Сети 2

13 марта 2026 г.

Цели разработки сетевых протоколов

У разных сетевых протоколов зачастую схожие цели проектирования: надежность (способность восстанавливаться после ошибок, сбоев или отказов); распределение ресурсов (совместное использование общего ограниченного ресурса); способность к развитию (поэтапное развертывание усовершенствованных версий протокола с течением времени); безопасность (защита сети от различных типов атак)

Надежность

Один из механизмов обнаружения ошибок в полученной информации использует коды для **выявления ошибок**. Полученная неверно информация может быть повторно передана до тех пор, пока не будет получена правильно.

Надежность

Другой механизм использует коды чтобы **исправлять ошибки**, при этом правильное сообщение восстанавливается из потенциально некорректных битов, которые были получены изначально.

Надежность

Оба этих механизма работают за счет добавления избыточной информации.

Надежность

Ещё одна проблема, связанная с надёжностью, — это поиск рабочего пути в сети. Часто между источником и пунктом назначения существует несколько путей, и в большой сети некоторые каналы связи или маршрутизаторы могут быть неисправны.

Надежность

Например, сеть в Берлине вышла из строя. Пакеты, отправленные из Лондона в Рим через Берлин, не пройдут, но вместо этого можно отправить пакеты из Лондона в Рим через Париж. Сеть должна принимать подобные решения автоматически. Это называется **маршрутизацией** (routing).

Распределение ресурсов

Архитектуры, сохраняющие должную работоспособность сети при ее росте, называются **масштабируемыми** (scalable)

Распределение ресурсов

Статистическим мультиплексированием (statistical multiplexing) — ресурсы распределяются в соответствии со статистикой запросов на них.

Распределение ресурсов

Проблема, возникающая на всех уровнях: как не допустить, чтобы быстрый отправитель перегрузил данными медленного получателя. Для ее решения часто используется обратная связь от получателя отправителю. Это называется управлением потоком (flow control).

Эволюционная способность

В последнее время применяется ключевой механизм поддержки изменений, работающий за счет разбиения общей задачи на составные части и сокрытия нюансов реализации, — **разделение протокола на уровни** (protocol layering).

Способность к развитию

Поскольку в сети много компьютеров, каждому уровню необходим механизм идентификации отправителей и получателей, участвующих в конкретном сообщении. Этот механизм называется **адресацией** или **именованием** на нижнем и верхнем уровнях соответственно.

Способность к развитию

Еще одна проблема — несовпадение максимального размера сообщений, которые могут быть переданы в разных сетях. Вследствие этого приходится создавать механизмы для разбиения сообщений на части, передачи и последующей их сборки. Это называется организацией межсетевого взаимодействия (internetworking).

Безопасность

Одна из угроз – перехват данных. От этой угрозы защищают механизмы обеспечения (confidentiality), используемые на многих уровнях.

Безопасность

Также существуют механизмы **аутентификации** (authentication), гарантирующие, что никто не сможет выдать себя за кого-то другого.

Безопасность

Механизмы обеспечения **целостности** (integrity) предотвращают внесение скрытых изменений в сообщения, например, замену сообщения "Списать с моего счета $10"на "Списать с моего счета $1000".

Безопасность

Все эти возможности основаны на криптографии.

Протоколы

Для уменьшения сложности конструкции большинство сетей организованы в виде стека из слоев или уровней, каждый из которых построен над нижележащим.

Протоколы

Цель каждого уровня — предоставлять определенные услуги вышестоящим уровням, скрывая при этом от этих уровней детали фактической реализации предлагаемых услуг.

Протоколы

В некотором смысле каждый слой представляет собой своего рода виртуальную машину, предоставляющую определенные услуги слою, расположенному выше.

Протоколы

Когда уровень n на одной машине ведет диалог с уровнем n на другой машине, правила и соглашения, используемые в этом диалоге, в совокупности известны как протокол уровня n.

Протокол

По сути, протокол — это соглашение между участвующими сторонами о том, как должна осуществляться коммуникация.

Уровни, протоколы, интерфейсы

Пиры

Сущности, образующие соответствующие уровни на различных компьютерах, называются пирами (peers).

