AMQP 1.0 and Qpid Dispatch

New possibilities with traditional messaging

Introduction

Introduction

- AMQP enthusiast
- Architect @ Risk IT product
- Representing Deutsche Boerse @ OASIS
- PMC Member @ Apache Qpid

- **All views in this talk are my own**

AMQP Protocol

History

- AMQP is a messaging protocol which was created together by companies from IT and financial industry
  - Compatibility between different competing implementation was one of the main goals
- Several versions were published by the AMQP Working Group
  - 0-8, 0-9, 0-9-1, 0-10
- Later, AMQP was taken under OASIS
  - OASIS is a non-profit consortium which drives standardization
  - Published AMQP version 1.0
- 1.0 is very different from the older versions 0-8, 0-9, 0-9-1 and 0-10

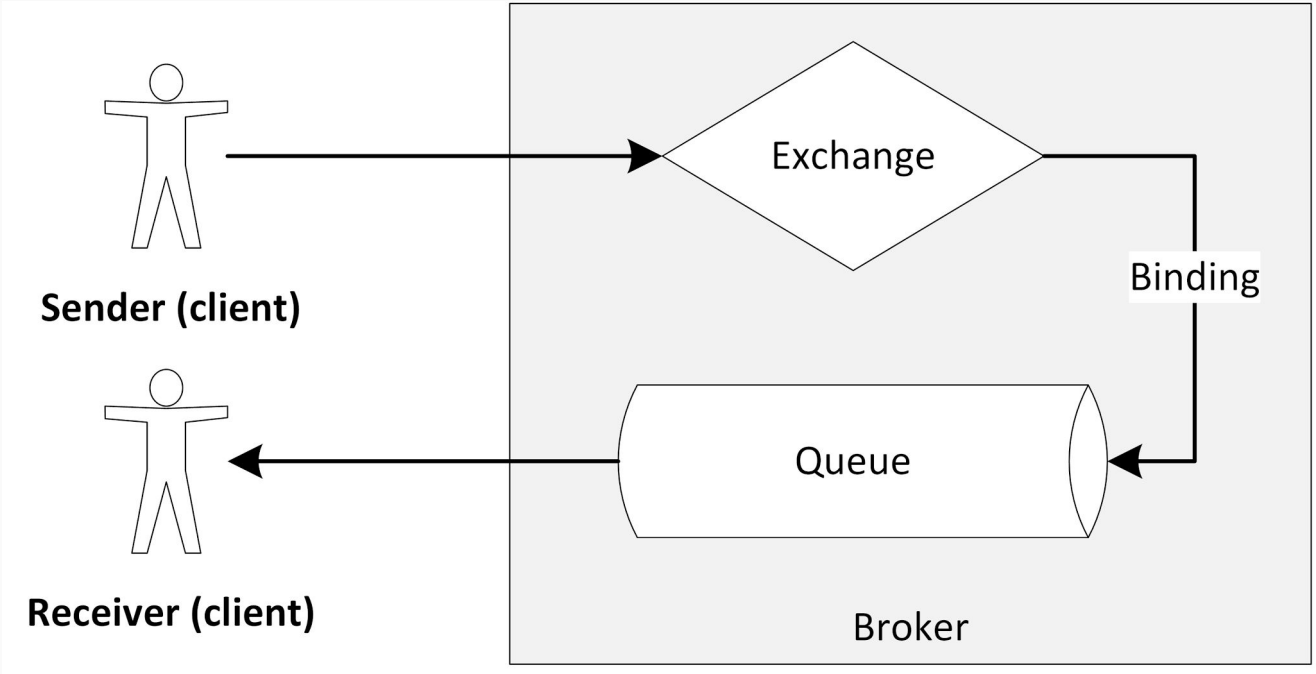
AMQP 0-8, 0-9, 0-9-1 and 0-10

- The older versions have different implementations
  - RabbitMQ implements 0-8, 0-9, 0-9-1
  - Apache Qpid C++ broker implements 0-10
  - Apache Qpid Java broker implements 0-8, 0-9, 0-9-1 and 0-10
  - Never achieved the idea many competing implementations
- Widely used in Deutsche Boerse
  - Energy is using RabbitMQ with 0-8, 0-9, 0-9-1
  - Clearing, Trading, Risk and some more are using 0-10 with Red Hat MRG-M (branded version of Qpid C++ broker)

AMQP 0-8, 0-9, 0-9-1 and 0-10

- Designed as asymmetric protocol
  - Distinguishes between two parties: CLIENT and BROKER
  - CLIENT cannot connect directly to another CLIENT
  - CLIENT can connect only to BROKER
  - BROKER cannot connect directly to another BROKER
  - BROKER can connect only accept connections from a CLIENT
- Defines the structure for the BROKER
  - EXCHANGE, BINDING, QUEUE
  - Including specific types and behaviour
  - Provides means to manage the structure

AMQP 0-8, 0-9, 0-9-1 and 0-10



AMQP 1.0

- Designed from scratch
- Symmetric protocol
  - CLIENT communicates with CLIENT
  - No BROKER exists in AMQP 1.0 specification
  - Since AMQP 1.0 doesn’t have the concept of BROKER, it also doesn’t define the structure
  - As a result AMQP 1.0 doesn’t have EXCHANGE, BINDING or QUEUE
- Easy to integrate into existing products
- Used outside of finance industry, for example in IoT
- Approved as ISO 19464

