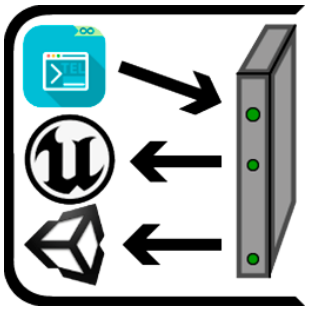
Спецификация протокола общения клиентов с сервером. (v1)

1. Предназначение



Протокол предназначен для сообщения серверу от клиента и клиентам от сервера определённых параметров, следующих с пакетом данных (что помогает серверу(клиенту) принять решение о том, как обработать пакет), а также для определения сервером и клиентом границ одного пересылаемого пакета и неправильно сформированные пакеты.

В общем-то использование протокола было бы необязательным, если бы небыло надобности как-то разделять пакеты (случай TCP) т.е. всё шло бы одним потоком, в котором были бы данные, начало которых восстанавливалось бы по неким текстовым меткам. Пример: “data:{width:10;height:40}”. Это налагало бы ограничение на текст, идущий как «данные», а не как сигнализатор начала. В общем, добавилось бы лишних проблем. С UDP тоже интересно вышло бы. В UDP, конечно, небыло бы проблемы с определением начала и конца сообщения, т.к. один пакет – одно сообщение. Однако, тут взиграла бы другая проблема – пакеты перемешиваются т.е. приходят не в том порядке, в котором их отослали. Как раз этот случай (в UDP) использование протокола общения клиента и сервера предотвращает.

Сервер поддерживает связь по протоколам TCP и UDP. Их предполагаемые роли в процессе взаимодействия клиентов (через сервер, разумеется) довольно различны. По TCP предполагается передавать только важную информацию, которая обязательно должна быть доставлена. К такой информации относятся update по связям нод (куб как нода), т.е. данные, в которых говорится о том какая нода с какой связана, что б каждый клиент мог обновить имеющуюся у него модель (и внести корректировки в играющую музыку). Так же через этот протокол предполагается запрос нового клиента к уже включенным для получения у них состояния игры. Если говорить в общем, то по TCP предполагается передавать данные, которые будут отосланы одинажды (какое-то событие, например) и обязательно должны быть приняты всеми.
С UDP по-другому. По UDP предполагается пересылать данные, которые меняются очень часто и сохранение скачков (1,20,2,19) в которых не критично (чтоб можно было интерполировать), потеря пакета данных не должна быть критична, так как в очень скором времени придёт другой пакет. К таким данным можно отнести позицию игроков (положение целиком, положение отдельных частей, положение рук, повороты всего этого, так как они меняется часто из-за того, что игроки двигаются, также из-за шумов в сенсорах положения).

2. Адреса и рассылка

Поговорим о сервере. Сервер, по сути, выполняет чисто передающую роль. Его задача – передать данные клиентам. Когда клиент отправляет данные сервер должен переслать их клиентам. Сервер поддерживает 2 типа пересылки: broadcast и unicast. Предполагается преиущественное

использование первого типа, поэтому сначала о нём. В этом типе клиент отправляет данные на сервер, а сервер пересылает эти данные всем клиентам, кроме того, который отправил данные ему. *Почему? Потому что у того клиента данные и так есть, и нет смысла перегружать сеть, сервер и клиент. Для того, чтобы не переслать данные клиенту, отправившему эти данные, нужны адреса.* Каждый клиент имеет свой адрес. Наличие адресов позволяет использовать ещё один тип рассылки – unicast. В нём клиент указывает какому конкретно клиенту он хочет переслать данные и сервер шлёт данные только на указанный адрес.

3. Использование TCP

Соединение TCP представляет из себя поток (stream). Для определения начала и конца сообщения вводится заголовок, состоящий из 5 байт. Устроен заголовок следующим образом (нумерация по индексам):

0 байт	1 байт	2 байт	3 байт	4 байт
length		control	length	

length – длинна сообщения без заголовка (длинна всего сообщения минус 5 байт). Имеет одинаковое значение в 0-1 и в 3-4 байтах. Нужно это для обнаружения ошибки определения заголовка. Если байт 0 не равен 3 байту или байт 1 не равен 4 байту, то сервер понимает, что в определении заголовка произошла ошибка и закрывает соединение с клиентом. *Почему закрывает? Потому что ошибка могла произойти только на стороне клиента, и, если что-то произошло, значит что-то не так с клиентом – нет уверенности, что он шлёт правильные данные ниже заголовка (с 5-го индекса).* Длинна представлена беззнаковым типом (unsigned short int, uint16).

control – это байт для атрибутов, точнее, 4 бита для атрибутов.