Пиры

Пиры могут быть программными процессами, аппаратными устройствами и даже людьми. Именно они взаимодействуют друг с другом с помощью протокола.

Интерфейсы

Данные не передаются напрямую с уровня n одного устройства на уровень n другого. Вместо этого каждый уровень отправляет данные и управляющую информацию на уровень, лежащий непосредственно под ним, пока не будет достигнут самый низший уровень.

Физическая среда

Под уровнем 1 располагается физическая среда (physical medium), через которую происходит фактический обмен информацией.

Интерфейсы

Каждая пара смежных уровней связана интерфейсом. Он определяет, какие базовые операции и службы предоставляет низший уровень высшему.

Интерфейсы

Четко заданные интерфейсы упрощают перевод уровня на совершенно другой протокол или реализацию.

Интерфейсы

Например, можно представить себе замену всех телефонных линий спутниковыми каналами. Это потребует новых протоколов и реализации, но услуги, предоставляемые верхнему уровню, останутся прежними.

Интерфейсы

Хосты нередко используют разные реализации одного протокола (зачастую написанные различными компаниями). На самом деле протокол на каком-либо уровне может поменяться совершенно незаметно для уровней над и под ним.

Интерфейсы

Каждый уровень должен выполнять конкретный набор четко определенных функций.

Архитектура сети

Набор уровней и протоколов называется **архитектурой сети**.

Архитектура

Спецификация архитектуры должна включать достаточное количество информации. Тогда тот, кто будет заниматься внедрением, сможет написать программу или создать аппаратное обеспечение для каждого уровня, должным образом следующие соответствующему протоколу.

Архитектура

Впрочем, ни нюансы реализации, ни спецификация интерфейсов не являются составными частями архитектуры, поскольку они скрыты внутри устройств и не видны извне.

