УДК 004.75

В. Гайдамако
Институт машиноведения, автоматики и геомеханики Национальной академии наук, Бишкек

МОДЕЛИРОВАНИЕ СЕТЕВОЙ ИНФРАСТРУКТУРЫ ОБЛАЧНОЙ ИНФОРМАЦИОННО-ИЗМЕРИТЕЛЬНОЙ СИСТЕМЫ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ NS3

В статье представлен пример моделирования структуры облачной информационно-измерительной системы с помощью симулятора NS3. Библиотека NS3 позволяет моделировать различные топологии сетей, протоколы передачи данных, возможно проследить прохождение по сети отдельного пакета. Разработано множество сторонних приложений для анализа трассировок и визуализации топологии. NS3 можно использовать совместно с другими библиотеками.

Ключевые слова: облачные информационно-измерительные системы, Центр Обработки Данных (ЦОД), моделирование, Беспроводная Сенсорная Сеть (БСС), NS3, Simgrid, проблема масштабирования

Введение. Современные распределенные системы обработки информации становятся все более масштабными и сложными, их изучение в реальности, на рабочих системах или стендах слишком дорого или вообще невозможно, поэтому актуальны вопросы моделирования таких систем. Модели позволяют снять риск и стоимость экспериментов, «проиграть» различные сценарии событий – всплески нагрузки, отказы, деградацию сети без риска и с очень небольшими затратами. Моделирование позволяет сравнить различные топологии, схемы балансировки, кэширование, репликацию, типы каналов, найти «бутылочное горлышко», применить горизонтальное/вертикальное масштабирование и оценить результаты. Моделирование позволяет определить начальную инфраструктуру приложения в облаке, определить, сколько серверов/ядер/пропускной полосы каналов связи нужно, чтобы выдержать заданное количество запросов на обслуживания с задержкой, оговоренной в соглашении об уровне обслуживания, гарантировать не только средние, но и верхние оценки задержек/очередей. Моделирование проявляет неочевидные эффекты, которые трудно предусмотреть и увидеть на работающей системе – блокировку начала очереди (head-of-line blocking) [1], инверсию приоритетов, конфликт блокировок (lock contention), рост очередей после узких участков. Планируемые изменения в аппаратном и программном обеспечении, смена политики сервисного брокера или алгоритма балансировки нагрузки, новая схема репликации базы данных – все это можно проверить на модели.

Библиотека моделирования NS3. Для компьютерного моделирования распределенных и облачных систем разработано множество библиотек и фреймворков, одной из самых популярных является библиотека моделирования сетей NS3[2]. NS3 написан на C++, при установке пользователь получает исходный код NS3, компилирует его в разделяемые (или статические) библиотеки, которые подключает к своим программам main(). В main() выполняется настройка конкретного сценария симуляции, осуществляется запуск и останов симуляции. Библиотеки NS3 можно комбинировать с другими внешними программными библиотеками, в том числе с библиотеками Simgrid [3,4]. Пользователям следует быть готовыми к работе в командной строке, NS3 в основном используется в системах Linux и macOS, существует поддержка систем BSD, а также фреймворков Windows, способных собирать код Linux, например, Windows Subsystem for Linux или Cygwin. Разработчик планирует в будущем реализовать поддержку Visual Studio для Windows.

Основные термины, применяемые при описании сетевой инфраструктуры, в NS3 реализуются как классы Node (узел), Application (приложение), Channel (канал), NetDevice (сетевое устройство). Узлы можно объединять в контейнеры. Каналы могут моделировать как простые кабельные соединения, так и большие Ethernet свичи или трехмерное пространство беспроводных сетей. Например, класс CsmaChannel моделирует подключение к

совместно используемому средству коммуникации Carrier Sense Multiple Access(CSMA) через кабель, удобно для моделирования сети Ethernet, класс PointToPointChannel – полнодуплексное соединение двух устройств, класс WifiChannel моделирует распространение пакетов, потерю сигнала, задержку в беспроводных сетях. Сетевое устройство используется для подключения узла к сети, это сетевая карта – Network Interface Card (NIC), абстракция сетевого устройства охватывает как программный драйвер, так и моделируемое оборудование. Как и в реальном компьютере, узел может быть подключен к нескольким каналам через несколько сетевых устройств. Специализированные устройства –- CsmaNetDevice, PointToPointNetDevice, WifiNetDevice. Для удобства управления элементами сети используются вспомогательные объекты (хелперы), которые упрощают настройки, требующие множества операций [5].

Параметры модели. Построим модель для беспроводной сети из n датчиков с топологией: БСС →координатор/шлюз/AP (Access Point, точка доступа) → Интернет → шлюз облака → облачная ЛВС (сервер данных + веб-сервер), где координатор периодически собирает данные с датчиков и отправляет их агрегированным пакетом в облако (см рис.1), при этом:

- датчики передают измерения в координатор-шлюз с периодом опроса pollPeriod (это имитация опроса; при желании можно заменить на «истинный pull» через UdpEcho). Датчики соединяются в сеть с помощью точки доступа Wi-Fi 802.11b/g/n, модуль для этой сети является встроенным;
- координатор принимает данные от датчиков локально, накапливает принятые пакеты, и раз в период агрегации aggPeriod отправляет один агрегированный UDP-пакет в облачный сервер данных через соединение со шлюзом облака;
- канал «Интернет» моделируется как PointToPoint со своей задержкой/полосой;
- в облаке есть сервер данных и веб-сервер на общей CSMA-шине.

