**AMQP (Advanced Message Queuing Protocol)** — открытый [протокол](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [прикладного уровня](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D1%8B_%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D1%8F) для передачи сообщений между компонентами системы. Основная идея состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет [маршрутизацию](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений.

Архитектуру протокола разработал John O’Hara из банка [JP Morgan Chase & Co](https://ru.wikipedia.org/wiki/JPMorgan_Chase)[[1]](https://ru.wikipedia.org/wiki/AMQP#cite_note-1).

**AMQP основан на трёх понятиях:**

1. Сообщение (message) — единица передаваемых данных, основная его часть (содержание) никак не интерпретируется сервером, к сообщению могут быть присоединены структурированные заголовки.
2. Точка обмена (exchange) — в неё отправляются сообщения. Точка обмена распределяет сообщения в одну или несколько [очередей](https://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). При этом в точке обмена сообщения не хранятся. Точки обмена бывают трёх типов:
   * fanout — сообщение передаётся во все прицепленные к ней очереди;
   * direct — сообщение передаётся в очередь с именем, совпадающим с ключом маршрутизации (routing key) (ключ маршрутизации указывается при отправке сообщения);
   * topic — нечто среднее между fanout и direct, сообщение передаётся в очереди, для которых совпадает маска на ключ маршрутизации, например, app.notification.sms.# — в очередь будут доставлены все сообщения, отправленные с ключами, начинающимися с app.notification.sms.
3. Очередь (queue) — здесь хранятся сообщения до тех пор, пока не будут забраны клиентом. Клиент всегда забирает сообщения из одной или нескольких очередей.

**Протокол можно разделить на два слоя:**

1. Functional Layer - определяет набор команд которые выполняют работу от имени приложения.
2. Transport Layer - обслуживает запросы приложения к серверу и сервера к приложению, управляет мультиплексированием каналов, фреймингом, кодировкой, heart-beating, представлением данныx, работой с ошибками.

**Примеры очередей:**

* **store-and-forward-queue** - хранит и распространяет сообщения для многих потребителей (consumers) по принципу [round robin](https://ru.wikipedia.org/wiki/Round-robin_(%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)). Постоянные, публичные
* **private reply queue** - хранит и распространяет сообщения для одного потребителя (consumer). Временные, приватные
* **private subscription queue** - хранит сообщения от многих источников и отправляет одному потребителю (consumer). Временные, приватные

Протокол не ограничивается этими тремя видами. Они указаны как пример реализации.

Терминология[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=2) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=2)]

* **Exchange** - сущность которая получает сообщения от приложений и при необходимости перенаправляет их в очереди сообщений.
* **Binding**: отношение между очередью сообщений и точками обмена.
* **Routing key**: виртуальный адрес который точка обмена использует для принятия решения о дальнейшей маршрутизации.

Exchange[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=3) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=3)]

Принимает сообщения от поставщика и направляет их в message queue в соответствии с предопределёнными критериями. Такие критерии называют bindings. Exchange - механизм согласования и маршрутизации сообщений. На основе сообщений и их параметров (bindings) принимают решение о перенаправлении в очередь или другой exchange. Не хранят сообщения.

Термин exchange означает алгоритм и экземпляр алгоритма. Также говорят exchange type и exchange instance.

AMQP определяет набор стандартных типов exchange. Приложения могут создавать свои exchange instance.

Каждый exchange реализует свой алгоритм маршрутизации. Существует несколько стандартных типов exchange, описанных в Functional Specification стандарта. Из них два являются важными:

* Direct exchange - маршрутизирует на основе routing key. Базовый exchange - это direct exchange
* topic exchange - маршрутизирует на основе шаблона маршрутизации.

Сервер создаст несколько exchange, включая direct и topic. Они будут иметь wellknown имена и клиентские приложения смогут работать с ними.

Жизненный цикл Exchange[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=4) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=4)]

Каждый AMQP-сервер предсоздаёт несколько экземпляров exchange. Эти экземпляры существуют когда сервер запущен и не могут быть уничтожены. AMQP-приложения могут также создавать свои собственные exchange. AMQP не использует для этого метод create, вместо этого экземпляр декларируется, который исходит из логики: "создать, если не создано, иначе продолжить". Можно сказать, что создание exchange [идемпотентно](https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BC%D0%BF%D0%BE%D1%82%D0%B5%D0%BD%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C). Вероятно приложения будут создавать exchange по мере необходимости, а затем уничтожать их за ненадобностью. AMQP предоставляет метод для уничтожения exchange.

