базовый прогон: запустите моделирование (симуляцию) с начальными настройками и выберите наиболее важные метрики, по которым вы будете оценивать вашу систему (например, средняя сквозная задержка, число сгенерированных задач, процент потерянных задач, средняя

загрузка очередей). Убедитесь, что модель работает и выдаёт адекватные результаты.

Далее, спланируйте что нужно сравнивать: как и в лабораторных работах, вам следует продемонстрировать влияние тех или иных факторов и метрик на качество работы системы.

Примеры экспериментов: изменить число устройств или частоту генерации данных (т.е. нагрузку на систему) – и посмотреть, как растут задержки; улучшить или ухудшить возможности Fog-уровня (например, ускорить там обработку, как в ЛР 3.1, или совсем убрать Fog для сравнения с чистым облаком) – и оценить разницу в производительности; варьировать размер очередей или интервал отправки данных с краёв (аналогично ЛР 3.2 и 3.3) – чтобы найти оптимальные параметры. Каждый такой эксперимент – это отдельный запуск симуляции с новым набором параметров. Соберите достаточную статистику: возможно, имеет смысл запустить несколько раз и усреднить результаты (заложив *random seed* для воспроизводимости).

Важно: фиксируйте не только средние значения, но и распределения/перцентили, как делалось в ЛР 3.2, 3.3, чтобы ваши выводы были более обобщёнными. *(Связь с лабораторными: набор экспериментов в вашем проекте может во многом повторять то, что вы делали в ЛР 3.3–3.3, только для вашего конкретного проекта. Подумайте, какие вопросы и метрики для вашей системы являются ключевыми – и моделируйте варианты.)*

6. Анализ результатов.

После получения данных симуляции наступает этап интерпретации (объяснения полученных вами результатов).

Постройте графики и диаграммы для наглядности. Например, график изменения задержки при росте числа устройств (как в ЛР 3.2), график заполнения буфера во времени (как в ЛР 3.2,. 3.3), сравнение разных типов устройств или

конфигураций (как в Л.Р 33) с помощью разной визуализации результатов, например, столбчатые диаграммы или несколько кривых на одном графике.

Выявите метрики и случаи, где система справляется, а где возникают проблемы.

Анализируя результаты, отвечайте на изначальные вопросы проекта: удалось ли достичь требуемого времени отклика? сколько устройств выдерживается без деградации? где узкое место – сеть, Fog или облако?

Подтвердите или опровергните гипотезы, которые вы выдвигали в начале проектирования вашей системы.

Возможно, по результатам моделирования вы захотите вернуться и внести изменения в архитектуру, модель или в параметры вашей системы (это нормально – модель позволяет быстро итерационно улучшать дизайн системы).

Особое внимание уделите дополнительным метрикам качества: вариативности задержек (джиттер), проценту потерянных сообщений, использованию ресурсов. Например, высокий 95-й перцентиль при нормальном среднем – сигнал, что в редкие моменты очередь сильно растёт (возможно, при этом ваша система нуждается в настройке).

Привяжите ваши выводы к причинно-следственным связям. Например, “увеличение интервала опроса датчиков привело к разгрузке сети, что видно по снижению потерь пакетов на 20%”. Такой стиль анализа демонстрирует понимание процессов в системе. *(Связь с лабораторными: в отчётах по ЛР 3.2–3.3 вы уже рассчитывали расширенный набор метрик и делали выводы. Теперь вы делаете полноценный анализ для своей системы.)*

7. Оформление отчёта.

Завершающий шаг – представить проделанную работу в форме структурированного отчёта или презентации. Для курсового проекта или ВКР это особенно важно.

Отчёт обычно включает:

Введение: краткое описание предметной области и цели проекта. Почему эта задача актуальна, какие решения существуют, что сделано в работе.

Описание системы и архитектуры: тут пригодятся ваши диаграммы из шага 2. Опишите словесно архитектуру (уровни, компоненты, их функции) и покажите схему. Укажите, какие предположения и упрощения заложены (например, “несколько устройств объединены в один типовой узел из-за ограничений симуляции”).

Методика моделирования: опишите выбранный подход к симуляции. Чем вы пользовались (напр., “смоделировано собственным скриптом Python, логика следующая…” или “использован iFogSim, конфигурация такая-то”). Перечислите основные параметры модели: диапазоны задержек, частоты, объёмы данных, размеры очередей и пр. Если уместно, приведите небольшие фрагменты кода или псевдокода, особенно если вы реализовали что-то нетривиальное (например, особый алгоритм распределения задач).