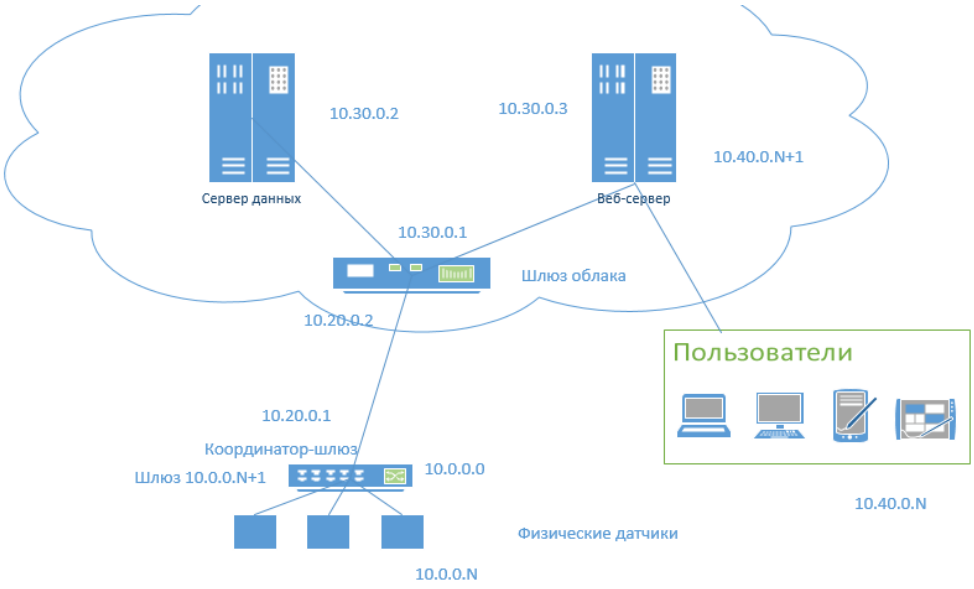

Рисунок 1 – Моделируемая система – датчики-шлюз облака–серверы–пользователи

При запуске симуляции количество датчиков в БСС и другие параметры задаются по умолчанию, их можно менять в командной строке:

```
static uint32_t g_nSensors   = 6;      // число датчиков
static double   g_simTime    = 60.0;   // время симуляции, с
static double  g_pollPeriod = 2.0; // период «опроса/передачи» датчиков,с
```

```
static double g_aggPeriod  = 5.0;    // период агрегации на шлюзе (с)
static uint32_t g_pktSize    = 256;    // размер пакета датчика (байт)
static uint16_t g_localPort  = 5000;   // порт на шлюзе для датчиков
static uint16_t g_cloudPort  = 6000;   // порт сервера данных в облаке

// «Интернет»-канал
static std::string g_inetDelay = "20ms";
static std::string g_inetRate  = "10Mbps";

// Wi-Fi стандарт
static WifiStandard g_wifiStd = WIFI_STANDARD_80211g;
```

Создание узлов: создаются контейнеры с датчиками, точкой доступа, одним шлюзом облака и одним сервером. Далее можно будет легко изменять количество узлов в контейнерах:

```
NodeContainer staNodes;
staNodes.Create(g_nSensors); // датчики
```

Можно создать контейнер, и узлы в нем, можно создать отдельные узлы и позже поместить их в контейнер. Далее создаются отдельные узлы.

```
Ptr<Node> apGwNode = CreateObject<Node>();// координатор БСС
Ptr<Node> cloudGwNode = CreateObject<Node>();  // шлюз облака
Ptr<Node> cloudDataNode = CreateObject<Node>();//сервер данных
Ptr<Node>  cloudWebNode  = CreateObject<Node>();// веб-сервер в облаке
```

Настройка беспроводной сети: для беспроводной сети обязательно указываются параметры мобильности и расположение узлов (датчиков), для этого используется вспомогательный объект MobilityHelper.

```
MobilityHelper mobility;
mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                              "MinX", DoubleValue(0.0),
                              "MinY", DoubleValue(0.0),
                              "DeltaX", DoubleValue(5.0),
                              "DeltaY", DoubleValue(10.0),
                              "GridWidth", UintegerValue(3),
                              "LayoutType", StringValue("RowFirst"));
```

Метод SetPositionAllocator определяет, где элементы БСС находятся в начале симуляции, при вызове метода MobilityHelper Install(). В данном случае датчики размещаются в двумерной решетке. Задаются начальные координаты X и Y первой точки сетки, горизонтальное и вертикальное расстояние между узлами., количество узлов в одном ряду и порядок заполнения – ряд или столбец заполняются первыми. То есть узлы БСС будут расположены следующим образом:

Ряд 0:	(0,0)	(5,0)	(10,0)
Ряд 1:	(0,10)	(5,10)	(10,10)
Ряд 2:	(0,20)	(5,20)	(10,20)

Далее необходимо определить мобильность узлов во время симуляции, пока будем считать, что узлы БСС и точка доступа неподвижны

```
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(staNodes);

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(aGwpNode);
```

Для случаев, когда датчики перемещаются, в NS3 существует множество встроенных моделей мобильности, которые можно использовать с методом mobility.SetMobilityModel, например, узлы могут быть статичными, двигаться по прямой, перемещаться случайно или по определенным траекториям, имитируя движение транспорта по улицам [5].