Routing Key[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=5) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=5)]

В общем случае exchange проверяет свойства сообщения, поля заголовка и содержимое его тела и, используя эти и, возможно, данные из других источников, решает, как направить сообщение. В большинстве простых случаев exchange рассматривает одно ключевое поле, которое мы называем **Routing Key**. **Routing Key** - это виртуальный адрес, который сервер exchange может использовать для принятия решения о направлении сообщения. Для маршрутизации типа **point-to-point** ключом маршрутизации обычно является имя очереди сообщений. Для маршрутизации [**pub-sub**](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_%E2%80%94_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%87%D0%B8%D0%BA) ключ маршрутизации обычно является значением иерархии топика (topic - смотри publication/subscruber). В более сложных случаях ключ маршрутизации может быть объединен с маршрутизацией по полям заголовка сообщения и/или его содержанием

Message Queue (очередь сообщений)[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=6) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=6)]

Когда клиентское приложение создаёт очередь сообщений - оно может указать следующие свойства:

* **name** - если не указано, сервер сам выбирает имя и отправляет его клиенту. Как правило, когда приложения совместно используют очередь сообщений они заранее договариваются об имени очереди сообщений, и когда приложение нуждается в очереди сообщений для своих собственных целей, оно позволяет серверу предоставлять имя.
* **exclusive** - если этот параметр установлен, то очередь существует пока существует текущее соединение. Очередь удаляется при разрыве подключения.
* **durable** - если установлен - очередь существует и активна при перезагрузке сервера. Очередь может потерять сообщения посланные во время перезагрузки сервера

Жизненный цикл сообщения[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=7) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=7)]

Сообщение AMQP состоит из набора свойств и непубличного содержимого. Новое сообщение создается **producer** с использованием клиентского API AMQP. **producer** добавляет контент в сообщение и, возможно, устанавливает некоторые свойства сообщения. **producer** маркирует сообщение с помощью маршрутной информации, которая внешне похожа на адрес, но может быть любой. Затем **producer** отправляет сообщение в **exchange**. Когда сообщение поступает на сервер, exchange (обычно) направляет его в набор очередей, которые также существуют на сервере. Если сообщение немаршрутизируемо, **exchange** может удалить его или вернуть приложению. **producer** сам решает как поступать с немаршрутизируемыми сообщениями.

Одно сообщение может существовать во многих очередях сообщений. Сервер может справиться с этим по-разному, например: копирование сообщения с помощью подсчета ссылок и т. д. Это не влияет на интероперабельность. Однако, когда сообщение направляется в несколько очередей сообщений, оно идентично в каждой очереди сообщений. Здесь нет уникального идентификатора, отличающего различные копии.

Когда сообщение поступает в очередь сообщений, она немедленно пытается передать его потребителю через AMQP. Если это невозможно, то сообщение хранится в очереди сообщений (в памяти или на диске по просьбе **producer** ) и ждет, пока **consumer** будет готов. Если отсутствует **consumer**, то очередь может вернуть сообщение **producer** через AMQP (опять же, если **producer** попросил об этом).

Когда очередь сообщений может доставить сообщение **consumer**, она удаляет сообщение из своего внутреннего хранилища. Это может произойти сразу же или после того, как **consumer** признает, что он успешно выполнил свою работу, обработал сообщение. **consumer** сам выбирает, как и когда сообщения будут "подтверждены". **consumer** также может отклонить сообщение (отрицательное подтверждение).

Сообщения **producer** и подтверждения **consumer** сгруппированы в транзакции. Когда приложение играет обе роли, что часто бывает, он выполняет смешанную работу: отправляет сообщения и отправляет подтверждения,а затем фиксация или откат транзакции.

Доставка сообщений от сервера к **consumer** не является транзакционной.

Producer[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=8) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=8)]

**Producer** - клиентское приложение, которое публикует сообщения в **exchange**.

По аналогии с устройством электронной почты, можно заметить, что **producer** не отправляет сообщения непосредственно в очередь (message queue). Иное поведение нарушило бы абстракцию в модели AMQ. Это было бы похоже на жизненный цикл сообщения электронной почты: разрешение электронной почты, обход таблиц маршрутизации MTA и попадание непосредственно в почтовый ящик. Это сделало бы невозможной вставку промежуточной фильтрации и обработки, например, обнаружение спама.

Модель AMQ использует тот же принцип, что и система электронной почты: все сообщения отправляются в одну точку **exchange** или **MTA**, который проверяет сообщения на основе правил и информации, которая скрыта от отправителя, и направляет их к точкам распространения, которые также скрыты от отправителя. (and routes them to drop-off points that are also hidden from the sender - здесь точки распространения это drop-off points из документации).

