References

1. Грузман И.С. Цифровая обработка изображений в информационных системах: учебное пособие. – Новосибирск: Изд-во НГТУ, 2002. – 352 с.
2. Yakimenko O.A. Unmanned aircraft navigation for shipboard landing // IEEE Transactions on aerospace and electronic systems. – 2002 – V. 38. – № 4. – P. 1181–1199.

УДК 004.057.4

ПРИНЦИПЫ РАЗРАБОТКИ ПРОТОКОЛА ДЛЯ ПРОИЗВОДИТЕЛЬНЫХ СЕТЕВЫХ ПРИЛОЖЕНИЙ

А.А. Мулюкин[1]
Научный руководитель – к.т.н., доцент И.А. Перл[1]
[1]Университет ИТМО

В работе рассматриваются основные проблемы, которые возникают при разработке протоколов для производительных сетевых приложений (таких как компьютерные игры). И представлены основные способы их решения. Также представлен анализ и сравнение различных способов синхронизации объектов виртуального мира, и рассмотрены основные методы уменьшения размера передаваемых данных.
Ключевые слова: сетевые протоколы, разработка приложений, сетевые приложения, компьютерные сети.

В основу разработки любых сетевых приложений входит проектирование и разработка сетевого протокола, с помощью которого приложения будут взаимодействовать друг с другом. И во время разработки требуется понимать, с какими проблемами придется столкнуться. Выделяют три основных проблемы.

1. Пропускной канал ограничен и не стабилен. Размер пропускного канала для передачи данных ограничен, причем также стоит учитывать тот факт, что он нестабилен. В один момент это может быть одна ширина канала, а через некоторое время он сильно уменьшается из-за высокой нагрузки на сеть в целом или из-за проблем на третьей стороне. В основном для большинства простых приложений, это не является проблемой, так как количество передаваемой информации в сети очень мало.
2. Данные могут быть потеряны. Еще одна из проблем разработки сетевого приложения – потеря данных. Причины кроются, в основном, лишь из-за сетевой среды (неустойчивое соединение, маленькая ширина канала передачи данных). В основном проблема решается с помощью протоколов транспортного уровня, например, с помощью протокола TCP. В других случаях нужно учитывать этот факт при разработке своего сетевого протокола, который будет использоваться в приложении и либо реализовывать свою проверку доставки данных, либо построить архитектуру, в которой потеря данных не будет являться критичной.
3. Клиентские данные в большинстве случаев неверны и потенциально опасны. Суть данной проблемы заключается в том, что данные могут быть неверны из-за различных внешних факторов, таких как: вмешательство третье стороны, умышленная пересылка неверных данных. И при разработке приложения следует это учитывать. В связи с этим требуется позаботиться о проверки переданных данных на целостность и корректность.

Эти три проблемы также распространяются и на производительные сетевые приложения, но степень их проявления разная. В отличие от простых сетевых приложений, в компьютерных играх требуется отображать актуальную информацию о состоянии виртуального мира в реальном времени. Для плавной картинки в среднем используется частота кадров, равная 60 кадрам в секунду. В зависимости от типа/жанра/характера игры информацию требуется обновлять либо в пределах

секунды, либо чаще. Из-за такой частоты передачи данных возникает острая проблема нехватки пропускной способности, как на стороне клиента, так и на стороне сервера. Даже при обновлении данных раз в 5 кадров, синхронизация одного объекта (id, {*x*, *y*, *z*}, {yaw, pitch, roll}, {life, ammo}) требует 36 байт или 432 Б/с (3,4 кб/с). А если посчитать требования для 100 игровых объектов, то уже потребуется 345 кб/с. Также не стоит забывать о различных событиях, которые могут возникать в игровом мире и о дополнительной системной информации, которая используется для создания пакетов данных, что требует дополнительную пропускную способность. И если посчитать это число, то в среднем может получиться около 600 кб/с. И это расчет лишь на одно подключение, если же будет несколько подключений с аналогичными данными, то требования к пропускному каналу стремительно возрастают. Конечно, для современных сетей это не проблема, однако не стоит забывать про первый пункт проблем разработки сетевых приложений. А также не стоит забывать, что современные технологии до сих пор присутствуют не везде. Чтобы уменьшить это значение разработчики прибегают к различным хитростям: сжатие данных, событийные протоколы, интерполяция и т.д.

Любой сетевой протокол прикладного уровня основывается на протоколе более низкого уровня (транспортного уровня), например, на основе TCP или UDP. Ниже представлена таблица сравнения данных протоколов.