Каналы связи: далее нужно создать каналы связи и определить физический уровень. Для этого есть вспомогательные объекты канала и физического уровня (PHY). Класс YansWifiChannelHelper упрощает создание беспроводного канала и настройку модели распространения сигнала (PHY Channel) на основе модели Yans (Yet Another Network Simulator) [6], используется для определения того, как радиосигналы распространяются между узлами, включая затухание (потерю мощности сигнала) и задержку распространения.

```
YansWifiChannelHelper chan;
YansWifiPhyHelper phy;

channel.AddPropagationLoss("ns3::FriisPropagationLossModel");
channel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
phy.SetChannel(channel.Create());
```

После описания физического уровня настраивается MAC (Media Access Control, управление доступом к среде) уровень для беспроводных сетевых устройств стандарта 802.11, за это отвечает WifiMacHelper. Уровень MAC определяет правила доступа к общему беспроводному каналу, включая режим работы сети (Ad-Hoc, Инфраструктура с точкой доступа) и правила совместного использования канала (CSMA/CA).

```
WifiMacHelper mac;
Ssid ssid = Ssid("wifi-network");
```

Для дальнейшей настройки используется метод SetType(), который в качестве первого аргумента принимает строку с полным именем класса NS3,_реализующего определенную логику MAC, и пары атрибут/значение для тонкой настройки этого класса (необязательные аргументы). Тип Сеть в режиме инфраструктуры (Infrastructure Mode) описывает сеть, в которой есть точка доступа (AP) и одна или несколько Станций (STA), они настраиваются по-разному, но идентификатор сети – SSID должен быть одинаковым. Точка доступа:

```
mac.SetType("ns3::ApWifiMac", "Ssid", SsidValue(ssid));
```

Станции (датчики):

```
mac.SetType("ns3::StaWifiMac",
            "Ssid", SsidValue(ssid),
            "ActiveProbing", BooleanValue(false));
```

Возможны другие типы сетей – сеть Ad-Hoc (Peer-to-Peer Mode), когда узлы равны и общаются напрямую друг с другом без центральной точки доступа, тип AdhocWifiMac, ячеистая сеть (Mesh Mode), тип MeshWifiMac [5, 7].

Далее обращаемся к вспомогательному объекту WifiHelper, который действует как центральный координатор для настройки беспроводного интерфейса и используется для упрощенной настройки и установки стека протоколов Wi-Fi (IEEE 802.11) на узлы. Он объединяет конфигурацию физического уровня (PHY) и уровня управления доступом к среде (MAC), а затем устанавливает их на указанные узлы.

```
WifiHelper wifi;
wifi.SetStandard(WIFI_STANDARD_80211g);
```

В БСС часто используется стандарт Zigbee, специально созданный для приборов с низким энергопотреблением. Симулятор NS3 не имеет прямого встроенного модуля с названием "Zigbee", но он полностью поддерживает базовый стандарт, на котором основана технология Zigbee – стандарт IEEE 802.15.4 [8]. Для этого используется модуль lr-wpan (Low-Rate Wireless Personal Area Network), который реализует физический уровень (PHY) и уровень управления доступом к среде (MAC) стандарта IEEE 802.15.4, в котором реализованы спецификации стандартов IEEE 802.15.4-2003, 802.15.4-2006 и 802.15.4-2011 [8]. Метод

Install принимает настроенные хелперы PHY и MAC, контейнер узлов и создает реальные сетевые устройства (`NetDevice`) на каждом узле.

```
NetDeviceContainer apDevice = wifi.Install(phy, mac, wifiApNode);
NetDeviceContainer staDevices = wifi.Install(phy, mac, wifiStaNodes);
```

Настройка интернет-канала. Координатор БСС связывается со шлюзом облака через интернет-канал, который будем моделировать как соединение точка-точка (point-to-point) – два устройства связаны каналом с фиксированной скоростью и задержкой. В принципе, это любое stateful соединение, соединение с сохранением состояния, то есть с образованием канала. Для настройки используется PointToPointHelper.

```
PointToPointHelper p2p;
```

Метод SetDeviceAttribute используется для задания пропускной способности канала:

```
p2p.SetDeviceAttribute("DataRate", StringValue(g_inetRate));
```

Настройки канала – задержка для всех пакетов:

```
p2p.SetChannelAttribute("Delay", StringValue(g_inetDelay));
```