Consumer[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=9) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=9)]

**Consumer** - клиентское приложение которое получает сообщения из очереди сообщений.

Наша аналогия с электронной почтой начинает разрушаться, когда мы смотрим на **consumer** (получателей). Почтовые клиенты пассивны - они могут читать почтовые ящики, но они не оказывают никакого влияния на то, как эти почтовые ящики заполняются. С помощью AMQP **consumer** также может быть пассивным, как и почтовые клиенты. То есть мы можем написать приложение, которое прослушивает определённую очередь сообщений и просто обрабатывает поступающую информацию. При этом очередь сообщений должна быть готова до старта приложения и должна быть "привязана" к нему.

Также consumer имеет следующие возможности:

* создавать/удалять очереди сообщений
* определять способ заполнений очереди используя **bindings**
* выбирать разные exchanges, что может полностью изменить семантику маршрутизации

Это как обладать почтовой системой, которая на уровне протокола может:

* создать новый почтовый ящик
* сказать MTA, что все сообщения с определённым заголовком должны быть скопированы в этот ящик
* полностью менять то, как именно почтовая система интерпретирует адреса и другие заголовки сообщения.

Автоматический режим[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=10) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=10)]

Большинство интеграционных архитектур не нуждаются в таком уровне сложности. Большинство пользователей AMQP нуждаются в базовой функциональности из коробки. AMQP обеспечивает это следующим образом:

* базовый (default) exchange для отправителей (producers) сообщений;
* базовый binding для очередей сообщений, который сортирует сообщения на основе совпадения routing key и имени очереди сообщений

В результате базовый binding позволяет producer отправлять сообщения напрямую в очередь сообщений, таким образом он эмулирует простейшую схему отправки сообщения получателю, которую люди ожидают от традиционного middleware.

Базовый binding не препятствует использованию очереди сообщений в более сложных конструкциях. Он позволяет использовать AMQP без конкретного понимания работы механизмов binding и exchange.

AMQP Command Architecture[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=11) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=11)]

Секция описывает процесс взаимодействия приложения и сервера

**Команды протокола (классы и методы)**[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=12) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=12)]

Промежуточное программное обеспечение является сложным, и при разработке структуры протокола его создатели попытались укротить эту сложность. Их подход состоял в том, чтобы смоделировать традиционный API, основанный на классах, которые содержат методы, при этом каждый метод должен делать ровно одну вещь и делать это хорошо. Это приводит к большому набору команд, но тот, который есть относительно легко понять.

Команды AMQP сгруппированы в классы. Каждый класс охватывает определенную функциональную область. Некоторые классы являются необязательными - каждый одноранговый узел реализует классы, которые он должен поддерживать.

Существует два различных метода диалога:

* **Synchronous request-response**, в котором один одноранговый узел отправляет запрос, а другой одноранговый узел отправляет ответ. Синхронные методы запроса и ответа используются для функциональных возможностей, которые не являются критичными для производительности.
* **Asynchronous notifications**, в котором один одноранговый узел отправляет данные, но не ожидает ответа. Асинхронные методы используются там, где производительность имеет решающее значение.

Чтобы упростить обработку метода, мы определяем отдельные ответы для каждого синхронного запроса. То есть, один метод не используется для ответа на два разных запроса. Это означает, что одноранговый узел, отправляя синхронный запрос, может принимать и обрабатывать входящие методы до получения одного из допустимых синхронных ответов. Это отличает AMQP от более традиционных протоколов RPC.

Метод формально определяется как синхронный запрос, синхронный ответ (на конкретный запрос) или асинхронный. Наконец, каждый метод формально определяется как клиентская сторона (т. е. сервер-клиент) или серверная сторона (клиент-сервер).

### Класс Connection[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=15) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=15)]

Соединение реализовано, чтобы быть долговечным и обрабатывать множество каналов.