Таблица 1. Сравнение протоколов транспортного уровня

TCP	UDP
+ Использование соединения + Нет потерянных данных + Порядок доставки пакетов совпадает с порядком отправки пакетов данных – Тяжеловесный – Данные могут быть обрезаны, остаток может дойти позже	– Отсутствует соединение – Данные могут быть не доставлены – Порядок доставки пакетов может не совпадать с порядком отправки + Данные приходят целостно

На первый взгляд видно, что протокол UDP имеет много минусов и это ставит под сомнение его использование в сетевых приложениях. Да, для простых приложений – это не лучший вариант. Но если посмотреть на ситуацию с другой стороны, то минусы данного протокола становятся его плюсами, так как данный протокол позволяет разработать свою более гибкую систему установки соединения и проверки доставки пакетов. Например, для части потока данных можно использовать передачу с проверкой доставки данных и проверкой порядка доставки, для других данных можно было бы не использовать эти проверки, что ускорило бы передачу данных данного типа.

Во время разработки сетевых приложений также стоит задумываться о том, какую архитектуру сетевого взаимодействия можно выбрать. Есть две основные архитектуры, которые широко применяются при разработке.

1. P2P (PeertoPeer) – каждый с каждым. В данной архитектуре каждый из клиентов связывается друг с другом, и обмениваются сообщениями. У данной архитектуры есть один весомый минус, вспоминая основные проблемы сетевых приложений, данные могут быть неверными, поэтому доверять им нельзя. Решение есть, но оно не полное, собирать сообщения со всех соединений и подтверждать событие, только после совпадения данных с остальных клиентов.
2. Клиент-серверная архитектура. При данной архитектуре, все клиенты подключаются к выделенному серверу и пересылают данные другим клиентам только через него. В этой связи клиенты могут доверять данным, которые пришли.

Но вот сервер не может доверять пользовательским данным и поэтому на его стороне требуется осуществлять проверку данных на правильность и честность.

Общение клиента и сервера происходит посредством передачи сообщений.

Сообщение – это специальная структура данных, которая содержит информацию о каком-либо событии или действии, а также дополнительную информацию, которая необходима для обработки данного сообщения на стороне приложения. При составлении структуры сообщений стоит учитывать, что нужно использовать, как можно меньше бит, которые отводятся для хранения этого значения. Например, для передачи типа сообщения использовать 4 байтного целого числа не выгодно. По этой причине можно использовать 1 байт, если кол-во возможных типов сообщений не превышает 256.

Так как в рамках работы рассматриваются производительные сетевые приложения (компьютерные игры), то в основе них лежит синхронизация виртуальных миров. Для этого требуется передавать состояния объектов в игровом мире и для этого существует ряд способов, которые применяются при разработке.

1. Синхронизация по таймеру. Суть данного метода заключается в отправке данных об объекте с фиксированной частотой. Однако возникает проблема высокой нагрузки на сеть. Для ее решения можно снизить частоту отправки данных, из-за чего появляется неприятный эффект подергивания объекта на стороне другого клиента. Для решения этой проблемы приходят на помощь алгоритмы интерполяции и экстраполяции.
2. Синхронизация свойств. Данный способ синхронизации является улучшенной версией предыдущего метода, призванного уменьшить объем передаваемых данных. Для достижения данной цели можно синхронизировать объекты не полностью, а лишь частично, используя инкрементальную систему изменений свойств объекта. При использовании данного метода синхронизация происходит в два этапа:
   – с определенной частотой (T=10) происходит полная синхронизация объекта;
   – с большей частотой (T=1) происходит пересылка только измененных параметров заданного объекта.

   Причем измененные данные можно упаковывать в отдельное сообщение, которое содержит список измененных значений, а не синхронизировать свойства отдельными сообщениями. Тем самым можно также уменьшить объем передаваемых данных.
3. Синхронизация событий. Еще один способ синхронизации игровых объектов. Он заключается не в пересылке параметров игрового объекта, а в пересылке только событий, которые влияют на эти параметры. Данный метод широко распространен в играх жанра стратегии. Но все равно передачи одних лишь событий не достаточно для синхронизации. Исходя из этого, с небольшой периодичностью происходит полная синхронизация игрового объекта. События могут быть разного характера, например: игрок нажал на кнопку, которая отвечает за перемещения, игрок отдал приказ переместиться в определенную точку.

Для уменьшения объема передаваемого трафика используют различные способы сжатия. Сжатие трафика позволяет не только эффективно использовать канал передачи данных, но и дополнительно усложнить процесс подмены данных, что может незначительно увеличить безопасность передаваемых данных.