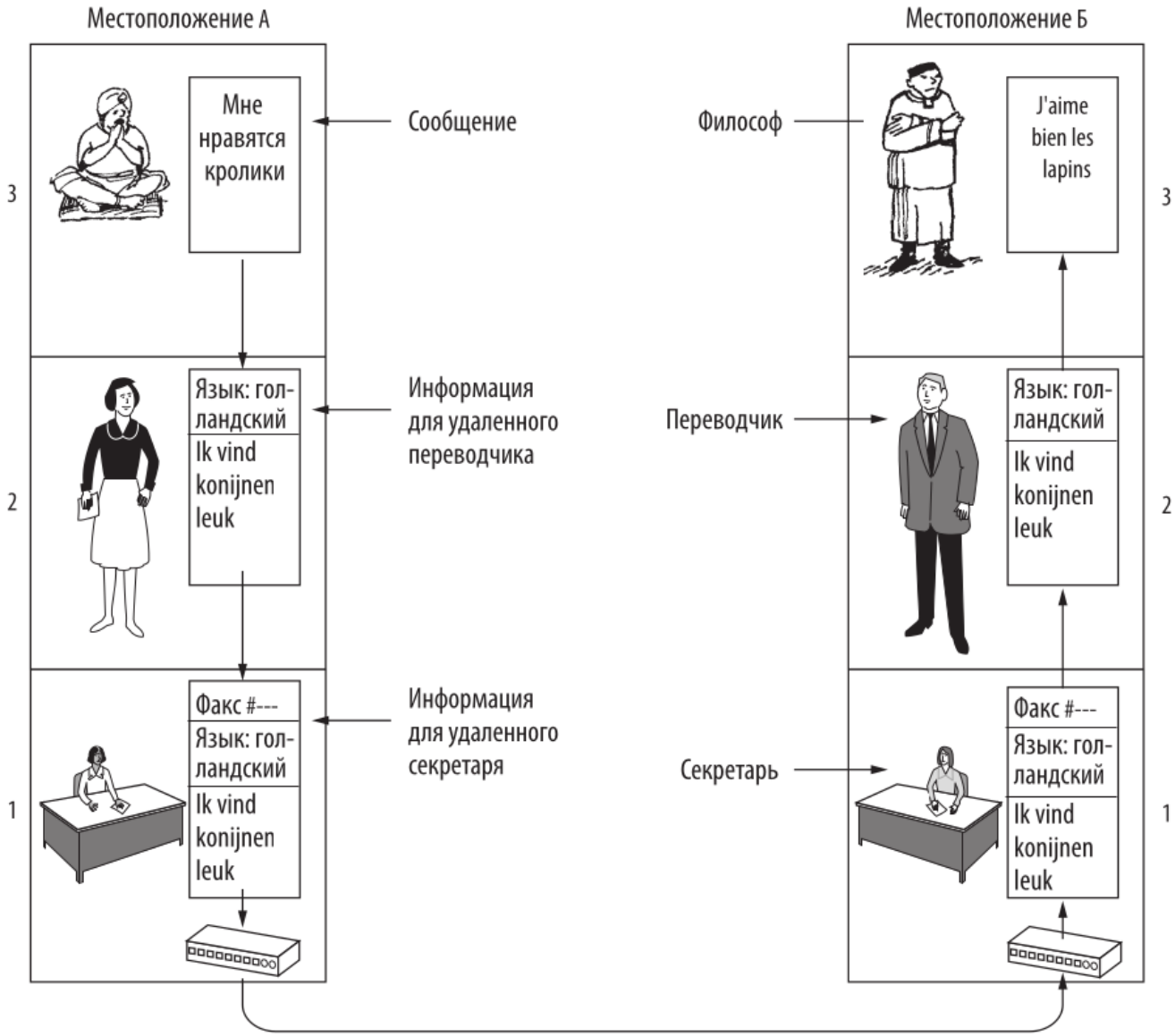
Архитектура

Интерфейсы на всех компьютерах сети даже могут быть разными, главное, чтобы каждый компьютер мог правильно использовать все протоколы.

