Итак, король протоколов прикладного уровня. 99% сайтов в интернете, протокол http прикладного уровня протокол. То есть он в себе, на самом деле, объединяет протокол как, значит…, такой протокола, как http-ng, который преобразует ваше представления в байты и байты обратно в представления. То есть протокол 6 уровня, а сам по себе является протоколом прикладного уровня. Это значит, что в нем уже реализовано много фишек, которые позволяют вам не просто кидать байты, а уже каким-то смысловым наполнением их наполнять. Наполнять смысловым наполнением. Вот так вот я читаю лекции. Значит, уже позволяет как-то управлять данными. То есть уже содержит в себе какую-то логику, логику обработки данных, а также, забегая вперед, содержит в себе логику, например, архивации данных – это все делается на 6 уровне.

Имеет несколько версий, версий в протоколе http 1.1 и версий в протоколе http 2 работают поверх TCP соединения. Версия протокола http 3 использует свое собственное соединение 4 уровня. То есть, как мы только что выяснили, да, не только TCP и UDP, есть еще несколько протоколов, которые также позволяют данными обмениваться, используют свой протокол, протокол QUIC.

Значит http – это протокол запрос-ответ, это значит, что, когда вы делаете запрос, у вас все данные уходят, это уже вас не интересует, как они там бьются на эти всякие TCP пакетики, да. Вас что интересует? У вас ушел запрос и у вас приходит ответ. И вот таким образом вы работаете. То есть, есть протокол, который, например, одновременно позволяет вам транслировать данные в обоих направлениях, http – это все-таки протокол, когда вы что-то спросили, вам что-то ответили. В принципе, тоже самое, что мы с вами только что делали с TCP и UDP.

И соответственно, если вы работаете с каким-то API, я не буду говорить с любым API, есть API работающие совершенно через другие протоколы, то скорее всего это будет что-то Over http. Особенно, если это API какого-то там, значит, сетевого

обменника или какой-то отдачи данных с базы данных, или сборки каких-то логов. Ну в общем, там практически всегда есть какой-то http.

Итак, http – протокол прикладного уровня. То есть мы не просто засовываем уже данные в виде массива байт в UDP пакеты и прислать какие-то ответы. Мы уже в этих ответах какую-то включаем логику. То есть, например, вот я используют клиент называемый insomnia. Кто, соответственно, помнит Postman, кто работал с ним – это такое ответвление от Postman-а, которое весит чуть меньше, работает чуть быстрее. В общем, это приложение заточено специально под работу с всякими HTTP-шными API.

И даже уже здесь мы что можем видеть? Во-первых, появились у нас адреса. То есть не просто у нас здесь адрес 8080 с port-ом. У нас, во-первых, есть приставка, которая обозначает: защищенное ли у нас соединение или нет? Уже мы видим инкапсуляцию код протоколов, то есть оно может быть шифрованное, нешифрованное.

У нас здесь есть методы, то есть у HTTP есть глаголы, которые позволяют вам специфицировать содержание этого запроса. Например, метод GET показывает, что вы можете передать в запросе параметры в строке, в строке запросом здесь в URL-е, так называемом, в адресе запроса. Вы можете передать туда так называемые header-ы. То есть header-ы это какая-то штука, которая сверху в HTTP-запрос кладется и позволяет его как-то специфицировать. Соответственно, вы можете передать отдельно header авторизации, то есть тоже, на самом деле, header. И можете что-то передать в body, но в GET-е этого не делается, это, например, делается в POST-е. То есть мы уже видим, здесь какая-то логика есть, здесь какие-то глаголы появились, куда-то можно еще обратиться, какие-то адреса есть.

При посылке запроса у нас есть ответ в ответе есть body – это строковое значение, но тут опять же можно передать в header-е, например, что эти значения будут определенного формата, например, формата JSON или YAML. Мы о них поговорим несколько позже в лекциях, и придут они именно в таком формате, то есть вы это можете проверять в своих клиентах и протокол вам позволяет эти все данные добавить, настроить и их принять.