Ранее говорилось, что для уменьшения объема передаваемых данных можно упаковывать одинаковые по смыслу сообщения в один пакет. За счет того, что общая неизменная часть сообщений выносится и передается лишь один раз, вместо того, чтобы каждый раз дублироваться. Это может уменьшить объем передаваемых данных. Рассмотрим пример передачи измененных свойств для объекта.

Игровой объект:

– позиция (*X*, *Y*, *Z*: float) = 12 Б;
– ориентация (*X*, *Y*, *Z*: float) = 12 Б.

Структура сообщения для синхронизации свойства (8 Б):

– тип сообщения (byte) = 1 Б;
– идентификатор объекта (short) = 2 Б;
– идентификатор свойства (byte) = 1 Б;
– значение свойства (в данном примере всегда 4 Б).

Структура сообщения для синхронизации свойств (>= 9 Б):

– тип сообщения (byte) = 1 Б;
– идентификатор объекта (short) = 2 Б;
– количество измененных свойств (byte) = 1 Б;
– идентификатор свойства (byte) = 1 Б;
– значение свойства (в данном примере всегда 4 Б).

Таблица 2. Сравнение размера передаваемых данных

Количество, шт.	Размер сообщения, бит	
	Single	Batch
1	8	9
2	16	14
3	24	19
4	32	24
5	40	29
6	48	34
7	56	39
8	64	44

Как видно из табл. 2, скорость роста объема передаваемых данных в 1,5 раза меньше при упаковке измененных данных в один пакет.

Одним из основных способов уменьшения объема передаваемых данных – это использование эффективной разрядной сетки для передаваемых данных. Рассмотрим несколько простых примеров.

1. Передача логических значений. Во многих языках программирования логическая переменная (тип Boolean) использует целый байт (8 бит) в памяти, что является не эффективным использованием разрядной сетки, так как в байт можно записать 256 различных состояний, для данного типа достаточно два состояния. Поэтому для эффективного использования, для логической переменной достаточно использовать всего лишь 1 бит. Тем самым можно добиться уменьшения объема передаваемых данных (для данного примера в 8 раз).
2. Передача числовых значений. Многие при разработке своего приложения используют тип int (32 бита) для работы с числовыми значениями, даже не задумываясь, с каким диапазоном данных они работают. При разработке сетевых приложений это стоит учитывать, так как это очень важно. При не оптимальном использовании канала связи, могут возникнуть проблемы с производительностью. Например, мы работаем с числами, которые спокойно помещаются в диапазон от 0 до 200, например все доступные сетевые команды. Как говорится выше, многие, не задумываясь, используют int (32 бита) для передачи идентификатора типа команды. Хотя в данном случае достаточно использовать лишь байт (8 бит). Учитывая то, что идентификатор команды передается для каждого сетевого сообщения, то сокращение на 24 бита, может сильно отразиться на общем объеме передаваемых данных.
3. Оптимальный выбор диапазона передаваемых данных. Обобщая предыдущий пункт, можно сказать, что можно применять особые хитрости для некоторых типов

передаваемых данных. Одна из таких хитростей – это использование смещения. Например, используется следующие значения для передачи по сети: число от 500 до 512. Остальные числа по сети передаваться не будут, поэтому можно использовать смещение, в данном случае очень хорошо подойдет смещение 500. Таким образом, по сети требуется передавать лишь значения от 0 до 12, которые очень хорошо укладываются в 4 бита. Если же не использовать смещение, то для передачи данных потребуется использовать 10 бит, или еще хуже использовать тип short (16 бит).

4. Динамическая размерность передаваемых данных. Еще одним из способов для сжатия данных можно использовать 7 битное кодирование числовых значений. Суть данного кодирования заключается в использовании динамической разрядности для хранения числовых данных. Его очень удобно применять, в случае если передаваемые значения варьируются в большом диапазоне, например, от 0 до 2^{28}-1. При статичной разрядности требуется использовать 28 бит, чтобы передавать эти значения. Для экономии можно воспользоваться кодированием, в котором не используется фиксированная длина. Данные разбиваются на блоки по 7 бит, добавляется еще 1 бит, который указывает, что следующий байт являются последним. В итоге минимальной единицей передачи данных будет байт (8 бит).

Если сравнить два способа передачи данных, то можно сразу увидеть выигрыш в экономии трафика. Предположим, что данные генерируется генератором с равномерным распределением, тогда можно составить табл. 3.