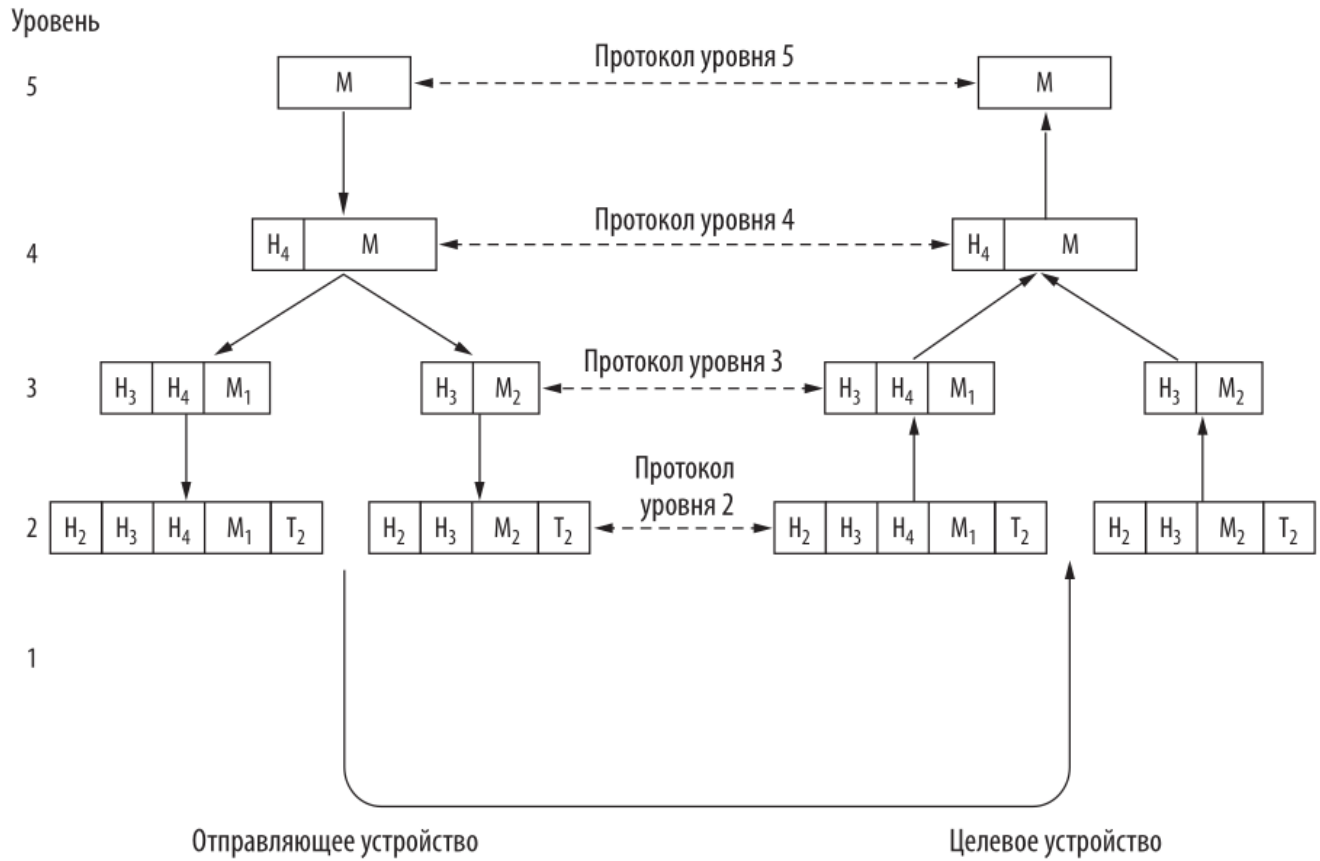
Стек протоколов

Список используемых конкретной системой протоколов, по одному на уровень, называется **стеком протоколов** (protocol stack).

Архитектура



Поток информации осуществляющей виртуальное общение на 5 уровне



Связи и надежность

Слои предоставляют два типа услуг вышележащим слоям: с **соединением** и без.

службы ориентированные на установление соединения

Сервис, **ориентированный на установление соединения**, создан по образцу телефонной системы.

службы ориентированные на установление соединения

Службы, **ориентированные на установление соединения** (connection-oriented service) сначала устанавливают соединение, затем используют и, наконец, освобождают его. Соединение ведет себя подобно трубе: отправитель вставляет объекты (биты) с одного ее конца, а получатель вытаскивает их с другого.

службы ориентированные на установление соединения

В некоторых случаях при установлении соединения отправитель, получатель и подсеть согласовывают параметры, которые будут использоваться, такие как максимальный размер сообщения, требуемое качество обслуживания и другие вопросы.

службы ориентированные на установление соединения

Канал связи (линия связи) – это другое название соединения с ассоциированными ресурсами, такими как фиксированная пропускная способность.

Службы без установления соединения

В противоположность службам, ориентированным на установление соединения, службы **без установления соединений** (connectionless) строятся по принципу работы обычной почты. Каждое сообщение (письмо) содержит полный адрес назначения и проходит через промежуточные узлы внутри системы независимо от всех последующих сообщений.

Службы без установления соединения

Вариант, при котором промежуточные узлы получают сообщение полностью, прежде чем отправлять его следующему узлу, называется **коммутацией с промежуточным хранением данных** (store-and-forward switching).

Службы без установления соединения

Альтернативный вариант, при котором дальнейшая передача сообщения узлом начинается до того, как оно будет полностью принято узлом, называется **сквозной коммутацией**.

Службы без установления соединения

Когда два сообщения отправляются в один пункт назначения, то обычно первое отправленное первым и прибывает. Впрочем, оно может задержаться в пути, тогда первым прибудет второе.

Службы без установления соединения

Устанавливать соединение нужно не для всех приложений. Например, спамеры отправляют нежелательную электронную почту сразу множеству адресатов.

Службы без установления соединения

Ненадежные (то есть без подтверждения получения) службы называются **службами отправки дейтаграмм** (datagram service), по аналогии с отправкой телеграмм, при которой отправитель тоже не получает уведомления о доставке.

Надежность

Службы, ориентированные на установление соединений и без них, можно охарактеризовать по их надежности. Некоторые сервисы надежны в том смысле, что они никогда не теряют данные.

Надежность

Как правило, надежная служба реализуется путем подтверждения получателем получения каждого сообщения, чтобы отправитель был уверен в его доставке.

