OSC 源创会
年终盛典 2015 · 北京

Erlang MQTT消息服务器开发与应用

Erlang, MQTT, Mobile IM/Push and IoT

李枫
〈feng@emqtt.io〉

内容

- eMQTT开源项目简介
- Erlang/OTP语言平台
- MQTT协议与应用
- eMQTT设计与应用

eMQTT开源项目简介

eMQTT项目历史与概况

- eMQTT: Erlang MQTT消息服务器与客户端库
- Erlang/OTP语言平台开发，集群和大规模连接
- GitHub项目地址: https://github.com/emqtt
- 2012年9月首次提交，18个月左右开发维护迭代
- 10+ Contributors, 48+ Releases，300+ Issues，600+Stars，200+ Forks，最新版本0.14

eMQTTD消息服务器

- 项目采用MIT开源协议(MIT License)
- 完整的MQTT V3.1/V3.1.1 协议规范支持
- 保持简单架构，专注接入层与消息路由
- Scalable, Scalable, Massively Scalable…
- 支持插件方式扩展认证与ACL，定制Push、移动IM、物联网等应用
- MQTT, HTTP Publish, WebSocket, Stomp, SockJS, CoAP多协议接口

eMQTT其他开源子项目

- eSockd：通用的Erlang TCP服务端框架
- emqttc：Erlang MQTT客户端库
- emqtt_benchmark：MQTT连接测试工具
- CocoaMQTT：Swift语言MQTT客户端库
- QMQTT：QT框架MQTT客户端库

Erlang/OTP语言平台

Erlang/OTP历史

- 爱立信、Joe Armstrong，1988年起近30年历史
- Erlang语言最初设计目的是开发电信设备与系统
- 1998年开源，2006多核CPU支持，互联网、即时消息、云计算应用
- C++、Java、C#面向对象系列截然不同的设计思路
- 以消息为主的移动互联网、物联网最佳服务端平台？

Erlang/OTP平台特点

- 高并发(Concurrency, Muti Core, Threads, Massive Processes)
- 低延时(Low-Latency)
- 软实时(Soft real-time)
- 容错(Fault-tolerant, monitor, link, supervisor tree)
- 分布(Distributed nodes, mnesia)
- 水平伸缩(Scalable)
- 热升级(Hot Upgrade)

Erlang虚拟机

- 类似操作系统：CPU Cores, Schedulers, Threads, Massive Processes
- 跨平台(Linux, FreeBSD, AIX, HP-UX, 树莓派…Windows)
- 出色的内存管理：process heap, binary, ets…
- 细粒度垃圾回收(Fine-grained Garbage Collected)
- 轻量进程、公平调度

Erlang编程语言(1)

- 函数编程(Functional Programming)
- 模式匹配(Pattern Match)
- 轻量进程(Lightweight Processes)
- 消息传递(Message Passing)
- 递归(Recursion)、尾递归(Tail Recursion)
- Actor-Oriented, Object-Oriented?

Erlang编程语言(2)

- Atom, Pid, Tuple, List, Binary, Port…
- List, Binary Comprehension
- Binary Match解析网络协议
- 闭包(Closure)与高阶函数(HigherOrder Functions)
- 参数化模块(Parameterized Module)

Erlang编程语言(3)

- List, Binary Comprehension

```
<< <<(serialise_utf(Topic))/binary, ?RESERVED:6, Qos:2>> || {Topic, Qos} <-
Topics >>;
```

```
routes(Topics, Pid) ->
    lists:unzip([{{Topic, Pid}, {Pid, Topic}} || Topic <- Topics]).
```

- 参数化模块

```
new(Sock, SockFun, Opts) ->
    {?MODULE, [Sock, SockFun, parse_opt(Opts)]}.
```

Erlang/OTP平台

- OTP(Open Telecom Platform)
- 行为(Behaviours)
- 监控(Supervisor)
- 应用(Applications)
- 发布(Releases)