AMQP 1.0

AMQP 1.0

- Why is AMQP 1.0 the future?
  - It is not only about ISO standardization
  - AMQP is not only about finance anymore, it expanded into new areas like IoT
  - Wide adoption among IT companies
  - Finally achieved the idea of competing but compatible implementations
  - Clients available for many platforms and programming languages, often with multiple implementations
  - Additional work is still going on at OASIS
    - Link pairing, WebSocket and JMS bindings, Management, Addressing, Claims Based Security

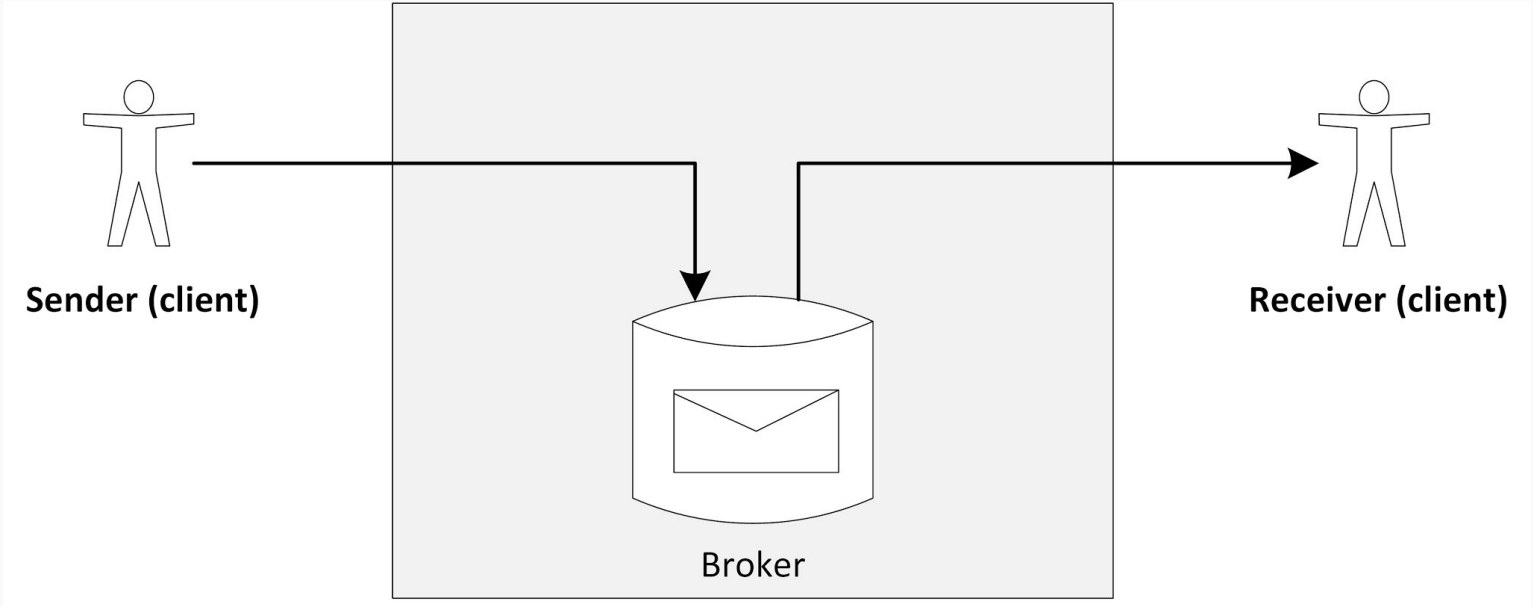
AMQP 1.0

- AMQP 1.0 clients
  - Qpid Proton, Qpid Messaging, Qpid JMS (Java, C, C++, Go, Python, Ruby, Perl, PHP)
  - Microsoft Azure AMQP, .NET Lite and uAMQP (C#, .NET, .NET Micro Framework, C)
  - node-amqp10 (Javascript)
  - Rhea (Javascript)
  - IBM MQ Light (Java, Ruby, Python, Javascript)
  - SwiftMQ (Java)
  - Vert.x (Polyglot)

Brokers

- Is brokered messaging dead?
  - **No!**
  - Brokered messaging provides decoupling or senders and receivers
  - Supports useful patterns like for example store and forward
  - But brokered messaging is only one particular use case
  - Broker isn’t suited for every communication
  - **Use brokers only where it makes sense!**

Brokers



Brokers

- Can brokers use AMQP 1.0?
  - BROKERS can still use AMQP 1.0
  - AMQP 1.0 gives broker much more freedom
  - It doesn’t enforce the EXCHANGE - BINDING - QUEUE schema
  - Makes it easy to support AMQP 1.0 in brokers with different schemas. For example JMS (Topic-Subscription, Queue) or WebSphere MQ.
  - Existing brokers based on EXCHANGE - BINDING - QUEUE schema can also easily support AMQP 1.0

Brokers

- Many brokers implement AMQP 1.0
  - Microsoft Azure Service Bus and Microsoft Azure Event Hubs
  - Apache Qpid C++ broker
  - Apache Qpid Java broker
  - Apache ActiveMQ, Apache ActiveMQ Apollo and Apache ActiveMQ Artemis
  - SwiftMQ
  - IBM WebSphere MQ Light and WebSphere MQ (since version 8)
  - RabbitMQ (through experimental plugin)

Brokers

- With freedom comes responsibility
  - Older versions of the AMQP protocol enforced EXCHANGE - BINDING - QUEUE schema
  - But they also provided management tooling around it (QueueDeclare, ExchangeDeclare, ...)
  - AMQP 1.0 doesn’t enforce the schema and doesn’t provide any tooling
  - Because of missing schema and tooling, it is not always trivial to create the broker resources using AMQP 1.0 protocol
  - For example: Create auto-delete queue with ring policy and max message count of 1000 messages and max size of 1MB and bind it to specific exchange with specific binding key was trivial with AMQP 0-10. It is complicated with 1.0.