Эксперименты и результаты: структурируйте эту часть на подпункты по каждому эксперименту. Опишите постановку: что варьировали, зачем. Затем представьте графики/таблицы результатов и рядом пишите наблюдения. Не перегружайте графики – лучше несколько отдельных с понятными подписями, чем один “супер-график”. Все рисунки и таблицы должны быть пронумерованы и иметь название. В тексте обязательно дайте ссылки на рисунки (например, “как видно из рис. 3, при 2000 устройств задержка превысила 5 с”).

Выводы: обобщите самые результаты и закономерности. Для этого ввернитесь к цели проектирования вашей системы из введения – достигнута ли цель? Например: “Моделирование подтвердило работоспособность концепции умного дома: средняя задержка оповещения ~0.5 с, что укладывается в требование 1 с. Однако при увеличении числа устройств свыше 100 наблюдается нелинейный рост задержек – потребуется масштабирование Fog-уровня.” Здесь

же можно указать направления развития или улучшения модели системы (что бы вы сделали, будь больше времени).

Ссылки: если вы опирались на какие-либо источники (стандарты, статьи, чужие проекты), оформите список литературы. В учебном проекте он может быть небольшим, но, например, сослаться на OpenFog Architecture или документацию MQTT будет полезно.

(Примечание: формат отчёта для ВКР обычно жёстче, чем для курсовой работы, в ВКР входят такие разделы как "Обзор литературы", "Технико-экономическое обоснование" и пр. В данном пособии описано только самые важные разделы технической части при проектировании).

Связь с курсовой и выпускной квалификационной работой

Тематика IoT-технологий и туманных вычислений прекрасно подходит для дипломных и курсовых работ по направлению "Перспективные информационные технологии". Выполнив описанные лабораторные и свой учебный проект, вы фактически получаете задел для ВКР.

Как использовать результаты лабораторных работ? Во-первых, можете взять за основу собственный сценарий (например, тот же умный дом или город) и углубить его в рамках диплома – добавить реальные данные, больше экспериментов, рассмотреть также вопросы безопасности или экономической эффективности. Во-вторых, используйте наработки: коды симуляций, диаграммы архитектуры, методику экспериментов – все это ляжет в разделы диплома. Лабораторные работы уже дали вам шаблон отчёта и содержание, соответствующее научному подходу (постановка задачи, модель, эксперимент, анализ результатов). В курсовой/ВКР нужно сохранить эту логику, расширив масштаб и проработанность. Обратите внимание на оформление: рисунки из отчёта должны быть отредактированы в соответствии с требованиями стандарта (подписи, нумерация, ссылка в тексте), текст – выдержан в едином стиле.

Возможно, стоит провести реальный эксперимент с парой устройств для валидации симуляции – об этом можно написать в главе с результатами. Также увяжите свою работу с общей теорией: во введении диплома дайте определения IoT и Fog (из лекций и учебников), опишите кратко современное состояние (можно сослаться на стандарты или прогнозы по количеству IoT-устройств). В заключении – подчеркните, как результаты симуляции могут быть применены или внедрены на практике. Таким образом, ваш учебный проект по симуляции станет центральной частью итоговой квалификационной работы, демонстрируя и технические навыки моделирования, и понимание перспективных технологий.

Пример: «Умный дом студента» – от идеи к реализации

Рассмотрим гипотетический пример, иллюстрирующий применение всех шагов.

Сценарий: студент живёт в общежитии/квартире и хочет автоматизировать контроль климата и электроники, чтобы экономить электроэнергию и создать комфортную среду для учёбы и отдыха.

Цель проекта: создать IoT-систему “умный дом” для одной комнаты, которая будет мониторить условия (температура, освещённость, качество воздуха), присутствие человека, и автоматизированно управлять отоплением, кондиционером, освещением, розетками. Система должна реагировать быстро (в течение секунд) и работать даже при перебоях интернета.

Архитектура: на краю – несколько датчиков: температурный датчик и датчик влажности воздуха; датчик освещённости; датчик движения или присутствия (чтобы понять, есть ли человек в комнате); умные розетки или реле на электроприборах. Эти устройства связаны с локальным хабом по Wi-Fi или Bluetooth. Fog-узел: в роли локального сервера выступает, например, мини-компьютер Raspberry Pi (или просто симулируется как “шлюз”), установленный в комнате. Он собирает данные с датчиков, выполняет локальную логику (например, если человек ушёл – выключить свет и отопление до минимального).