Таблица 3. Сравнение разрядности данных при разной организации разрядной сетки

Диапазон чисел	Вероятность	Длина, бит	
		Статическая сетка	Динамическая сетка
0–2^7-1	0,25	28	8
2^7–2^{14}-1	0,25	28	16
2^{14}–2^{21}-1	0,25	28	24
2^{21}–2^{28}-1	0,25	28	32
Итог:		28	20

Как видно из табл. 3, при использовании данного кодирования можно сэкономить в среднем 8 бит информации.

Для экономии передаваемого трафика также можно воспользоваться различными способами сжатия данных. В отличие от ранее рассмотренных примеров, сжатие данных лучше производить не на отдельно взятых полях сообщения, а целого сообщения, так как алгоритмы сжатия в большинстве своем рассчитаны на данные большого объема, и поэтому они будут работать эффективнее на целом сообщении. При разработке своего протокола можно использовать один алгоритм, который будет применяться для всех передаваемых сообщений. Или же можно реализовать гибкую схему, в которой некоторые сообщения подвергаются сжатию, а некоторые нет. Потому что не стоит забывать и о том, что для сжатия данных требуется время, как со стороны сервера, чтобы сжать данные, так и со стороны клиента, чтобы распаковать их. И это не всегда может окупаться за счет сокращения времени передачи данных.

В зависимости от разрабатываемого приложения приходятся применять те или иные решения для реализации сетевого протокола. Но в большинстве случаев, как показывает практика, требуется использовать сразу несколько решений в рамках одного сетевого приложения. Зная о существующих проблемах разработки сетевых приложений, можно спроектировать правильный протокол, основываясь на предложенных способах решения данных проблем, чтобы в дальнейшем при росте нагрузки на сетевое приложение, разработанный протокол не дал сбой.

Литература

1. Алиев Т.И. Сети ЭВМ и телекоммуникации. Учебное пособие. – СПб.: СПбГУ ИТМО, 2013. – 400 с.
2. Ричард Стивенс У. TCP/IP Illustrated, volume 1. The Protocols / Протоколы TCP/IP. Практическое руководство / Пер. с англ. А.Ю. Глебовский. – СПб.: БХВ-Петербург, 2003. – 671 с.

УДК 004.9; 004.5

ФОРМИРОВАНИЕ КОНТЕНТА В СИСТЕМАХ ДОПОЛНЕННОЙ РЕАЛЬНОСТИ

К.А. Мурзанова[1]
Научный руководитель – к.т.н., доцент А.В. Меженин[1]
[1]Университет ИТМО

В работе рассматривается формирование контента как в общем, так и в частности, на примере формирования контента при создании приложения, с использованием одной из наиболее популярных систем дополненной реальности Metaio SDK. Производится обзор существующих SDK дополненной реальности и обоснование выбора Metaio SDK.
Ключевые слова: дополненная реальность, AR, Augmented Reality, контент.

Дополненная реальность (Augmented reality, AR) – это технология, которая позволяет наложить информации в форме текста, графики, аудио и других виртуальных объектов на реальные объекты в режиме реального времени [1]. Чаще всего применяется компьютерная графика в формате 2D или 3D. В отличие от виртуальной реальности (Virtual Reality, VR), которая подразумевает полное погружение человека в виртуальную среду, дополненная реальность расширяет пользовательское взаимодействие с окружением [2].

Для создания мобильных приложений с дополненной реальностью разрабатываются специальные комплекты SDK под различные платформы (windows, android, ios). Например, такие как: ARToolKit, Qualcomm Vuforia, Metaio SDK.

Vuforia SDK – это библиотека для мобильных устройств, которая позволяет создавать различные AR-приложения. Vuforia SDK поддерживает 2D- и 3D-форматы, а также имеет возможность использовать системы без маркеров. Vuforia SDK создает приложения под операционные системы (ОС) IOS и Android.

ARToolkit – это библиотека для создания приложений с дополненной реальностью. ARToolkit использует возможности видеослежения, а также расчеты ориентации и реального положения камеры в соотношении с квадратным физическим маркером в режиме реального времени. 3D-модель размещается на реально существующий маркер, в тот самый момент, когда положение камеры известно.

В настоящее время ARToolkit функционирует как проект с открытым исходным кодом. ARToolKit широко используется, но имеет ограниченные возможности.

Metaio SDK – программное обеспечение для создания мобильных приложений с дополненной реальностью. Metaio использует технологию компьютерного зрения, что позволяет распознавать, отслеживать реальные объекты в режиме реального времени. Для того чтобы виртуальный объект казался частью реального мира, точка зрения зрителя на объект соотносится с их точкой зрения на изображение [3].

Для исследования формирования контента было выбрано Metaio SDK – бесплатная библиотека, которая постоянно улучшается и способна использоваться для создания приложения под различные ОС.