Brokers

- What can we do with it?
  - AMQP Management protocol is *work in progress* at OASIS
    - Will provide standardized mechanisms for managing all kind of AMQP resources
    - Will not standardize the schema for brokers, but will give you standard mechanisms how to manage whatever schema the broker is using
    - First drafts are already implemented by some of the AMQP implementations
  - Use broker only where it makes sense
    - In the past we used these features on places where broker maybe isn't the right tool
    - AMQP 1.0 gives us better means how deal with many of these use cases

Beyond brokers

- If broker is not the solution, what is?
- Routers
  - Qpid Dispatch
  - SwiftMQ
- Bridges
  - Apache Kafka, Apache Spark
  - OpenMAMA, Sqlstream
  - Vert.x AMQP Bridge
- APIs

Apache Qpid Dispatch

Apache Qpid Dispatch

- Lightweight AMQP 1.0 router
- Written in C, based on Apache Qpid Proton library
- High performance, scalable interconnect between AMQP endpoints
- Supports both listeners as well as connectors
- Supports the latest draft of the AMQP Management specification
- Allows new use cases which were not possible before
- Routers can be connected into networks

Broker versus router

- Broker
  - Decouples the clients
    - Client always talks with the broker, not directly to another client
    - Settlements are between broker and client, not client and client
    - Clients should not care if the other client is online or not
  - Takes ownership of the message
  - Usually has a persistent / reliable storage
  - Often has complex internals for message queueing and routing

Broker versus router

- Router
  - Doesn’t decouple the clients
    - Messages are settled between the clients
    - Router doesn’t take ownership of the message
    - When receiving client is not online, message is not delivered
  - No reliable / persistent storage (since router doesn’t take ownership, it doesn’t need storage)
  - Simple and straight forward mechanisms for message distribution

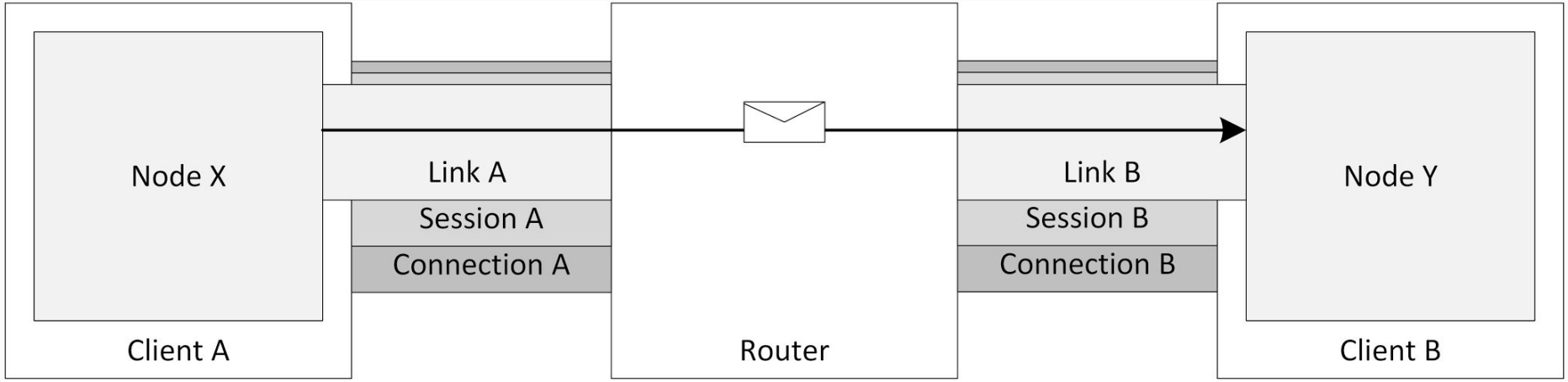
Routing

- Routing is done based on addresses
  - Links are opened against addresses
  - Messages are sent to / received from addresses
  - Addresses are used to configure the routing semantics

Message routing

- Message routing
  - Connection, session and links are between client and router
  - Messages and their settlements are router between clients
  - Messages can be sent to / received from multiple sources
  - Link level features do not propagate to the client on other side (e.g. filters)