Надежность

Процесс подтверждения приводит к накладным расходам и задержкам, которые зачастую оправданны, но иногда цена, которую необходимо заплатить за надежность, оказывается чрезмерной.

Надежность

Существуют два незначительно отличающихся варианта надежных служб с установлением соединения: последовательность сообщений (message sequences) и байтовые потоки (byte streams).

Надежность

В первом варианте сохраняются границы сообщений. Если было отправлено два сообщения по 1024 байта, они прибудут в место назначения в виде двух сообщений по 1024 байта, а не одного сообщения в 2048 байт.

Надежность

Во втором варианте соединение представляет собой просто поток байтов, без каких-либо границ сообщений. Когда в приемник поступает 2048 байт, нет никакой возможности определить, представляли ли они собой при отправке одно сообщение из 2048 байт, два сообщения по 1024 байта или 2048 сообщений по 1 байту.

Надежность

При отправке по сети страниц книги для фотонабора в виде отдельных сообщений сохранение границ может играть важную роль. С другой стороны, для скачивания фильма байтовый поток с сервера на компьютер пользователя — как раз то, что нужно. Границы сообщений (отдельных кадров) внутри фильма не имеют значения.

Надежность

В некоторых случаях неудобно создавать соединение для отправки одного-единственного сообщения, но важна надежность. Для подобных сценариев подой- дет служба отправки **дейтаграмм с подтверждением получения** (acknowledged datagram service). Она напоминает отправку заказного письма с уведомлением о вручении.

Надежность

Сама идея использования ненадежных коммуникаций может сначала привести в недоумение. В конце концов, как можно предпочесть ненадежную связь надежной?

Надежность

Прежде всего, надежная связь (в нашем понимании — с подтверждением получения) может оказаться недоступной на конкретном уровне. Например, Ethernet не обеспечивает надежного обмена данными. Пакеты периодически могут повреждаться в пути, а обеспечить их восстановление должны протоколы более высоких уровней. В частности, многие надежные службы строятся поверх ненадежных служб отправки дейтаграмм.

Надежность

Кроме того, присущие надежным службам задержки могут оказаться неприемлемыми, особенно для работающих в режиме реального времени приложений (например, мультимедийных). Поэтому и надежные, и ненадежные виды коммуникаций существуют параллельно.

Надежность

Для некоторых приложений неприемлемы транзитные задержки, возникающие вследствие подтверждения получения. Одно из этих приложений — цифровой голосовой трафик (VoIP). Возникающий время от времени небольшой шум на линии не так раздражает пользователей, как зависание разговора в ожидании подтверждений.

Надежность

Аналогично и при видеосвязи: небольшое число неправильно переданных пикселей не представляет собой проблемы, а вот подергивание изображения, когда поток данных останавливается для исправления ошибок, или долгое ожидание поступления идеального видеопотока раздражает пользователей.

Надежность

Еще один вид служб — **запрос/ответ** (request-reply). Отправитель передает одну дейтаграмму с запросом и получает дейтаграмму с ответом. С помощью схемы «запрос/ответ» часто реализуется связь в клиент-серверной модели: клиент отправляет запрос, а сервер на него реагирует.

Надежность

Например, клиент с мобильного телефона отправляет запрос картографическому серверу, чтобы получить список ближайших китайских ресторанов, а сервер присылает ему этот список.

Типы служб

	Служба	Пример
Ориентированные на установление соединения	Надежный поток сообщений	Последовательность страниц
	Надежный байтовый поток	Скачивание фильма
	Ненадежное соединение	Передача голоса по IP
Без установления соединений	Ненадежная дейтаграмма	Нежелательная электронная почта
	Дейтаграмма с подтверждением	Обмен текстовыми сообщениями
	Запрос/ответ	Запрос базы данных

Примитивы служб

Службы формально описываются набором **примитивов** (primitives), то есть операций, доступных обращающимся к ним пользовательским процессам.

Примитивы служб

С помощью этих примитивов можно указать службе выполнить какое-либо действие или сообщить о действии, которое совершил объект на том же уровне.

Примитивы служб

Если стек протоколов располагается в операционной системе (как это обычно и бывает), примитивы представляют собой системные вызовы.