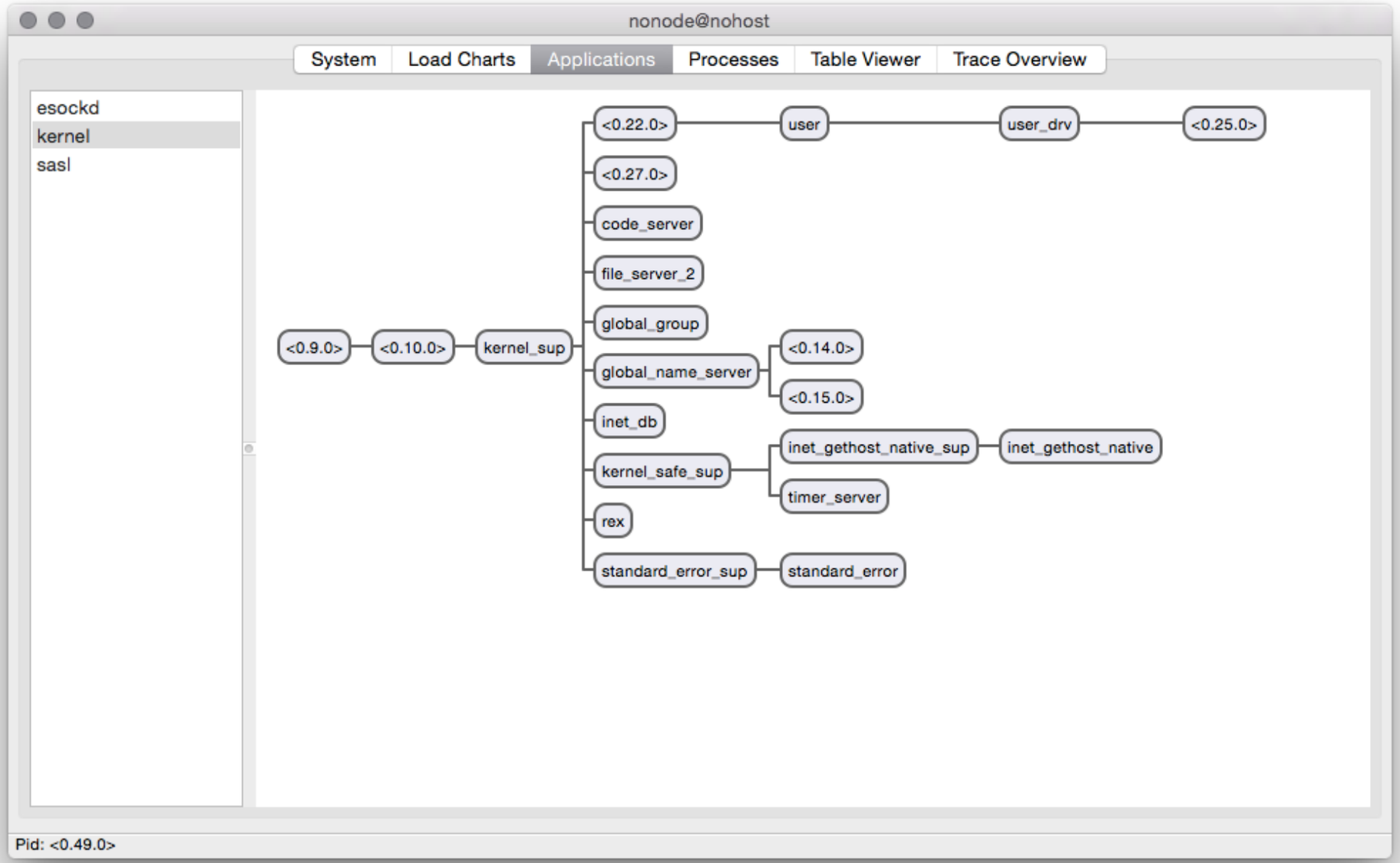
Erlang/OTP平台-Behaviour

- gen_server(客户端服务器)
- gen_fsm(有限状态自动机)
- gen_event(事件通知)

Erlang/OTP平台-Supervisor

- Supervisor Restart Strategies
  - one_for_all
  - one_for_one
  - rest_for_one
  - simple_one_for_one

Erlang/OTP平台-Application



Erlang/OTP平台-Release

- ERTS + Boot脚本 + Applications => Binary Package

MQTT协议与应用

MQTT协议

- MQTT V3.1/V3.1.1协议规范(IBM)
- 发布订阅模式(Publish/Subscribe)
- 基于Topic消息路由(Topic based Subscription Model)
- QoS 0,1,2 Messages
- Transient, Persistent Sessions
- Last Will, Retained Message
- KeepAlive and Two Bytes Heartbeat
- MQTT Over WebSocket