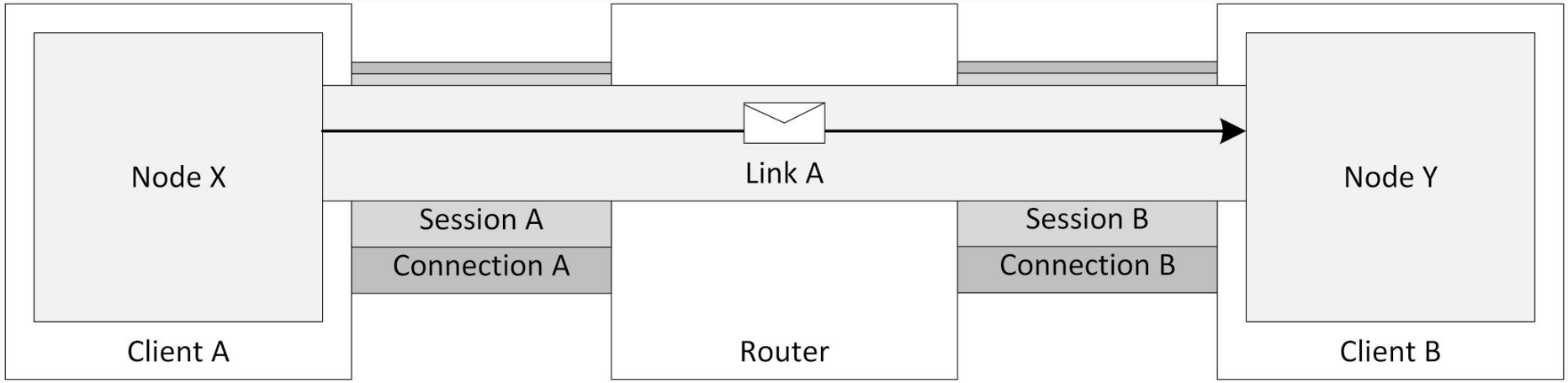
Message routing



Link routing

- Link routing
  - Connection and session are between client and router but links are between clients
  - Everything on link level is routed between clients
  - One link can be routed only to single counterparty. It cannot be shared by many clients
  - Everything on link level is forwarded to the counterparty
    - Filters
    - Capabilities

Link routing



Security

- Support for SSL encryption
- Multiple connectors / listeners with separate configuration
- Wide range of supported SASL mechanisms for authentication
- Extensive policies for authorization including resource limits

Dispatch Use Cases

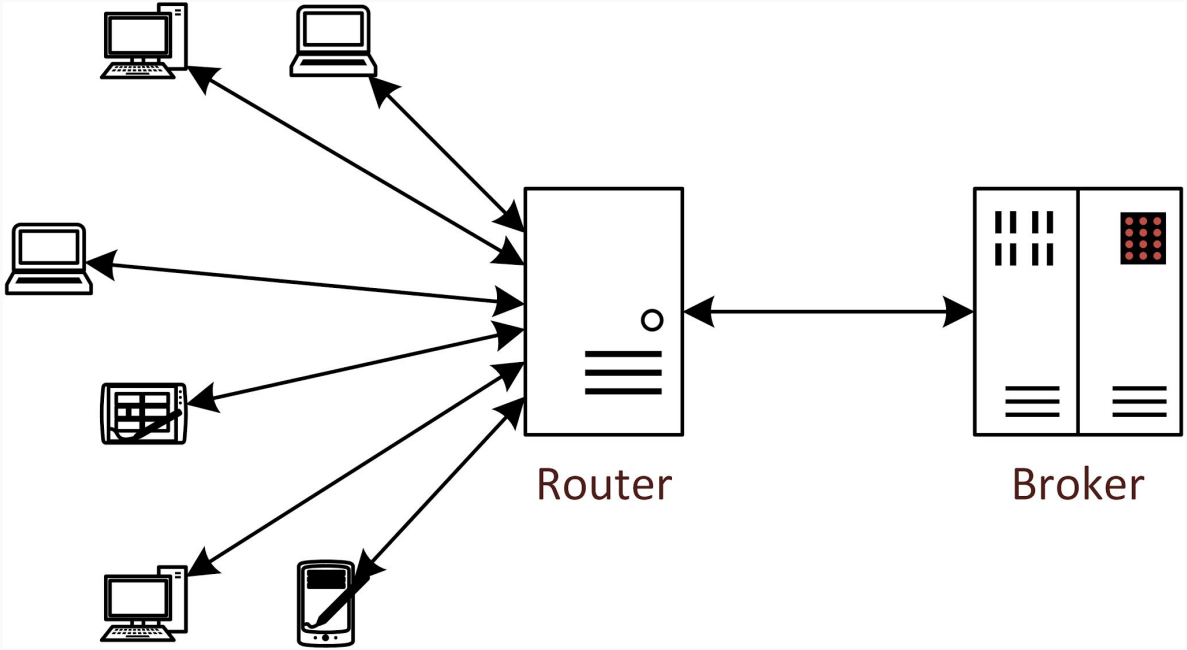
Dispatch Use Cases

- Covers some possible use cases for AMQP router like Qpid Dispatch
- Doesn’t demonstrate all possibilities ;-)

Connection aggregation

- Applications may want to use many connections, but the AMQP broker allows only limited number of connections
- Use Dispatch router :-)
  - Connect the router to the infrastructure with a single connection
  - Open as many connections as you want between your clients and the router
  - Router will route traffic between the clients and the broker
  - Broker will not be under load from handling many separate clients