#### Жизненный цикл соединения**[**[**править**](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=16)**|**[**править код**](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=16)**]**

* Клиент открывает соединение TCP/IP к серверу и отправляет заголовок протокола. Это единственная доступная для отправки клиентом информация, которая не форматирована как метод.
* Сервер отвечает версией протокола и другими свойствами, включая список механизмов безопасности которые он поддерживает (the Start method)
* Клиент выбирает механизм обеспечения безопасности (Start-Ok).
* Сервер инициирует процесс аутентификации который использует модель [SASL](https://ru.wikipedia.org/wiki/Simple_Authentication_and_Security_Layer), он отправляет клиенту challenge (Secure).
* Клиент отправляет authentication response (Secure-Ok). Например, используя механизм аутентификации 'plain' response, содержит имя и пароль.
* Сервер повторяет challenge (Secure) или переходит к переговорам, отправляя набор параметров, в числе которых maximum frame size (Tune).
* Клиент принимает или занижает эти параметры (Tune-Ok).
* Клиент формально открывает соединение и выбирает виртуальный хост (Open).
* Сервер подтверждает выбор виртуального хоста (Open-Ok).
* Теперь клиент использует соединение по своему усмотрению.
* Один узел (клиент или сервер) закрывает соединение (Close).
* Другой узел отправляет данные о закрытии подключения (Close-Ok).
* Сервер и клиент закрывают соответствующие соединению сокеты.

Обмен информацией для ошибок неполностью открытых соединений не производится. Узел, который обнаружил ошибку, **должен** закрыть сокет без дополнительных уведомлений.

### Класс Channel[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=17) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=17)]

AMQP-это многоканальный протокол. Каналы обеспечивают возможность мультиплексирования тяжелого TCP/IP-соединения в несколько легких соединений. Это делает протокол более "дружественным к брандмауэру", поскольку использование портов предсказуемо. Это также означает, что формирование трафика и другие функции QoS сети могут быть легко использованы.

Каналы независимы друг от друга и могут выполнять различные функции одновременно с другими каналами, при этом доступная полоса пропускания разделяется между конкурентными задачами.

Ожидается и поощряется, что многопоточные клиентские приложения могут часто использовать модель “channel-per-thread” для удобства разработки. Однако открытие нескольких соединений с одним или несколькими серверами AMQP от одного клиента также вполне приемлемо. Жизненный цикл канала таков:

* Клиент открывает новый канал (Open)
* Сервер подтверждает открытие канала (Open-Ok)
* Клиент и сервер используют канал по своему усмотрению
* Один из узлов (клиент или сервер) закрывает канал (Close)
* Другой узел подтверждает закрытие канала (Close-Ok)

### Класс Exchange[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=18) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=18)]

Позволяет приложению управлять экземплярами exchange на сервере. Этот класс позволяет приложению писать свой собственный сценарий обработки сообщений, не полагаясь на какую-либо конфигурацию.

Примечание: большинство приложений не нуждаются в таком уровне сложности, и устаревшее промежуточное программное обеспечение вряд ли сможет поддерживать эту семантику.

#### Жизненный цикл Exchange**[**[**править**](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=19)**|**[**править код**](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=19)**]**

* Клиент просит сервер убедиться, что exchange существует (Declare). Клиент может уточнить это следующим образом: " создайте exchange, если он не существует "или" предупредите меня, но не создавайте его, если он не существует".
* Клиент публикует сообщения в exchange
* Клиент может решить удалить exchange (Delete)

### Класс Queue[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=20) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=20)]

Класс queue позволяет приложению управлять очередями сообщений на сервере. Это основной шаг почти во всех приложениях, которые получают сообщения, по крайней мере для проверки того, что ожидаемая очередь сообщений действительно существует.

#### Жизненный цикл очереди**[**[**править**](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=21)**|**[**править код**](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=21)**]**

Протокол предусматривает два жизненных цикла очереди:

* **Durable message queues** - используются несколькими потребителями и существуют независимо от наличия потребителей которые могли бы принимать сообщения
* **Temporary message queues** - приватные очереди для конкретного потребителя. Очередь удаляется при отсутствии потребителей.

#### Жизненный цикл для durable message queue**[**[**править**](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=22)**|**[**править код**](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=22)**]**

* Клиент объявляет очередь сообщений (Declare с аргументом "passive")
* Сервер подтверждает существование очереди (Declare-Ok)
* Клиент читает сообщения из очереди

#### Жизненный цикл для temporary message queues**[**[**править**](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=23)**|**[**править код**](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=23)**]**

* Клиент создаёт очередь сообщений (Declare часто без имени очереди, так что сервер сам даст ей имя). Сервер подтверждает создание (Declare-Ok)
* Клиент инициализирует consumer для созданной очереди.
* Клиент останавливает consumer либо явно, либо путем закрытия канала и / или соединения
* Когда последний consumer исчезает из очереди сообщений и после вежливого тайм-аута сервер удаляет очередь сообщений

AMQP реализует механизм подписок на темы в виде очередей сообщений. Это позволяет создавать интересные структуры, в которых подписка может быть сбалансирована по нагрузке между пулом совместно работающих абонентских приложений.