Может вам прийти в ответ вообще HTTP, то есть как наш браузер это делает – загружает страницы и отсылает HTTP. Например, если я захожу в Wikipedia-ю, кликаю Inspect, вижу тут вкладочку Network. Здесь я, например, делаю команду All и, например, вижу, что здесь у меня появились какие-то script-ы, какие-то документы, какие-то у них статусы есть. В общем, все классно. И здесь в принципе, происходит довольно-таки, много вещей.

Опять же у запросов…, у ответов на запросы есть код, который вам показывает, что запрос там удался, не удался. Есть header-ы, опять же, присылаемые, которые вам показывает в каком виде приходит ваш контент, какая его длина, когда он пришел. Вы можете передать специальные файлики, которые хранятся в браузере. То есть это такой полноценный, прикладной, это значит, что он реализуется на уровне приложения Format, которым вы можете пользоваться для того, чтобы передавать много всяких данных и над ними какую-то логику навешать. Потому что вы можете здесь опять же, передавать дополнительные header-ы, передавать какие-то данные, передавать… То есть это не просто какая-то строка байт, которую передали и все. У вас тут есть уже какое-то логическое разбиение.

В основе своей, естественно, это все кодируется и опять же в строку байт – кодируется и раскодируется, но тут уже намного больше всяких крутилок, которые

вы можете двигать для того чтобы свои приложения писать, свои сервера писать, своих клиентов писать.

И давайте посмотрим на простой пример http-сервера, который я сейчас вам покажу. Опять же он простой, потому что я хочу показать концептуальное отличие протоколов прикладного уровня от транспортных протоколов. И да, http работает поверх TCP, напоминаю. То есть под капотом там все то же самое, но опять же, с логикой.

Для поднимания http-сервера мы опять же будем пользоваться великим и ужасным пакетом net, но его разделом http. Соответственно, здесь в пакете net существует еще и mail, существует rpc и прочие приятные нам штуки. Соответственно, http сервер здесь полноценный – он умеет и в cookie, он умеет и в обработку различных динамических адресов, но что мы здесь видим, что нам важно здесь? – Мы видим handler-а, то есть http сервера работают вообще по принципу: ты даешь значит ему url… У url-а есть после слеша. То есть это просто…, это как раз описано в стандарте. У url-а после слеша есть какое-то слово или 2 слова, или 3, или 5, что-то разделенное слешами, и эти слеши являются matching-ом адреса. То есть мы в данном случае говорим, что «/hello» – это будет handler hello. Слэш headers будет handler-ом headers обрабатываться. Дефолтный адрес для http: 80. Соответственно, я повесил на 88, потому что на моей машине существует http-сервер, поднятный на 80 port-у – это не важно, на самом деле. 8080 является стандартным дебашным port-ом для http.

И здесь у меня всего для метода. 1 метод просто: опять же, здесь мы видим – здесь есть пафосный ResponseWriter, который в себе имеет header-ы, который может что-то там писать в себя, писать header-ы. Почему? Потому что в ответе на запрос есть header-ы – это часть ответа на вопрос. Вследствие чего здесь Writer немножко другой, но, тем не менее, он поддерживает этот стандартный метод. Вы

туда тоже можете как со стримом работать, с ним. И, соответственно, сам http все сделает: http вызовет все handler-ы и соответственно, обработает. В первом случае он просто выведет hello – мы это видим уже как раз в пакете, то есть вот здесь он отдал нам hello.

И соответственно, headers нам напечатает все header-ы, которые у нас были запросе. Давайте посмотрим, как это выглядит.