MQTT协议-报文(1)

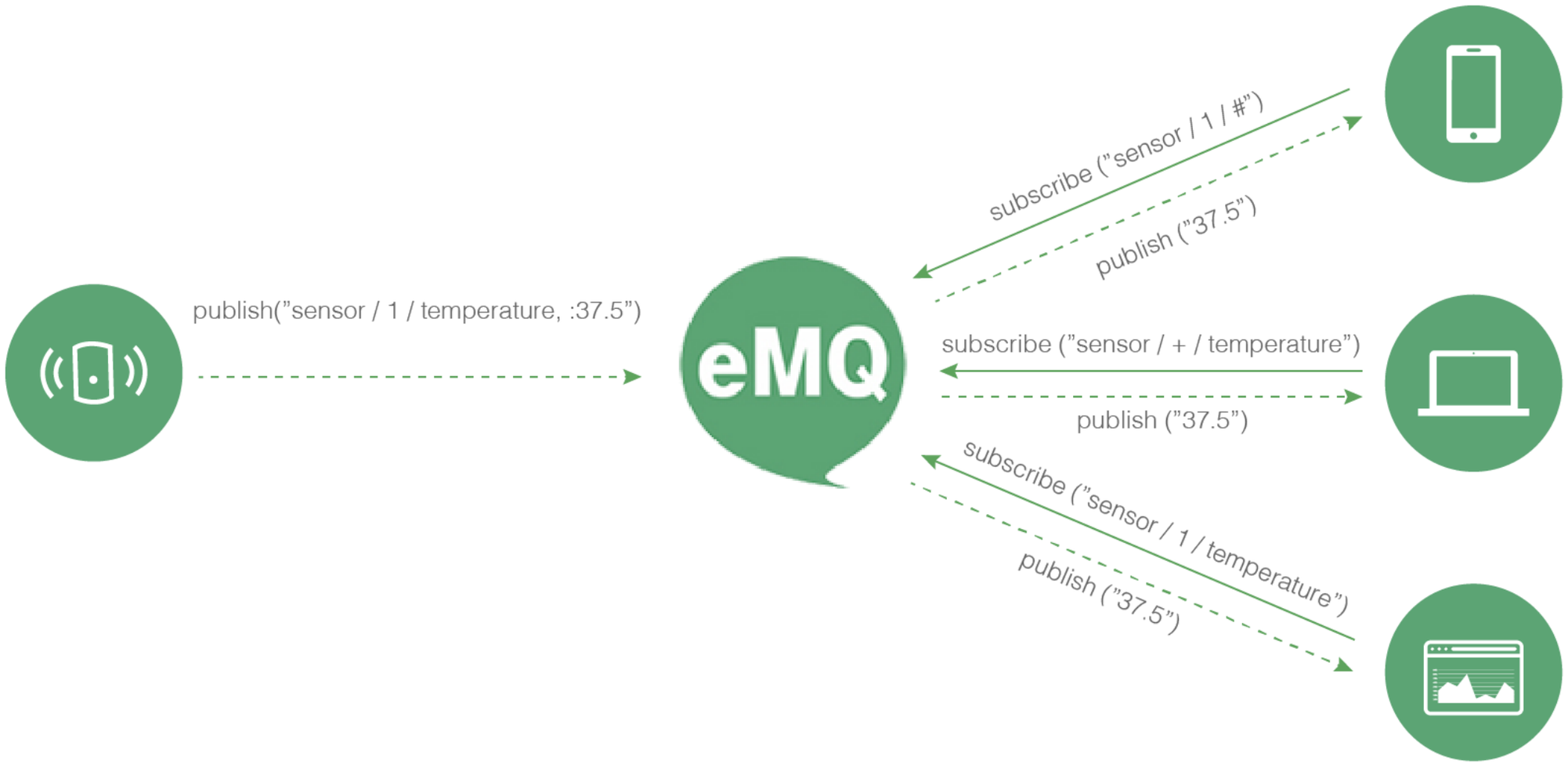
Fixed header, present in all MQTT Control Packets
Variable header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2…	Remaining Length							

MQTT协议-报文(2)