Тип очереди и размер буфера – метод SetQueue:

```
p2p.SetQueue("ns3::DropTailQueue<Packet>",
        "MaxSize", StringValue("50p")); // 50 пакетов
```

Можно менять и тип очереди, или дисциплину управления трафиком, влияющую на качество обслуживания и маршрутизацию пакетов. Это может быть простая очередь, очередь с контролируемой задержкой, со случайным ранним обнаружением, адаптивная очередь, очередь маркерного ведра, мультиочередь с приоритетами [5] (должны быть подключены нужные модули). Метод Instal привязывает всё это к узлам и возвращает NetDeviceContainer:

```
NetDeviceContainer devs = p2p.Install(aGwNodeContainer);
```

Метод Install можно вызывать несколько раз для разных пар узлов, каждый раз создаётся новый канал. Так же моделируется соединение пользователей с веб-сервером.

Настройка локальной сети ЦОД. Предположим, что в ЦОД используются сети типа Ethernet. Реальный Ethernet использует схему CSMA/CD (Carrier Sense Multiple Access with Collision Avoidance/Detection, множественный доступ с контролем несущей и обнаружением коллизий) с экспоненциально возрастающей задержкой передачи при конкуренции за общую среду передачи. Устройство CSMA и канал NS3 моделируют подмножество этой схемы. Класс CsmaHelper используется для симуляции проводных сетей и позволяет создать канал (CsmaChannel), моделирующий общую физическую среду (например, Ethernet-кабель или коммутатор), установить сетевые устройства (CsmaNetDevice) на узлы и настроить скорость передачи данных (data rate) и время задержки распространения сигнала (delay) для этого канала. Создадим вспомогательный объект CSMA и контейнер для серверов в облаке – сервера данных и веб-сервера.

```
CsmaHelper csma;
NodeContainer cloudLan;
cloudLan.Add(cloudGwNode);
cloudLan.Add(cloudDataNode);
cloudLan.Add(cloudWebNode);
NetDeviceContainer cloudCsma = csma.Install(cloudLan);
```

Установка скорости канала 100 Мбит/с:

```
csmaHelper.SetChannelAttribute("DataRate", StringValue("100Mbps"));
```

Установка задержки распространения 2 микросекунды:

```
csmaHelper.SetChannelAttribute("Delay", StringValue("2us"));
```

Установка сетевых устройств на узлы:

```
NetDeviceContainer csmaDevices = csma.Install(csmaNodes);
```

Стек протоколов и настройка адресов. Для того, чтобы узлы могли использовать стандартные протоколы связи (IP-адресация, маршрутизация, TCP или UDP), им необходимо "установить" программное обеспечение стека протоколов. Класс InternetStackHelper используется для установки полного стека протоколов интернет (TCP/IP) на узлы симуляции, добавляя на каждый узел необходимые компоненты – IPv4 или IPv6 (поддержка протокола межсетевого уровня), TCP/UDP (поддержка транспортного уровня), базовые механизмы маршрутизации.

```
InternetStackHelper stack;
stack.Install(staNodes);
stack.Install(apGwNode);
stack.Install(cloudGwNode);
stack.Install(cloudDataNode);
stack.Install(cloudWebNode);
```

Для настройки IP адресов используем вспомогательный объект Ipv4AddressHelper:

```
Ipv4AddressHelper addr;
```

Метод SetBase конфигурирует подсеть, из которой будут выделяться адреса, и маску. Метод Assign принимает контейнер сетевых устройств (NetDeviceContainer) и автоматически назначает следующий доступный IP-адрес из настроенной подсети каждому устройству в контейнере. Например, первому устройству будет назначен 10.1.1.1, второму 10.1.1.2, и так далее. Возвращает контейнер интерфейсов (Ipv4InterfaceContainer), содержащий информацию о назначенных адресах. Для БСС:

```
addr.SetBase("10.10.0.0", "255.255.255.0");
Ipv4InterfaceContainer staIfs = addr.Assign(staDevs);
Ipv4InterfaceContainer apIf   = addr.Assign(apDev);
```

Для Интернет соединения со шлюзом облака (p2p):

```
addr.SetBase("10.20.0.0", "255.255.255.0");
Ipv4InterfaceContainer inetIfs = addr.Assign(p2pDevs);
// inetIfs.GetAddress(0) — адрес AP/GW; inetIfs.GetAddress(1) — адрес
   CloudGW
```