Соответственно, что здесь, у меня? У меня здесь стандартные header-ы, и мы здесь видим, что, во-первых, стандартный header есть Accept. Опять же это часть одного из форматов описывающего http, что клиент готов принять. Это значит, что он что угодно готов съесть. Вы можете передать туда например, что client принимает application json, и любая имплементация http client-а практически, в любом языке отклонит ваш пакет, если сервер вернет что-то другое. И user-Agent – это важно, это тоже одна из частей спецификации. Она позволяет вам показать, что вы это вы. То есть что вы, например, компьютер под управлением Windows с браузера Chrome, что вы Firefox. Он может быть любым – это свободно назначаемая строка, но он есть какой-то default-ный. Я могу сюда передать какой-то еще header в мой сервер вполне спокойно. Опять же обращаю внимание: запрос-ответ. Я послал запрос – мне вернулся ответ. Никаких тебе открытых каналов с динамическим возвратом. Они на самом деле, есть в http2, то есть там можно сделать такое duplex-ное соединение, когда еще и обратно данные льются. Но в целом http работает так.

Здесь у меня есть новый header, опять же формат header-ов – это имя и, соответственно, что там внутри. В общем-то в общих чертах – это все. В типе Request у нас содержится метод http соединения, значит, содержится что у нас в соединении? – Протоколы, версия протокола http, содержатся header-ы соединения, body и всякая длина строки. Также запрашиваем соединение, то есть

host соединения. Это машина, от которой запросили соединение. И прочие всякие вещи, уже относящие к непосредственно http.

Http очень большой, поэтому здесь я покажу только базовые вещи. Как вы можете видеть да, здесь уже все более цивильно. У вас не просто read и write-ы на соединениях, у вас уже какой-то там ResponseWriter, из которых можно что-то получить. У вас какая-то целая структура, в которую вы можете что-то засунуть, то есть всё уже не так просто м в то же самое время – не так сложно.

Поверх http делается очень много различных протоколов, уже различных API-протоколов, так как называемых over-http-протоколов, но они, как правило, специфицируют конкретную логику, что для моего сервера этот GET будет обозначать, что ты только читаешь данные, а POST будет обозначать, что ты меняешь данные. Поэтому мой сервер будет так на них реагировать. Но это уже не имеет отношения к самому протоколу http. Описывается обычно другими протоколами.

Небольшое дополнение: http сервера бывают http, бывают https, то есть защищенное соединение. Соответственно, под капотом самого протокола, то есть у вас тоже все GET-ы, POST-ы, все дела – сохраняются, но на представительском уровне…, вернее, даже на, наверное, сессионном уровне – у вас появляется дополнительная прослойка так называемого security. То есть прослойка, которая кодирует все ваши данные и отправляет зашифрованный пакет, потому что http – это по сути своей просто текст, и он никак не зашифрован. Если вы просто по интернету кидаете http-запросы, и любой человек, который сидит на обмене трафиком, анализирует трафик, может этот запрос перехватить и прочитать, поэтому если у вас есть какие-то secur-ные запросы, то их надо бы шифровать.

Шифрование ассиметричное, соответственно, с использованием публичного приватного ключа. Такие же, практически, как мы генерируем для SSH. И для этого есть такой метод ListenAndServeTLS, который в себя принимает сертификаты, ключевые файлики, а дальше все то же самое, то есть переделать ваш http сервер в https – это настолько же просто, насколько изменить один метод и добавить туда сертификаты.

Итак, я показал, как работает сервер, теперь давайте посмотрим, как работает client в Go. Кстати, давайте также обратим внимание на то, что если я пришлю запрос на несуществующий адрес, то мне вернется какая-то стандартная ошибка. Это часть тоже протокола http: 404 ошибка not found. И она уже зашита обычно в http-серверах – это тоже преимущество протоколов прикладного уровня и преимущество автоматизации, то есть преимущество стандартизации, что вы точно знаете, что, как правило, в 99% случаев это когда у вас нет адреса, не обслуживается адрес этот сервером – у вас будет ошибка 404, очень удобно.