Примитивы служб

Набор доступных примитивов зависит от вида предоставляемой службы. Примитивы для ориентированных на соединения служб отличаются от примитивов служб без установления соединений.

Ориентированная на на установление соединения служба с 6 примитивами

Примитив	Значение
LISTEN (прослушивать)	Блокирующее ожидание входящего соединения
CONNECT (подключиться)	Установка соединения с находящимся в состоянии ожидания одноранговым процессом
ACCEPT (согласиться)	Принятие входящего соединения от однорангового процесса
RECEIVE (принять)	Блокирующее ожидание входящего сообщения
SEND (отправить)	Отправка сообщения одноранговому процессу
DISCONNECT (отключиться)	Завершение соединения

Примитивы служб

Эти примитивы можно использовать для механизма «запрос/ответ» в клиент-серверной среде. Для иллюстрации их работы покажем схему работы простого протокола, реализующего службу получения дейтаграмм с подтверждением.

Примитивы служб

Прежде всего сервер выполняет примитив LISTEN, чтобы оповестить о готовности к приему входящих соединений. Примитив LISTEN обычно реализуют при помощи блокирующего системного вызова. После выполнения этого примитива серверный процесс блокируется (приостанавливается) до тех пор, пока не появится запрос на соединение.

Примитивы служб

Далее клиентский процесс выполняет CONNECT, чтобы установить соединение с сервером. В вызове CONNECT должно быть указано, с кем соединяться, поэтому у него может быть параметр для адреса сервера. После этого операционная система отправляет пакет одноуровневому процессу с запросом на соединение. Клиентский процесс приостанавливается до получения ответа.

взаимодействие типа «клиент-сервер» получения дейтаграмм с подтверждением

Примитивы служб

Когда пакет прибывает на сервер, операционная система видит, что он запрашивает соединение. Она проверяет наличие прослушивающего процесса и разблокирует его, если он есть. Далее серверный процесс может устанавливать соединение при помощи вызова ACCEPT, в результате которого клиентскому процессу отправляется ответ (2) с согласием на соединение.

Примитивы служб

Поступление этого ответа приводит к снятию блокировки с клиента. И сервер, и клиент в этот момент уже работают, и между ними установлено соединение.

Примитивы служб

Следующий шаг: выполнение сервером операции RECEIVE для подготовки к получению первого запроса. Обычно сервер делает это сразу же после снятия блокировки примитивом LISTEN, прежде чем клиент получит подтверждение. Вызов RECEIVE блокирует сервер.

Примитивы служб

Далее клиент выполняет примитив SEND для передачи своего запроса (3) с последующим RECEIVE для получения ответа. Поступление пакета с запросом разблокирует сервер, и он может обработать запрос.

Примитивы служб

После завершения необходимых действий сервер отправляет ответ клиенту (4) с помощью примитива SEND. Получение этого пакета разблокирует клиента, который теперь может обработать ответ и отправить дополнительные запросы, если таковые у него есть.

Примитивы служб

По окончании работы с сервером клиент выполняет DISCONNECT для завершения соединения (5). Обычно первый DISCONNECT представляет собой блокирующий вызов, который приостанавливает клиента, а серверу отправляется пакет с сообщением, что соединение больше не нужно.

Примитивы служб

При получении этого пакета сервер также выполняет свою операцию DISCONNECT, подтверждая клиенту получение, и освобождает соединение (6).

Примитивы служб

При поступлении серверного пакета на компьютер пользователя клиентский процесс освобождается и соединение разрывается. Такова краткая схема работы связи с использованием соединений.