Таблица битов (по значениям) в control:

128	64	32	16	8	4	2	1
through	through	through	through	affecting	affecting	affecting	affecting

through – «сквозные» биты. Они не принимают никакого участие в работе протокола общения клиента и сервера и, по сути, являются просто данными. *Однако, рекомендуется закладывать сюда «тип» сообщения клиенту (не обрабатывающей стороне протокола, а части, где происходит интерпретация данных, пришедших в блоке данных). По этому типу интерпретирующая часть может определять какие данные идут после 5 байта (чтоб не занимать лишний байт): структура с моделью связей, структура с какими-то другими данными, вообще не структура и т.д.*

affecting – биты, предназначенные для обработчика протокола. Эти данные определяют путь пакета и наличие содержимого.

0 – broadcast-пересылка данных (начинаются с 5 индекса, 5 индекс включается в данные)

1 – запросить список активных адресов (т.е. адресов, на которые можно слать данные unicast'ом), пакет идёт только до сервера (не пересылается), дополнительных байтов данных быть не должно (размер пакета равен 5 байтам), сервер даст ответ со значением affecting = 1 (как и клиент посылал). Ответ может быть пустым, если клиентов кроме пославшего нет. Если клиенты есть, то их адреса будут занимать по байту (1 адрес – 1 байт в блоке данных). Пример ответа:

0 байт	1 байт	2 байт	3 байт	4 байт	5 байт	6 байт	7 байт
length		control	length		addr.	addr.	addr.
3		1	3		1	2	8

Заголовок Блок данных

2 – unicast-пересылка данных. В байт по индексу 5 ставится адрес назначения. Сервер меняет этот адрес на адрес отправителя, чтобы получатель мог отправить ответ. 5 байт в unicast становиться неофициальной частью заголовка. «Неофициальной» потому, что включается в длину блока данных (length). Т.е. длинна данных отправителя = length – 1. Пример:

0 байт	1 байт	2 байт	3 байт	4 байт	5 байт	6 байт	7 байт
length		control	length		addr.	data	data
3		2	3		2	120	10

Блок данных

Фактический блок данных

3 – получить состояние клиента по адресу (активен/не активен) (самому клиенту не пересылается). *По сути, можно получить список индексов, а потом проверить, есть ли нужный среди них, но это не рекомендуется, так как медленнее (в силу реализации на сервере в том числе). Так что для проверки состояния клиента лучше использовать команду «3».* Нужно это может быть при обмене сообщениями unicast'а, чтоб понять, что другой клиент активен и ему можно слать сообщения. Длинна пакета для запроса и при ответе равна 6 байт. 5 байт – заголовок, как помните. 1 байт при отправке – адрес клиента, активность которого нужно проверить; при получении – состояние (1 - активен/0 - не активен).

4 – обнулить счётчик UDP для данного клиента (об UDP читайте далее). Не содержит блока данных, не пересылается. Вот такой пакет:

0 байт	1 байт	2 байт	3 байт	4 байт
length		control	length	
0		4	0	

Остальные команды на данный момент не используются.

4. Работа сервера

Поговорим немного о TCP клиентах «глазами сервера». На каждого клиента отводится отдельное подключение (особенность TCP). Это соединение является основным для сервера. Без этого соединения UDP-рассылки работать будут некорректно, так как именно TCP соединение получает адрес, который впоследствии, ставится во все, исходящие от клиента, UDP-пакеты *(в случае broadcast; в случае unicast ставится адрес получателя)*. Создаётся некое «UDP-соединение», которое снабжается (адресом) через TCP-соединение. Когда клиент подключается, для него создаётся *индивидуальное окружение*. Туда поступают пакеты, идущие к клиенту *(на самом деле функций у окружения куда больше; одна из таких функций – хранение состояния активности клиента)*. После создания соединения и его индивидуального окружения, второе по окончанию соединения не удаляется, а устанавливает значение неактивности. Зачем это знать? Даже на неактивные соединения можно слать пакеты, вот только уйдут они вникуда, но зато соединение, отправившее пакет на неактивный адрес, не будет сброшено, так как пакет просто пойдёт в неактивное окружение, откуда удалится при обновлении такта. Все пакеты отсылаются каждый такт, а потом удаляются с сервера. Есть ещё один момент, связанный с определением неактивности клиентов. Может быть так, что клиент не завершит соединение корректно (кстати, сервер соединение корректно завершает всегда). Тогда получится так, что соединения фактически нет, но на сервере оно существует и является активным. Однако, активность такого соединения поправима – при любой попытке отправить что-то (TCP или UDP) в это соединение, сервер поймёт, что оно неактивно. Пакеты удалятся, а в окружение соединения будет установлено значение неактивности. Если в этот клиент часто направляются пакеты (не важно broadcast или unicast), то вероятность «висящего» соединения заметно уменьшается. На самом деле, «висящие» соединения не создают никаких осложнений кроме 2 моментов: запрос состояния клиента, получение другим соединением адрес повисшего соединение. *Да, повисшее соединение будет занимать адрес, а пока адрес будет занят, сервер не сможет дать его другому клиенту.*