Fog-узел хранит кратковременную историю, может выполнять простую аналитику (например, вычислять среднее значение температуры за час, следить, чтобы не превышались пороги). Cloud: удаленный сервер (или облачный IoT-платформенный сервис, например, Blynk, ThingsBoard, либо собственный сервер) – получает агрегированные данные раз в какое-то время, чтобы сохранять длительную историю, выполнять более сложный анализ (например, строить графики потребления энергии, применять ML для прогнозирования) и предоставлять приложение для пользователя. Пользователь (студент) взаимодействует через смартфон – получает уведомления (например: “в комнате никого, выключаю кондиционер”), может удалённо просматривать параметры и управлять устройствами. Коммуникации: датчики → Fog-хаб: по MQTT (легковесно и локально); Fog → Cloud: по HTTPS (отправка пакета данных каждую минуту) или также MQTT через интернет. Cloud → смартфон: через Web dashboard или мобильное приложение (может быть push-уведомления).

Симуляция: прежде чем собирать железо, студент решает промоделировать систему. Он закладывает в модель: 5–6 edge-устройств (разные датчики и актуаторы), 1 fog-узел, 1 облако, 1 пользователь. Настраивает задержки: датчики передают по Wi-Fi с задержкой 10–30 мс, Fog-обработка занимает 50–100 мс (имитируя вычисления), отправка в облако – через интернет с задержкой, допустим, 100 мс и вариацией (может быть больше, если сеть загружена). Закладывает логику: датчики шлют данные каждые 5 сек, Fog-узел сразу реагирует на некоторые события (если движение = False, человек отсутствует, тогда через 60 сек выключить устройства). Он моделирует сутки работы: по сценарию утром человек уходит на пары – датчики движения перестают срабатывать, система гасит свет; вечером возвращается – датчики активны, система включает свет и климат.

Эксперименты: студент прогоняет симуляцию с разными условиями. Например: *Эксперимент А:* идеальные условия сети против плохого Wi-Fi –

увеличивает в модели сетевую задержку и процент потери пакетов, смотрит, влияет ли это на вовремя ли выключится свет (оказывается, Fog-узел и так работает локально, так что критические функции не зависят от облака, а облако может получать данные с опозданием – и это приемлемо). *Эксперимент B:* сравнивает, что если *нет Fog-узла* (все датчики шлют сразу в облако, а управление идёт обратно из облака) – симулирует такой вариант: задержка действий увеличивается, при потере интернета вовсе не работает. Это убеждает в нужности Fog. *Эксперимент C:* масштабирует число датчиков (представив, что в доме 3 комнаты с такими же системами, итого 3× больше устройств) – проверяет, справится ли один Raspberry Pi-hub или нужно по одному на комнату.

Анализ: на основе симуляции студент получает количественные подтверждения: например, время реакции на отсутствие человека – ~60 секунд (задержка тайм-аута), средняя задержка доставки данных в облако – 0.2 с (что не критично для мониторинга), потери – 5% пакетов при плохом Wi-Fi (приемлемо, т.к. данные не критичны), нагрузка Fog – CPU загрузка 30% при 15 устройствах (т.е. один Pi может обслужить до ~50 устройств). Эти результаты он представляет в графиках и таблицах.

Отчёт: студент оформляет полученное как мини-отчёт или презентацию. В ней есть схема “умного дома студента” с Edge/Fog/Cloud, есть график “событие -> реакция” во времени, сравнение сценариев (с Fog и без Fog), и вывод: локальный Fog-контроллер заметно повышает автономность и быстродействие системы, а симуляция помогла подобрать параметры (например, решить, что интервал контроля присутствия 60 с оптимален). Дальше, этот проект может быть расширен до дипломного – добавлением реальной реализации (построить систему с Raspberry Pi и датчиками и сравнить с моделью).

Этот пример демонстрирует подход к построению IoT-проекта с нуля. Аналогично вы можете взять любую идею (умный сад, трекинг транспорта,

безопасность помещения и т.п.) и, используя знания IoT и Fog, разработать её сначала в симуляции, а затем и в реальности.