Службы и протоколы

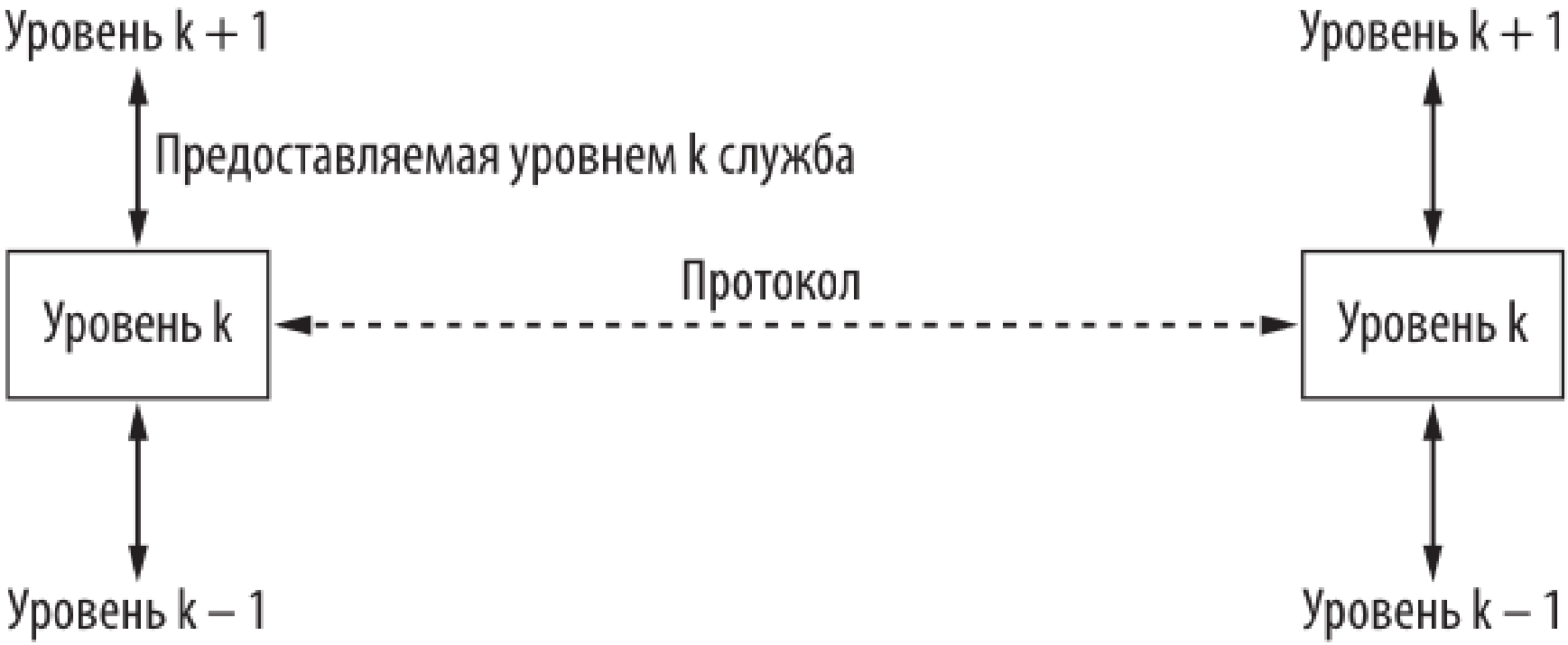
Службы — это набор примитивов (операций), которые нижележащий уровень может делать для вышележащего. Служба описывает операции, которые уровень может выполнять для своих пользователей, но при этом не упоминается о том, как эти операции реализуются. Служба описывает интерфейс между двумя уровнями, один из которых (расположенный ниже) является поставщиком службы, а другой (расположенный непосредственно над первым) — ее потребителем.

Службы и протоколы

Протокол, напротив, представляет собой набор правил, определяющих формат и смысл пакетов (сообщений), которыми обмениваются сущности внутри одного уровня. Протоколы используются такими сущностями для реализации описаний служб.

Службы и протоколы

Службы имеют непосредственное отношение к интерфейсам между уровнями. Протоколы же имеют непосредственное отношение к пакетам, пересылаемым между одноранговыми сущностями на различных компьютерах.



Службы и протоколы

Уместно будет провести аналогию с языками программирования. Служба подобна абстрактному типу данных или объекту в объектно-ориентированном языке. Она описывает, какие операции можно выполнять над объектом, но не уточняет, как они должны быть реализованы. А протокол относится к реализации службы и сам по себе пользователю службы не виден.

Службы и протоколы

Службы и протоколы

Службы и протоколы

Службы и протоколы

Службы и протоколы