Важно знать, что адрес назначения UDP, отправляющихся к клиентам берётся из адреса входящих UDP (хранится в окружении соединения TCP, для каждого TCP-клиента собственный адрес назначения для UDP). Для того чтобы получать пакеты UDP, клиенту нужно сначала самому отправить пакет UDP на сервер (со своим TCP-адресом). Тогда сервер поймёт, на какой адрес (IP + port) необходимо отправлять UDP и к какому клиенту они будут направляться.

После установления «UDP-соединения» его нужно поддерживать. Для этого клиенту делать ничего не нужно – сервер делает всё сам. Каждые 30 секунд он отправляет на все известные UDP-адреса пакет со значением

conrol = 4. Клиенту не нужно никак реагировать на данный пакет, его можно просто отбросить.
Нужно сказать, что сервер пересылает пакеты не в режиме реального времени. Есть некоторая задержка. Задержка эта специальная и нужна она для замены старых UDP-пакетов. На сервере стоит таймер, который срабатывает каждые 50 миллисекунд (1/20 секунды).

5. UDP: процессинг и использование

В пакетах UDP так же присутствует заголовок, но в нём уже 11 байт. UDP не создаёт соединений, поэтому пакеты раздельны и в stream не превращаются. У данных, отправляемых UDP есть максимальный размер – это 1432 байта без учёта заголовка. Максимальный размер для блока данных – 1421 байт. *Процесс рассылки UDP устроен несколько сложнее, однако, это не даёт значимого минуса к скорости, но позоляет упорядочить UDP-рассылку.*
Вот такую структуру имеет заголовок UDP в протоколе:

0	1	2	3	4	5	6	7	8	9	10
length		type	control	time				length		addr.

length – всё так же, как и в TCP, только поменялось положение второго “length”. В UDP “length” играет исключительно контрольную функцию. К сравниванию “length” здесь ещё добавляется сравнение с реальной длиной блока данных (длинна пакета - 11). Если они не совпадут, то пакет отбросится.
addr. – это адрес клиента-отправителя пакета в случае broadcast и адрес клиента-получателя в случае unicast. В случае unicast адрес отправителя теряется.
type – это некоторые данные клиента, которые можно отнести к блоку данных. Это почти то же самое, что и through-биты в TCP. Однако, тут функция больше. Заключается она в участии этого поля в группировке UDP пакетов и в отбросе старых UDP.
time – поле unsigned int (uint32), в котором хранится текущее время отправителя пакета. Это время используется для определения старого UDP пакета.
Процесс идёт так: клиент шлёт несколько UDP пакетов серверу. Пакет «А» имеет тип (значение в type) «32» и время «8»; пакет «Б» - type 32, time 9; пакет «В» - type 16, time 10. Обратите внимание на то, что клиент в поле time записывает значение счётчика пакетов (что никак не связано со временем) т.е. чем больше пакетов отправлено, тем больше счётчик. Когда счётчик клиента дошёл до наибольшего значения unsigned int, счётчик клиента сбрасывается, а на сервер отсылается TCP пакет для сброса (control = 4) счётчика пакетов клиента на сервере. Задача использования счётчика – за 50 миллисекунд оставить для пересылки по одному пакету каждого типа (type). Если пакета какого-то типа для пересылки нет, то он добавится для пересылки без учёта значения счётчика, а если есть, то