Connection aggregation

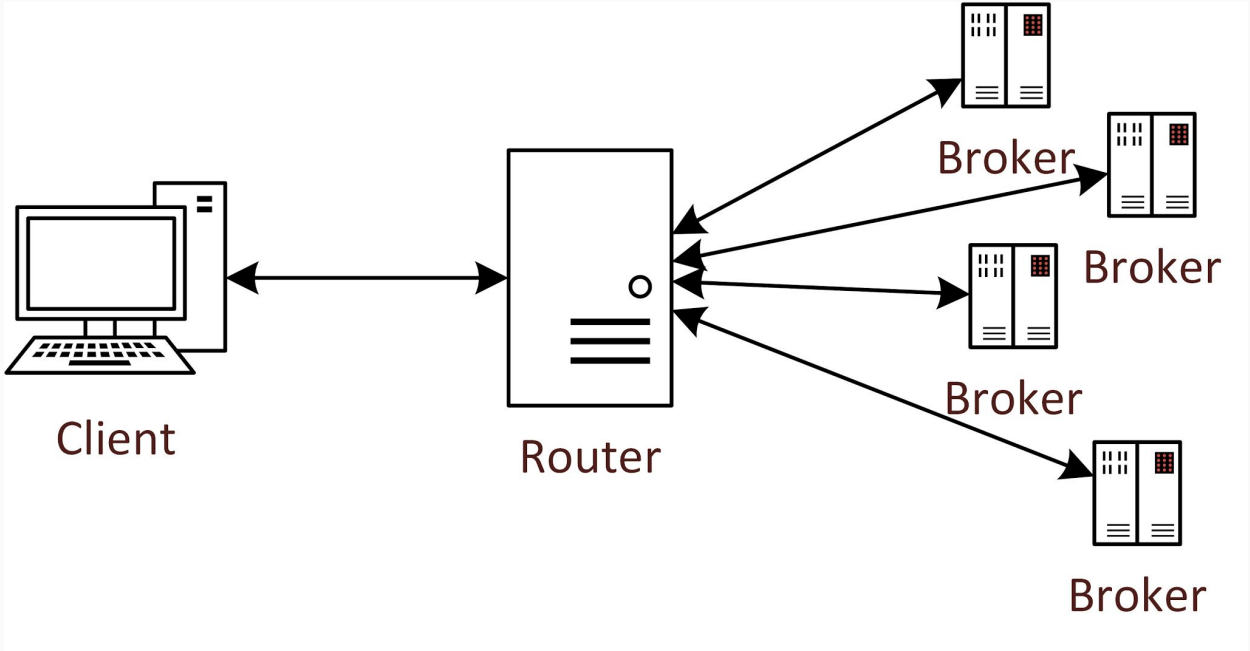


Demo

Connection separation

- Client needs to connect to different AMQP brokers
- Maintaining many connections adds complexity
- Use Dispatch router :-)
  - Connect the router to the different AMQP brokers to forward the messages to / from them
  - Connect the client to the router with a single connection
- Can be also used to shield your application from the AMQP infrastructure
  - The details about which queue is on which broker need to be known only to the router, not to the client application

Connection separation



Demo

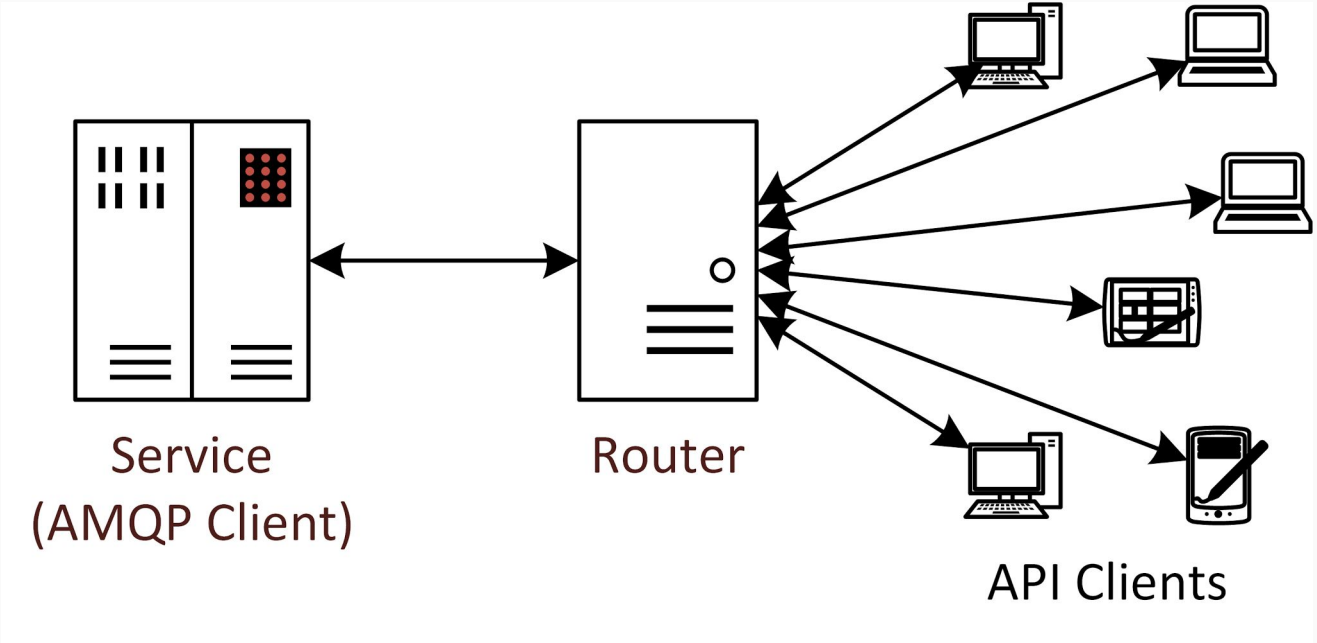
Connection aggregation and separation

http://jsch.cz/blogdispatcheurexclearing

API companion

- APIs are not only HTTP … AMQP based APIs can have many advantages
- As symmetric protocol, AMQP 1.0 makes it easy to build AMQP based APIs into your application
  - But implementing AMQP server is still a lot of work, even with the help of existing libraries
- Use Dispatch router :-)
  - Use only an AMQP client in your service and connect it to the router
  - Configure the router to receive the client connections and forward their request to the service