Для подсети облака (CSMA):

```
addr.SetBase("10.30.0.0", "255.255.255.0");
Ipv4InterfaceContainer cloudIfs = addr.Assign(cloudCsma);
// индексы: 0:CloudGW, 1:Data, 2:Web

Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```

Если сеть разделяется на подсети, узлам необходимо «знать» маршруты к другим подсетям. Класс IPv4GlobalRoutingHelper епрощает настройку и использование статической маршрутизации для сетей, использующих протокол IPv4, когда не нужно моделировать динамические протоколы маршрутизации. Он настраивает таблицу маршрутизации на каждом узле, сканирует топологию сети и автоматически создает необходимые статические маршруты для обеспечения сквозного IP-соединения между всеми узлами. Инструктируем стек IP-протоколов использовать механизм глобальной маршрутизации по умолчанию:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```

Настройка приложений. Для моделирования сбора данных будем использовать сервер UDP, для его быстрой настройки используется класс UdpServerHelper. Приложение сервера просто ожидает входящие UDP-пакеты на указанном порту. При получении пакетов оно может измерять статистику – количество полученных байтов и пакетов, и при необходимости отправлять подтверждение отправителю (эхо). Настроим приложение на шлюзе/AP, это UDP-сервер, который принимает данные от датчиков, задаем номер порта и создаем объект севера:

```
  UdpServerHelper localSrv(g_localPort);
  ApplicationContainer localSrvApp = localSrv.Install(apGwNode);
  localSrvApp.Start(Seconds(0.5));
  localSrvApp.Stop(Seconds(g_simTime));
  g_localUdpServer = DynamicCast<UdpServer>(localSrvApp.Get(0));
```

Метод Install() принимает контейнер узлов (NodeContainer) и устанавливает приложение UDP-сервера на каждый узел в контейнере. Он возвращает контейнер приложений (ApplicationContainer).

Каждое приложение должно иметь определенное время начала и остановки активности. ApplicationContainer::Start задает это время для всех приложений контейнера. Метод принимает один параметр типа ns3::Time: Метод `nApplicationContainer::Stop` устанавливает единый момент времени, когда все приложения в контейнере должны прекратить свою активность.

Для датчиков настраиваем приложение UdpClient, генерирующее поток UDP-пакетов, каждый из которых содержит 32-битный порядковый номер и 64-битную метку времени. Это приложение полезно для тестирования сетевого соединения и измерения задержки и процента потерянных пакетов. UdpClientHelper имеет несколько конструкторов - UdpClientHelper(), UdpClientHelper(Address ip, uint16_t port), UdpClientHelper(const Address &addr). Для каждого датчика БСС создаем экземпляр приложения с указанием адреса и порта и устанавливаем атрибуты – "MaxPackets" (максимальное количество пакетов), "Interval" (интервал), "PacketSize" (размер пакета), устанавливаем его на узел и задаем время старта и останова. Датчики шлют данные в шлюз/AP раз в pollPeriod

```
  for (uint32_t i = 0; i < g_nSensors; ++i) {
    UdpClientHelper cli(apIf.GetAddress(0), g_localPort);
    cli.SetAttribute("MaxPackets", UintegerValue(0xffffffff)); // по сути
«бесконечно», ограничит Stop()
    cli.SetAttribute("Interval", TimeValue(Seconds(g_pollPeriod)));
    cli.SetAttribute("PacketSize", UintegerValue(g_pktSize));
    ApplicationContainer app = cli.Install(staNodes.Get(i));
    // небольшое рассинхронизированное начало, чтоб не все сразу
    double jitter = 0.2 * (i % 5);
    app.Start(Seconds(1.0 + jitter));
    app.Stop(Seconds(g_simTime));
  }
```

Приложение сервера данных в облаке

```
UdpServerHelper cloudDataSrv(g_cloudPort);
  ApplicationContainer                    cloudDataApp                    =
cloudDataSrv.Install(cloudDataNode);
  cloudDataApp.Start(Seconds(0.5));
  cloudDataApp.Stop(Seconds(g_simTime));
```

Веб-сервер — пока просто «заглушка» (можно повесить HTTP-трафик позже) Для демонстрации повесим UdpServer на порт 7000 (неиспользуемый)

```
  UdpServerHelper cloudWebSrv(7000);
  ApplicationContainer cloudWebApp = cloudWebSrv.Install(cloudWebNode);
  cloudWebApp.Start(Seconds(0.5));
  cloudWebApp.Stop(Seconds(g_simTime));
```

Сбор информации о событиях симуляции, трассировка. Система трассировки (tracing) в симуляторе NS3 позволяет собирать детальную информацию о событиях, происходящих во время симуляции для дальнейшего анализа. Вспомогательные объекты трассировки (Trace Helpers) генерируют стандартную трассировку, но можно подключать собственный код, изменяя формат генерируемых выходных данных или добавляя новые

источники трассировки без изменения ядра симулятора. Также можно модифицировать ядро симулятора, чтобы добавить новые источники и приемники трассировки.

Основой системы трассировки NS3 являются независимые источники и приемники. Источники трассировки – это сущности, которые могут сигнализировать о событиях, происходящих в процессе моделирования, и предоставлять доступ к интересующим базовым данным. Приёмники трассировки являются потребителями событий и данных от источника, это функции-обработчики (callbacks). ASCII-трассировка, или текстовая трассировка, включается через вспомогательный объект и текстовые файлы в формате tr.