Соответственно, как происходят запросы со стороны client-а? Вы делаете метод, то есть вы пользуетесь пакетом http net…, то есть «net/http», который в себе содержит методы GET, POST, соответствующие этим глаголам. Тут есть еще остальные глаголы, но они менее используются, поэтому здесь они не представлены, их нужно делать через сборку запросов, я покажу это чуть позже.

Вы получаете ответ, такой полноценный response со Status-ом, со StatusCode-ом, с Protocol-ом, с какими-то ключами и с Body. Body – это как раз, то, что у вас приходит тут. Body надо прочитать, Body – это readCloser-ы, это то есть интерфейс, который в себе соединяет reader-а, из которого можно читать и closer-а, который можно закрыть. То есть в spec-е http написано, что это ответственность вас закрыть body, потому что http тоже может приходить нарезанное кусочками, и вы должны это всё прочитать.

Дальше мы используем обычно наш scanner, чтобы прочитать этот body, и что мы в scanner-е прочитали – мы можем вывести. Есть какая-то ошибка, тут у меня в данном случае panic. В вашем client-е вы можете просто закрыть соединение и вызвать следующий запрос. Здесь ничего особенного, здесь мы просто вызываем запрос, читаем текст и его выдаем.

И соответственно, более сложный запрос, то здесь я обращаюсь к url headers, то есть можно видеть здесь адрес опять же. И опять же, обращаюсь с помощью метода GET, но я здесь собираю Reguest. То есть Reguest – это уже некоторая структура, которой можно передать параметры, то есть методы, и которая будет передаваться в дальнейшем в Client. То есть этот Get, который мы используем – это, на самом деле, просто оболочка над Get-ом, который просто собирает этот Request сам по себе.

Мы в данном случае тоже делаем GET, но здесь может быть PUT, может быть OPTIONS, может быть что угодно. И вы можете к этому Request-у добавить header-ы, также здесь я не использую http.Get, то есть http содержит в себе стандартный client, который создается по умолчанию. Можете создать client-а со своими настройками. Здесь, опять же, их куча: здесь транспорты, следователи, Redirect-ы, потому что в http также есть в стандарте определение о том, что вы можете, например, вернуть протокол 301, 302 и в body будет определенный ответ. И этот ответ будет содержать другой url, и ваш браузер автоматически по нему перейдет, либо ваше приложение автоматически по нему перейдет. Можете переопределить это поведение в вашем клиенте, например, можете переопределить время ожидания ответа на запрос и так далее. То есть здесь какие-то есть настройки, поэтому можете создать своего client-а и собирать запросы самостоятельно, добавлять в них header-ы, например, потому что через вот такой вот метод я ничего не могу сделать – никакие header-ы добавить. И

через Do получить такой же response, который вы также можете scanner-ом прочитать.

Здесь я читаю еще дополнительно header-ы, здесь можно увидеть, что это никак не отличается от того, что я на сервере пишу header-ы. Точно также я через for, range читаю header-ы и пишу их на сервере. Здесь я просто их вывожу в консоль на клиенте.

И, соответственно, тоже также печатаю статус и печатаю ответ на запрос.

Эти два метода работают последовательно друг за другом. Это ответ первого метода, то есть response статус, и текст ответа, body ответа.

Во втором методе я тут показываю: у меня есть header-ы, которые мне ответил сервер, а сам сервер ответит, что ему передал client, то есть в данном случае выведется такая портянка – это то же самое, что я здесь передавал, то есть наши тестовые header-ы.

Можно обратить внимание, что опять же User-Agent по умолчанию есть какой-то. Go-http-client – это User-Agent самого клиента, вы можете туда подставить что угодно свое. И таким образом сделать методы, которые у вас работают с http.

Вот, как-то, на самом деле, довольно кратко, без вдавания в детали, подробности самого протокола http. Мы чуть глубже погрузимся в следующих лекциях, где мы будем разбирать различные протоколы и форматы, а также писать свой http-сервер, уже содержащий логику.