API companion



Demo

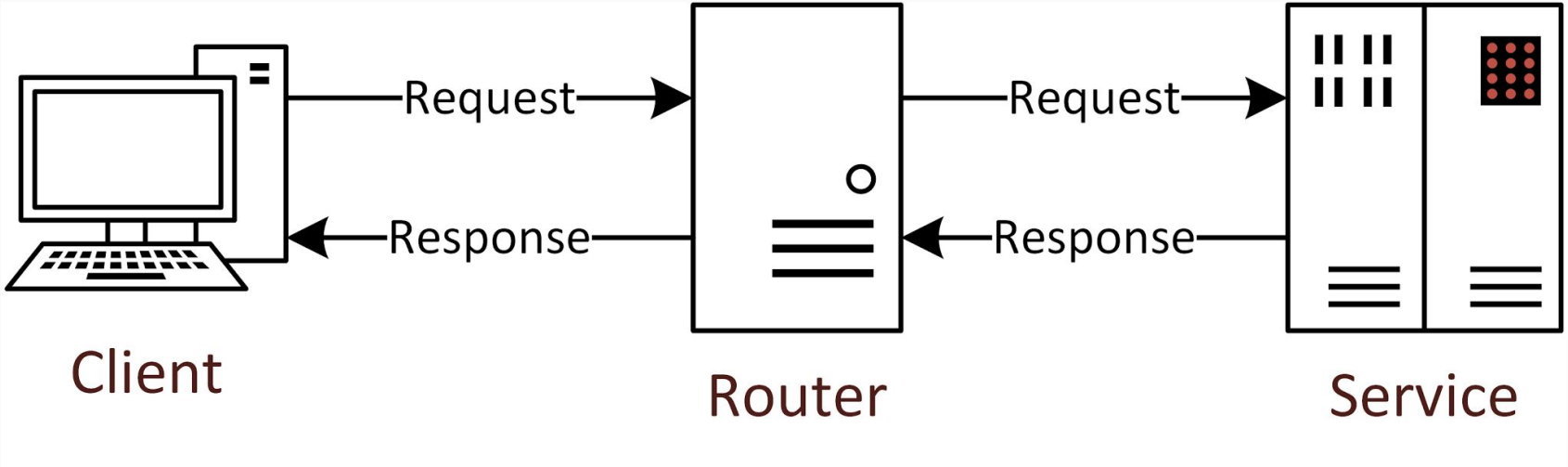
API companion

http://jsch.cz/blogamqpapis

Request - response communication

- Request - response communication often doesn’t need decoupling
  - Requestor usually wants to receive the response in reasonably short time
  - Responder doesn’t want to process requests which nobody wants anymore
  - Broker with a queue can store requests and responses for a long time, but often nobody needs them anymore
- Use Dispatch router :-)
  - Connect your service to the router
  - Open a receiver to a temporary address with your client
  - Send request from your client and use reply-to to tell the service where to send the response