#### Жизненный цикл подписки**[**[**править**](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=24)**|**[**править код**](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=24)**]**

* Клиент создаёт очередь сообщений (Declare), сервер подтверждает (Declare-Ok)
* Клиент сопоставляет очередь сообщений с exchange темы (Bind) и сервер подтверждает сопоставление (Bind-Ok)
* Клиент использует очередь сообщений так как описано выше

### Класс Basic[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=25) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=25)]

Базовый класс реализует возможности обмена сообщениями, описанные в этой спецификации. Он поддерживает следующую семантику:

* Отправка сообщений от клиента - серверу которая происходит асинхронно (Publish)
* Запуск и остановка consumers (Consume, Cancel)
* Отправка сообщений с сервера на клиент, которая происходит асинхронно (Deliver, Return)
* Подтверждение сообщений (Ack, Reject)
* Получение сообщений из очереди синхронным способом (Get)

### Класс Transaction[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=26) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=26)]

AMQP поддерживает два вида транзакций:

1. Автоматические транзакции, в которых каждое опубликованное сообщение и подтверждение обрабатывается как автономная транзакция.
2. Локальные серверные транзакции, в которых сервер заносит в буфер опубликованные сообщения и подтверждения и фиксирует (commit) их по требованию клиента.

Класс Transaction (“tx”) даёт приложениям доступ ко второму типу транзакций, локальным транзакциям сервера. Семантика класса следующая:

1. Приложение запрашивает серверные транзакции в каждом канале в котором оно хочет получить такие транзакции (Select)
2. Приложение выполняет работу (Publish, Ack)
3. Приложение выполняет commit или rollback работы (Commit, Roll-back)
4. Приложение продолжает работу

Транзакции занимаются публикацией контента и подтвеждениями, ***не*** доставкой. Таким образом rollback не помещает в очередь повторно и не инициирует повторную доставку. Клиент может подтвердить эти сообщения в следующей транзакции.

## Транспортная архитектура AMQP[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=27) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=27)]

В этом разделе объясняется, как команды сопоставляются с [wire-level](https://en.wikipedia.org/wiki/Wire_protocol) протоколом.

### Описание[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=28) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=28)]

AMQP - это двоичный протокол. Информация организуется в фреймы различных типов. Фреймы содержат методы протокола и другую информацию. Все кадры имеют один и тот же общий формат: заголовок кадра, полезная нагрузка и конец кадра. Формат полезной нагрузки кадра зависит от типа кадра.

На транспортном уровне предполагается использование стека TCP/IP или аналогов.

В пределах одного сокетного соединения может существовать несколько независимых потоков управления, называемых каналами. Каждый кадр пронумерован номером канала. Чередуя свои кадры, различные каналы совместно используют это соединение. Для любого данного канала кадры выполняются в строгой последовательности, которая может использоваться для управления анализатором протоколов (обычно это state machine).

Мы строим фреймы, используя небольшой набор типов данных, таких как биты, целые числа, строки и таблицы полей. Поля фрейма плотно упакованы, не делая их медленными или сложными для разбора. Относительно просто создать слой кадрирования механически из спецификаций протокола.

[Wire-level](https://en.wikipedia.org/wiki/Wire_protocol) форматирование разработано таким образом, чтобы быть масштабируемым и достаточно универсальным для использования в произвольных протоколах высокого уровня (а не только в AMQP). Мы предполагаем, что AMQP будет расширяться, улучшаться и иным образом изменяться с течением времени, и wire-level формат будет поддерживать это.

### Типы данных[[править](https://ru.wikipedia.org/w/index.php?title=AMQP&veaction=edit&section=29) | [править код](https://ru.wikipedia.org/w/index.php?title=AMQP&action=edit&section=29)]

Типы данных AMQP используемые в фреймах:

* Целые числа Integers (от 1 до 8 октетов), используются для представления размеров, величины, лимиты и т.д. Целые числа всегда беззнаковые и могут быть невыровнены в кадре.
* Биты
* Короткие строки, используемые для хранения свойств короткого текста. Короткие строки ограничены 255 октетами и могут быть проанализированы без риска переполнения буфера. (Я подозреваю, что речь об одном октете в 255 состояний а не о 255 октетах)
* Длинные строки, используемые для хранения частей двоичных данных
* Поля таблиц, содержащие пары имя-значение. Значения полей вводятся в виде строк, целых чисел и т. д