```
  AsciiTraceHelper ascii;
  phy.EnableAsciiAll(ascii.CreateFileStream("wifi-gateway.tr"));
```

PCAP трассировка создает файлы «захвата пакетов» в формате .pcap (сокращение от packet capture). Файлы могут обрабатываться внешними программами, самая популярная – Wireshark (ранее Ethereal). Существует множество анализаторов трассировки трафика, использующих этот формат пакетов, например tcpdump. Для включения трассировки PCAP достаточно одной строки.

```
  phy.EnablePcap("wifi-gateway", apDevice.Get(0));
```

Кроме того, есть система логирования и вывод сообщений через поток на терминал или в файл. Пример вывода через систему логирования, команда `NS_LOG_UNCOND("[AGG] Send " << newly << " pkts, total " << bytes`, идет логирование каждого пакета на шлюзе БСС.

```
[AGG] gateway/AP: aggregate 12 pkts = 3072 bytes → send to cloud data server
[AGG] gateway/AP: aggregate 18 pkts = 4608 bytes → send to cloud data server
[AGG] gateway/AP: aggregate 12 pkts = 3072 bytes → send to cloud data server
```

Вспомогательный объект FlowMonitor собирает информацию по соединениям и суммирует ее после окончания симуляциии. До начала симуляции создается объект,

```
..FlowMonitorHelper flowmon;
  Ptr<FlowMonitor> monitor = flowmon.InstallAll();
```

После окончания симуляции подводим итоги по шлюзу БСС и серверу данных:

```
  Ptr<UdpServer>      dataSrvPtr      =      DynamicCast      <UdpServer>
(cloudDataApp.Get(0));
  std::cout << "\n=== SUMMARY ===\n";
  std::cout << "Local gateway/AP received from sensors (pkts): "
            << g_prevReceived << "\n";
  if (dataSrvPtr)
    std::cout << "Cloud data server received (pkts): "
              << dataSrvPtr->GetReceived() << "\n";
```

Для 6 датчиков, период опроса 2 с, период агрегации и пересылки 5 с, размер пакета данных измерений 256 байт:

```
Local gateway/AP received from sensors (pkts): 162
Cloud data server received (pkts): 11
```

По устройствам:

```
  monitor->CheckForLostPackets();
  auto stats = monitor->GetFlowStats();
  auto                              classifier                                =
DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
  for (const auto& kv : stats) {
    auto five = classifier->FindFlow(kv.first);
    const auto& s = kv.second;
```

```
    if (s.rxPackets == 0) continue;
    double avgDelay = s.delaySum.GetSeconds() / s.rxPackets;
    std::cout << "Flow " << kv.first << " "
              << five.sourceAddress << "->" << five.destinationAddress
              << " rx=" << s.rxPackets
              << " avgDelay=" << avgDelay << " s\n";
```