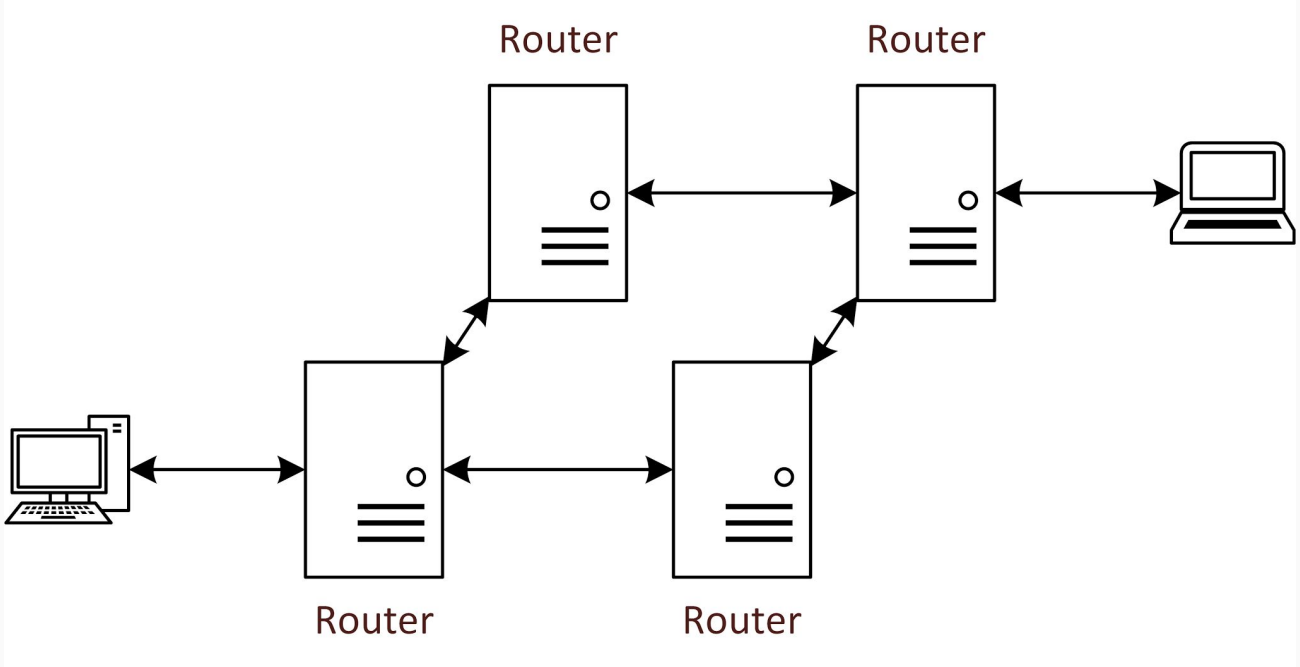
Request - response communication



Router network

- Dispatch routers are horizontally scalable by design
- Dispatch can be configured into router networks
- Router networks provide resiliency and redundancy
- Networks also provide intelligent routing of messages between clients connected to different network nodes

Router network

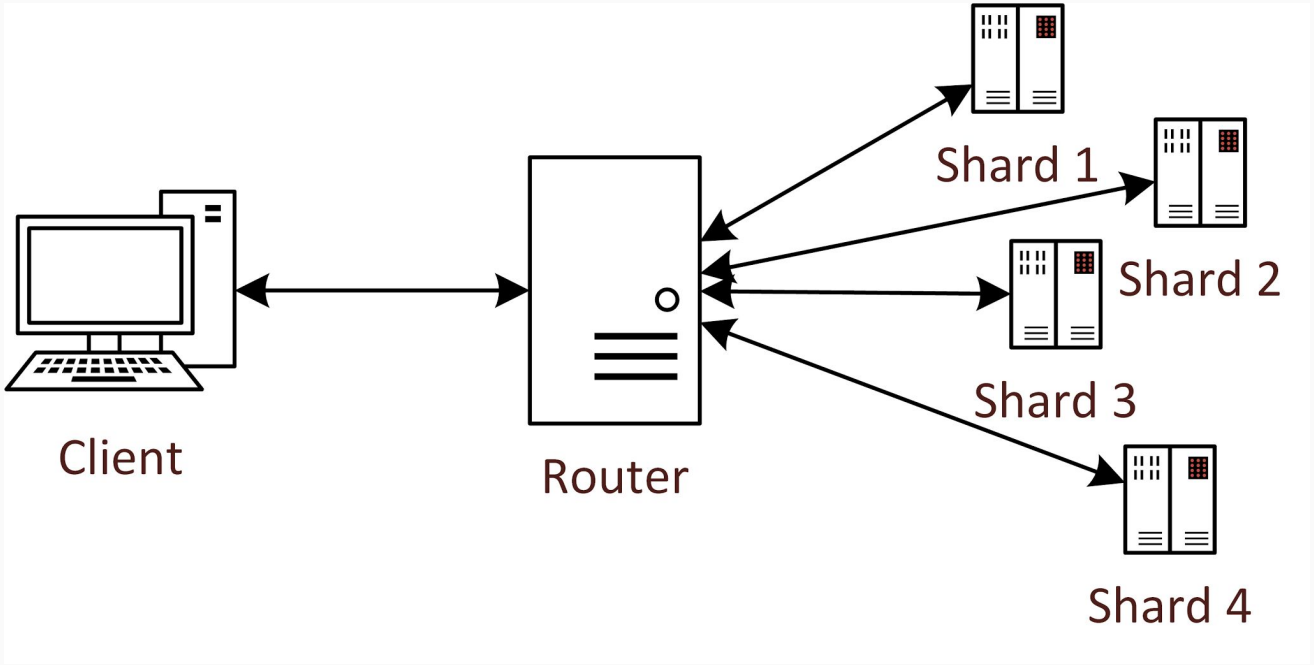


Demo

Sharded queue

- Most AMQP brokers usually support only vertical scalability
  - Queues can be distributed between multiple brokers
  - But once a single queue outgrows the broker it's a problem
- Use dispatch to scale brokers vertically :-)
  - Run several AMQP brokers with the same queue
  - Configure Dispatch router to connect to all these brokers
  - Connect your sender to the router and send messages
  - Connect your receiver to the router and receive messages
  - Dispatch will be automatically distributing the sent messages between all broker instances and serve the messages from all brokers to the receiver