будет проверено значение счётчика в пришедшем пакете и если оно будет больше того, которое хранится в окружении соединения, то пришедший пакет заменит старый такого же типа. Счётчик окружения станет равным счётчику пришедшего пакета. Таймер на сервере ведёт счёт (считает кол-во прошедших по 50 миллисекунд). Считает он глобально (число счётчика одинаково для всех клиентов), независимо от наличия клиентов и их активности. Значение этого счётчика ставится в поле time UDP пакетам, которые идут от сервера клиентам.
Вернёмся к «АБВ». Итак, клиент отправляет пакеты А, Б, В так, что они приходят к одному такту сервера (таймер ещё не изменил значение счётчика и пакеты такта ещё не начали отправляться, а только собираются для замены старых пакетов новыми). Сначала приходит А (допустим, первым из типа «32») – он помещается на отправку. Потом приходит Б – он заменяет А так как у него такой же тип, но счётчик больше. Теперь на обработку идёт В – он помещается на отправку, но не заменяет ни А, ни Б так как у него другой тип, хоть счётчик больше.
Со стороны клиента в большинстве случаев можно принимать все UDP пакеты (не обращая внимания не time). Это потому, что в большинстве случаев пакеты, отправленные с разницей в 50 миллисекунд, не смогут перемешаться, так как будут далеко друг от друга.
Может возникнуть вопрос: послан запрос на обнуление счётчика пакетов соединения на сервер, сервер выполняет обнуление, но, вдруг, на сервер приходит пакет со старым номером (большим числом, которое близко к максимальному) и что делать? Ответ такой: ничего делать не надо. Сервер по разнице значений поймёт, что пришедший пакет старый и отбросит его. Разница для отбрасывания должна составлять 2147 483647 и более (что равняется ~3.4 годам по 50 миллисекунд).
control – 1 байт, 4 бита из которых, как и в TCP, - through, 4 – affecting. *В отличие от TCP, тут «тип» рекомендуется помещать не в control/through, а в type, чтобы «тип» принимал участие в группировке пакетов.* Так же, в отличие от TCP, тут позиции «1» и «2» являются флагами, т.е. управляют параллельно.
1 [флаг] – если установлен, то механика группировки пакетов применяется к этому пакету, если нет, то независимо от типа и значения time пакет будет помещён на отправку. time в этом случае проходит сервер «насквозь», т.е. не меняется от клиента к клиенту.
2 [флаг] – если установлен, то пакет имеет тип unicast, если нет – broadcast. *(Обратите внимание на установку адреса при изменении типа пакета.)*
4 [не флаг] – «заявить о себе» в UDP без пересылки broadcast/unicast, пустой блок данных. Означает что пакет может быть отброшен. *Отбрасываемые сообщения нужны для сохранения «UDP-соединения» клиентов с сервером. Рассылаются сервером автоматически.*
8 [не назначен] (на данный момент) – отбросить пакет.

6. Процесс взаимодействия

Тут будет рассказано о том, как выстроить взаимодействие клиента с сервером. Итак, начнём. Начнём с того, что все примеры кода будут приведены на языке C#, так как это «родной» язык сервера, однако, это никак не влияет на возможность программ на других языках подключаться к нему.

Для подключения по TCP в C# используйте socket (System.Net.Sockets):

```
Socket TCPSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

После подключения ждите получения пакета с адресом (сервер отсылает автоматически). Код может получиться примерно таким:

```
TCPSocket.Connect(ipPoint);
while (TCPSocket.Available == 0) Thread.Sleep(20);
{
    byte[] addr = new byte[6]; //Тут объявляем массив с 1 байтом в блоке данных
    TCPSocket.Receive(addr); //Тут получаем данные в массив
    my_addr = addr[5]; //Тут получаем адрес клиента
}
```

Получить собственный адрес можно только один раз.

После получения адреса можно отправлять пакеты на сервер и получать рассылки и ответы (получение и отправка должны быть асинхронны друг от друга для хорошей производительности). Однако, перед началом отправки и приёма сообщений, нужно включить KeepAlive-сообщения в TCP сокете. В C# это делается так:

```
TCPSocket.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.KeepAlive, true);
```

Теперь можно послать любой пакет UDP на сервер (broadcast/unicast), чтобы «заявить о себе». Вот пример «заявления о себе»:

```
UDPSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
byte[] buffer = new byte[] {1, 0,  0,   0,    0, 0, 0, 0, 1, 0, my_addr, 0xFF};
//                                  len|type|con|  time     |len   |    addr.     | data |
UDPSocket.SendTo(buffer, ipPoint);
```

Надо заметить, что заявлять о себе в UDP не обязательно. Можно работать только с TCP. Если не заявить о себе по UDP, то не будет надобности реализовывать приём UDP так как UDP не будет отсылаться на клиент, который «не заявил» о себе.

Спецификация закончена, приятного пользования протоколом!