```
Flow 1 10.10.0.1->10.10.0.7 rx=30 avgDelay=0.00064683 s
Flow 2 10.10.0.6->10.10.0.7 rx=30 avgDelay=0.000573711 s
Flow 3 10.10.0.2->10.10.0.7 rx=30 avgDelay=0.000248417 s
Flow 4 10.10.0.3->10.10.0.7 rx=30 avgDelay=0.000371709 s
Flow 5 10.10.0.4->10.10.0.7 rx=30 avgDelay=0.000384077 s
Flow 6 10.10.0.5->10.10.0.7 rx=30 avgDelay=0.000238305 s
Flow 7 10.20.0.1->10.30.0.2 rx=1 avgDelay=0.0242098 s
Flow 8 10.20.0.1->10.30.0.2 rx=1 avgDelay=0.0237636 s
```

Удобно по ходу симуляции осуществлять вывод в файл, например, в формате csv.

Результаты симуляции в зависимости от количества датчиков. Для представленного примера посмотрим, как меняется время доставки пакетов на разных участках сети в зависимости от количества датчиков. Для этого была проведена серия симуляций с заданными значениями переменных, рассмотрим некоторые из них. Результаты для 10 датчиков, 5 пользователей, период опроса датчиков 1 с, период агрегации и пересылки – 5 с, 5 пользователей приведены в таблице 1.

Таблица 1 – Результаты симуляции для 10 датчиков

srcAddr	dstAddr	txPackets	rxPackets	lostPackets	avgDelay ms	minDelay ms	maxDelay ms
10.10.0.1	10.10.0.11	59	59	0	0.596761	0	10
10.10.0.10	10.10.0.11	59	59	0	0.563167	0	9
10.10.0.2	10.10.0.11	59	59	0	0.557905	0	7
10.10.0.3	10.10.0.11	59	59	0	0.570341	0	10
10.10.0.4	10.10.0.11	59	59	0	0.653017	0	11
10.10.0.5	10.10.0.11	59	59	0	0.614123	0	9
10.10.0.6	10.10.0.11	59	59	0	0.661744	0	11
10.10.0.7	10.10.0.11	59	59	0	0.490994	0	2
10.10.0.8	10.10.0.11	59	59	0	0.5995090	0	9
10.10.0.9	10.10.0.11	59	59	0	0.525717	0	7
10.20.0.1	10.30.0.2	5	4	2	30.13112	0	40
10.30.0.2	10.30.0.3	25	25	0	2.46817	2	14
10.30.0.3	10.30.0.2	25	25	0	2.34017	2	11
10.40.0.1	10.40.0.6	5	5	0	2.815724	1	11
10.40.0.2	10.40.0.6	5	5	0	2.015728	1	7
10.40.0.3	10.40.0.6	5	5	0	2.815724	1	11
10.40.0.4	10.40.0.6	5	5	0	1.415728	1	4
10.40.0.5	10.40.0.6	5	5	0	2.015728	1	7
10.40.0.6	10.40.0.1	5	5	0	2.4261280	1	9
10.40.0.6	10.40.0.2	5	5	0	2.226128	1	8
10.40.0.6	10.40.0.3	5	5	0	2.8261240	1	11
10.40.0.6	10.40.0.4	5	5	0	1.4261280	1	4
10.40.0.6	10.40.0.5	5	5	0	1.626128	1	5

Потеря пакетов (2 пакета) при передаче от шлюза-координатора на сервер данных, это узкое место, для его устранения в данном случае нужны соединения с лучшей полосой пропускания, при большем количестве пакетов можно применить горизонтальное масштабирование – увеличить количество серверов данных и каналов соединения. В БСС все передачи от датчиков на координатор выполнены. Если увеличить количество датчиков до 100, картина существенно меняется (Таблица 2), пакеты теряются уже при передаче от датчика на координатор, задержка при передаче на сервер данных становится неприемлемо высокой.

Таблица 2 – Результаты симуляции для 100 датчиков

srcAddr	dstAddr	txPackets	rxPackets	lostPackets	avgDelay ms	minDelay ms	maxDelay ms
10.10.0.1	10.10.0.101	59	0	50	0.0	0	0
10.10.0.10	10.10.0.101	59	0	49	0.0	0	0
10.10.0.100	10.10.0.101	59	59	0	1.09254	0	9
10.10.0.11	10.10.0.101	59	0	50	0.0	0	0
…							
10.10.0.69	10.10.0.101	59	59	0	1.1138700	0	12
10.10.0.70	10.10.0.101	59	59	0	1.18814	0	5
10.10.0.71	10.10.0.101	59	59	0	0.985055	0	3
10.10.0.72	10.10.0.101	59	59	0	1.18545	0	7
10.10.0.73	10.10.0.101	59	59	0	1.056320	0	7
10.10.0.74	10.10.0.101	59	59	0	1.15974	0	10
10.10.0.91	10.10.0.101	59	59	0	1.04291	0	5
…							
10.10.0.97	10.10.0.101	59	59	0	1.01804	0	9
10.10.0.98	10.10.0.101	59	59	0	1.08124	0	11
10.10.0.99	10.10.0.101	59	59	0	0.871741	0	4
10.20.0.1	10.30.0.2	5	4	2	66.29974	0	88
10.30.0.2	10.30.0.3	25	25	0	2.38817	2	12
10.30.0.3	10.30.0.2	25	25	0	2.500170	2	15
10.40.0.1	10.40.0.6	5	5	0	1.615728	1	5
10.40.0.2	10.40.0.6	5	5	0	2.815724	1	11
10.40.0.3	10.40.0.6	5	5	0	2.815724	1	11
10.40.0.4	10.40.0.6	5	5	0	3.0157240	1	12
10.40.0.5	10.40.0.6	5	5	0	2.815724	1	11
10.40.0.6	10.40.0.1	5	5	0	2.4261280	1	9
10.40.0.6	10.40.0.2	5	5	0	2.8261240	1	11
10.40.0.6	10.40.0.3	5	5	0	2.8261240	1	11
10.40.0.6	10.40.0.4	5	5	0	2.8261240	1	11
10.40.0.6	10.40.0.5	5	5	0	3.226124	1	13

Проблема масштабирования. Не всем датчикам удалось передать свои данные даже координатору, наблюдается потеря пакетов (например, датчики 10.10.0.1, 10.10.0.10, 10.10.0.11), средняя задержка соединения датчик-шлюз облака существенно увеличилась. При увеличении числа датчиков получаем проблему масштабирования WSN → Gateway – чем больше датчиков, тем чаще приходят пакеты, тем больше пакетов приходят на шлюз в

период агрегации, размер агрегированного пакета становится огромным даже при небольших пакетах сенсорных данных, тем выше задержка передачи, тем больше нагрузка на облако и CPU. Поэтому придется ввести некоторые ограничения, также, как это делается в реальных IoT/WSN/Edge системах [9]:

- ограничение размера агрегированного пакета (Maximum Transmission Unit, MTU), добавляем параметр maxAggBytes = 4096 KB, можно сделать его гибким, как у систем LoRaWAN/LPWAN/MQTT [10];