Sharded queue



Demo

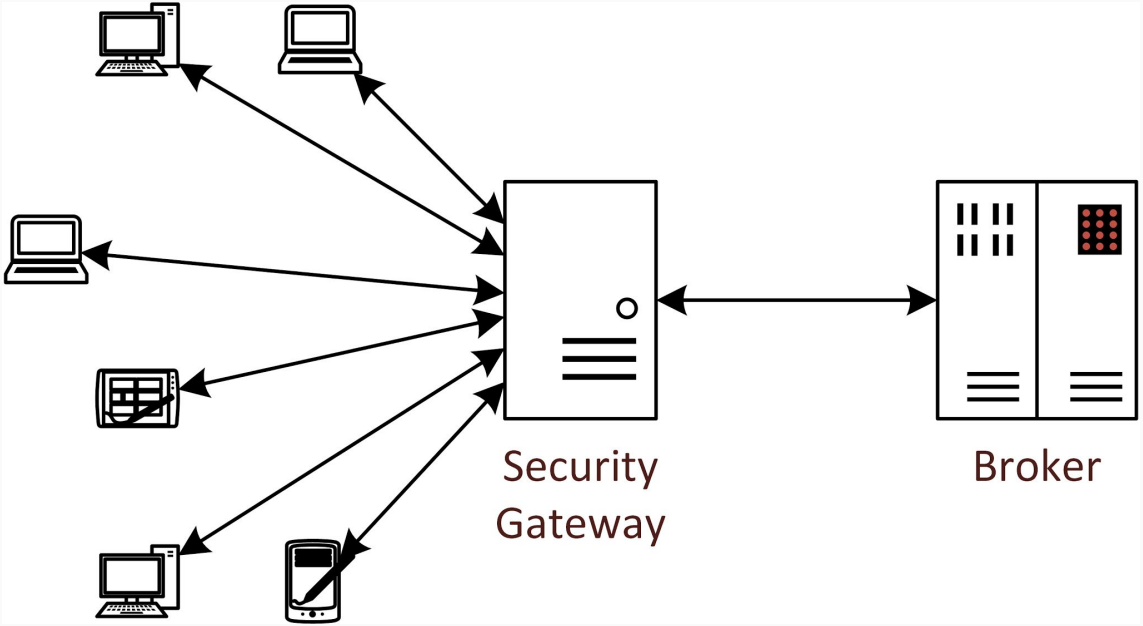
Sharded queue

http://jsch.cz/blogkubernetesscalableamqp

Security Gateway

- Dispatch has extensive security options
- Can be used as a security gateway between clients and an AMQP broker / endpoint
  - Handle authentication
  - Protect consumed resources
  - Handle authorization for different users
- No need to implement authentication & authorization for all your clients in the AMQP endpoint, Dispatch can handle it

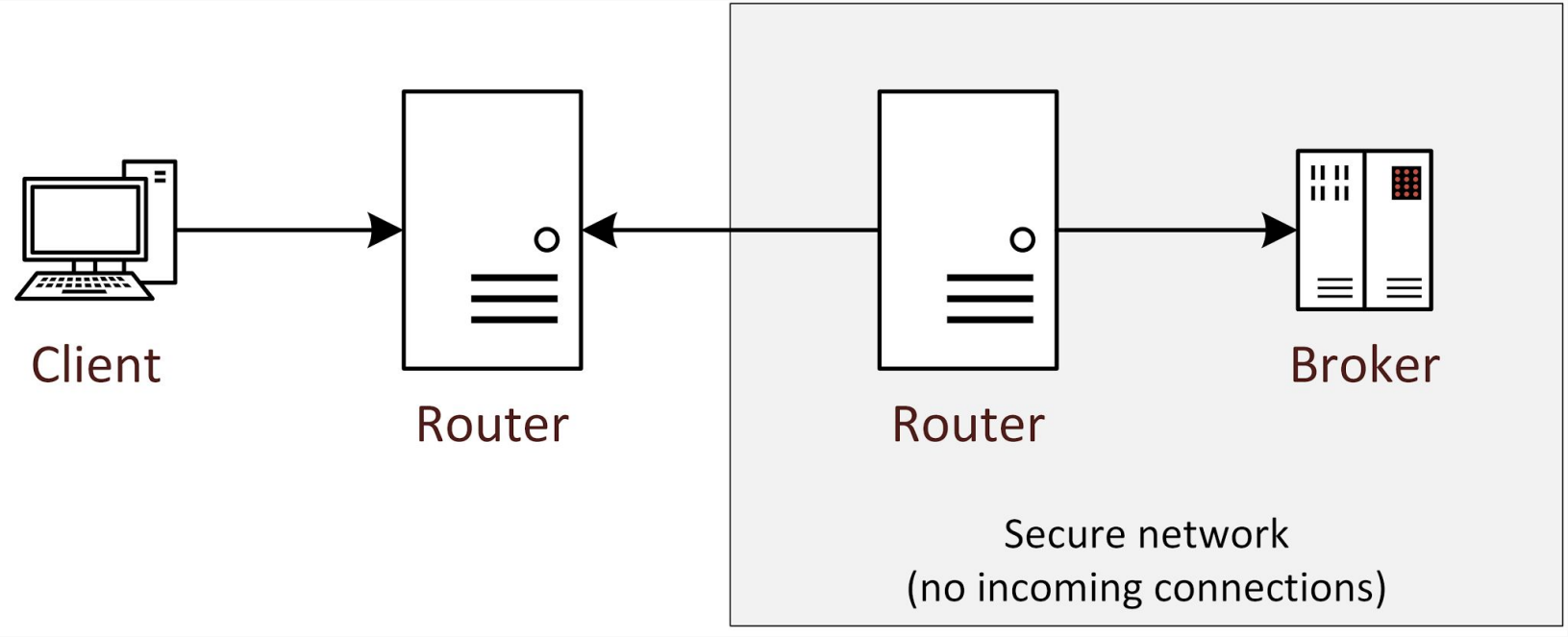
Security Gateway



Network Gateway

- AMQP broker is running in network which doesn’t allow incoming connections, only outgoing
- Client wants to connect to the broker from outside
- Use two Dispatch routers :-)
  - Router A will run in the outside network and work as listener
  - Router B will run in the protected network and open outgoing connection to the broker as well as to router A
  - Client will connect to router A and send / receive messages to / from the broker

Network Gateway



Examples

Examples

http://jsch.cz/amqpdispatchworkshop