- ограничение числа пакетов в агрегации, maxPacketsPerAgg = 20, тогда даже при 200 датчиках в агрегации будет не больше 20 пакетов, это идеальный вариант для анализа влияния периода опроса;
- период агрегации должен быть пропорционален числу датчиков, при 50 датчиках, и агрегации 3 секунды, периоде опроса в 1 секунду, в агрегации будет 150 сообщений.
  aggPeriod = pollPeriod * (k), k = 1, 1.5.
  Пример тестовых серий:

датчики	период опроса, с	период агрегации, с
20	1.0	2.0
40	1.0	1.0
60	1.0	0.8

Для большего числа датчиков можно уменьшить период агрегации, чтобы не собирать слишком большой пакет данных. Также можно рассмотреть зависимость времени и потерь пакетов от периода опроса, периода агрегации и проводить различные эксперименты с сетью, менять количество пользователей, серверов, добавлять другие сетевые устройства.

Заключение. Был рассмотрен пример простой топологии облачной информационно-измерительной системы, в которой измерения от беспроводной сенсорной сети передаются на сервер данных в облаке через шлюз-координатор БСС и шлюз облака. К серверу данных могут обращаться пользователи через веб-сервер. Определены минимальное, максимальное и среднее время прохождения данных и количество переданных, полученных и потерянных пакетов для участков сети при разном количестве датчиков. При увеличении количества датчиков возникает проблема масштабирования датчики-агрегатор, которая может быть разрешена через ограничение размера агрегированного пакета, ограничение числа пакетов, подбор периода агрегации или горизонтальное масштабирование.

В целом можно сделать вывод, что такая модель может быть расширена, можно менять количество серверов данных и веб-серверов, шлюзов облака, шлюзов БСС, количество БСС, количество датчиков в БСС, выделить «узкие места» в топологии, применить различные методы решения и проверить их эффективность. Рассмотренная модель создавалась с целью анализа эффективности применения сетевых моделей NS3 совместно с Simgrid моделями [4]. Так как природа моделирования разная для NS3 (симулятор уровня пакетов) и Simgrid (симулятор уровня потока, передача симулируется как единый поток данных, занимающий пропускную способность канала), связка SimGrid-NS3 содержит только общие для обеих систем функции. В SimGrid доступны только модели NS3 TCP и Wi-Fi, и не все файлы платформы SimGrid можно использовать совместно с NS3 (маршруты должны быть длиной 1), платформа NS3, генерируемая из описания SimGrid, очень проста. Для использования этой модели необходимо установить NS3 и перекомпилировать SimGrid соответствующим образом [3].

Но можно использовать сетевую модель NS3 для модели Simgrid и другим способом – модель сети создать в NS, а логику приложений – в Simgrid (actors). NS3 может дать реалистичные сетевые задержки, которые затем SimGrid использует для планирования пересылки данных и выполнения кода на сервере и шлюзе. Таким образом можно использовать NS3 для обеспечения сетевой платформы Simgrid параметрами сети, измеренными при симуляции в NS3. Это хорошо также и для демонстраций, для NS3

существует множество сторонних приложений, позволяющих визуализировать топологию сети и даже запускать анимацию, показывающую прохождение пакетов.

Литература

1. What is Head-of-Line Blocking? https://jumpcloud.com/it-index/what-is-head-of-line-blocking
2. NS3 Network Simulator [Электронный ресурс] Режим доступа: https://www.nsnam.org/ (дата обращения: 21.11.2025).
3. Simulation of Distributed Computer Systems. [Электронный ресурс]. Режим доступа: https://simgrid.org/ (дата обращения: 05.08.2026).
4. Гайдамако В.В. Моделирование облачной информационно-измерительной системы с помощью библиотеки Simgrid // Проблемы автоматики и управления, г. Бишкек, Илим, №1 (36), 2019, с.90-99
5. NS3 tutorial. [Электронный ресурс] Режим доступа: https://www.nsnam.org/docs/release/3.46/tutorial/html/index.html (дата обращения: 21.08.2026).
6. Mathieu Lacage, Thomas Henderson. Yet Another Network Simulator. [Research Report] RR-5927, INRIA. 2006, pp.16. ⟨inria-00078318v2⟩ [Электронный ресурс] Режим доступа: https://inria.hal.science/inria-00078318/document (дата обращения: 21.08.2026).
7. Дядюнов, А. Н. Моделирование беспроводных сенсорных сетей / А. Н. Дядюнов,К. Н. Кузнецов // Научный вестник МГТУ ГА, серия Радиофизика и радиотехника. – 2009. – № 139. [Электронный ресурс] Режим доступа: https://cyberleninka.ru/article/n/modelirovanie-besprovodnyh-sensornyh-setey (дата обращения: 27.08.2026).
8. Беспроводные сети ZigBee и IEEE 802.15.4. [Электронный ресурс] Режим доступа: http://book.itep.ru/4/41/zigbee.htm (дата обращения: 21.08.2026).
9. Савин И. Ю., Блохин Ю. И. Об оптимизации размещения сети датчиков интернета вещей на пахотных угодьях // Бюл. Почв. ин-та. 2022. №110.
10. Efficient IoT Communication with LoRaWAN & MQTT. EMQX Team, Nov 5, 2024. [Электронный ресурс] Режим доступа: https://www.emqx.com/en/blog/lorawan-and-mqtt (дата обращения: 27.08.2026).