CONNECT	发起连接	CONNACK	连接回执
PUBLISH	发布消息	PUBACK	发布回执
PUBREC	QoS2消息回执	PUBREL	QoS消息释放
PUBCOMP	QoS2消息完成	DISCONNECT	断开连接
SUBSCRIBE	订阅Topic	SUBACK	订阅回执
UNSUBSCRIBE	取消订阅	UNSUBACK	取消订阅回执
PINGREQ	PING请求	PINGRESP	PING响应

MQTT协议-PubSub



MQTT协议-Topic Name/Filter

Publish to Topic Name:

- “chat/room/1”
- “sensor/10/temperature”
- “$SYS/broker/metrics/packets/received”

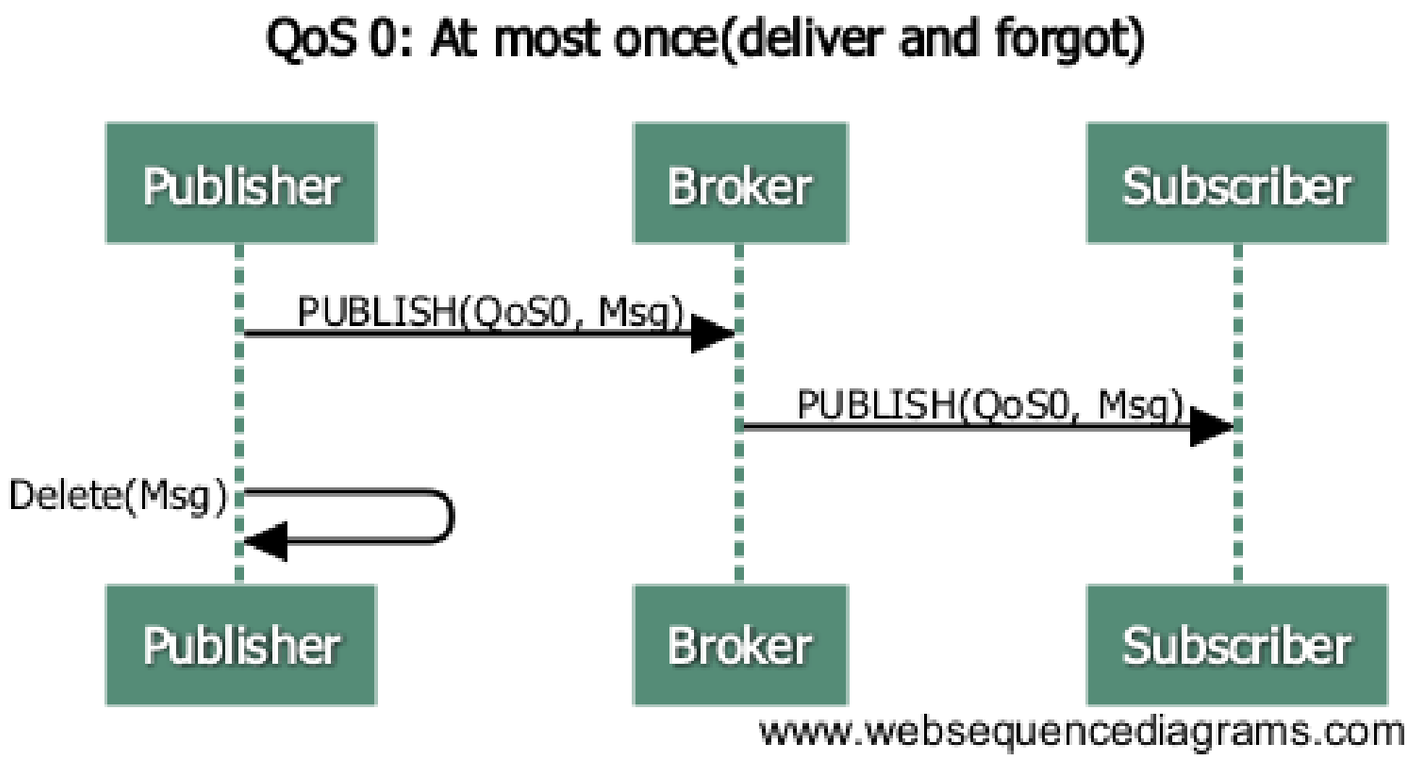
Subscribe Topic Filter:

- “chat/room/1”
- “sensor/+/temperature”
- “$SYS/broker/metrics/#”

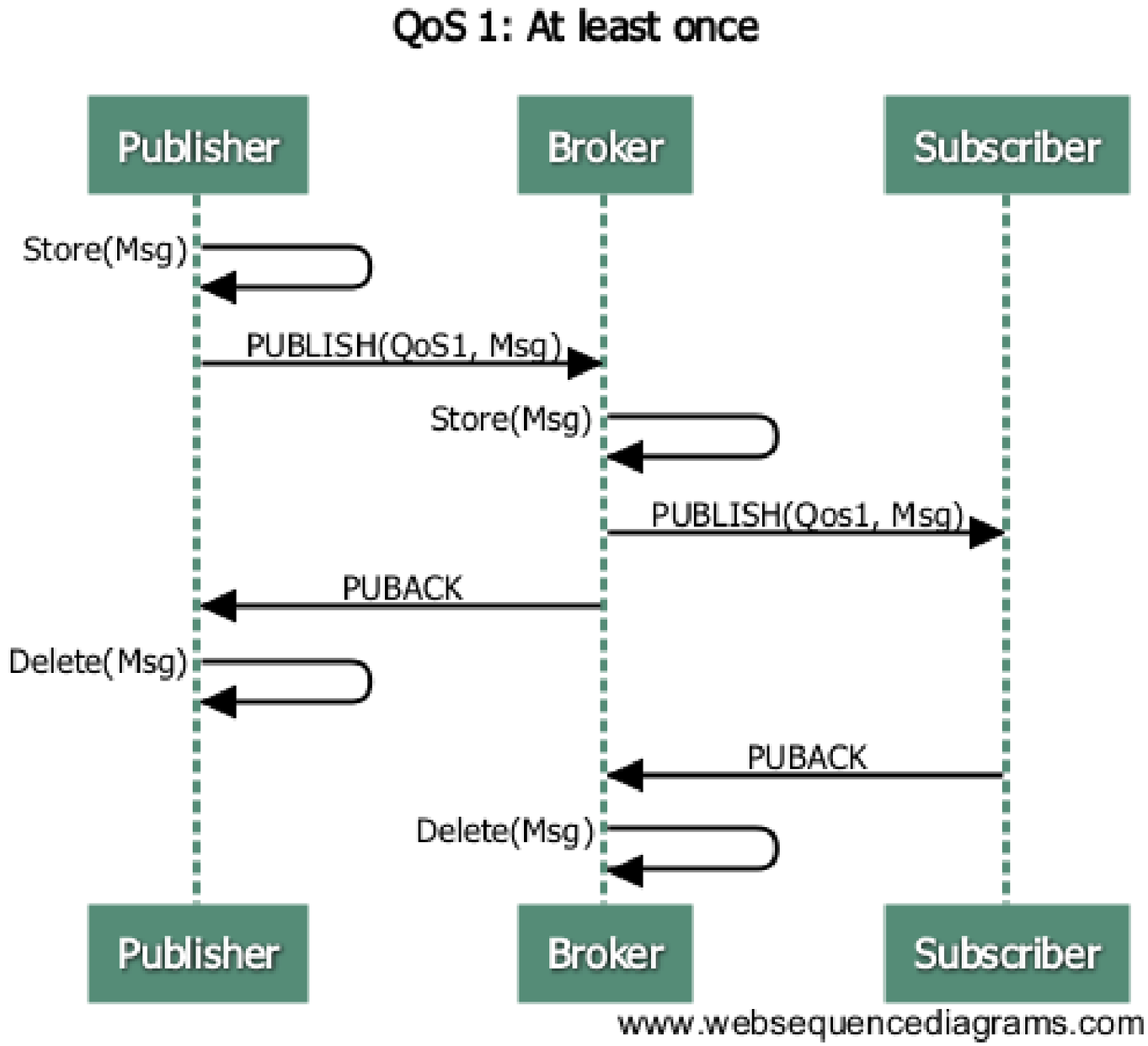
MQTT协议-QoS0/1/2

发布消息的QoS	Topic订阅的QoS	接收消息的QoS
0	0	0
0	1	0
0	2	0
1	0	0
1	1	1
1	2	1
2	0	0
2	1	1
2	2	2

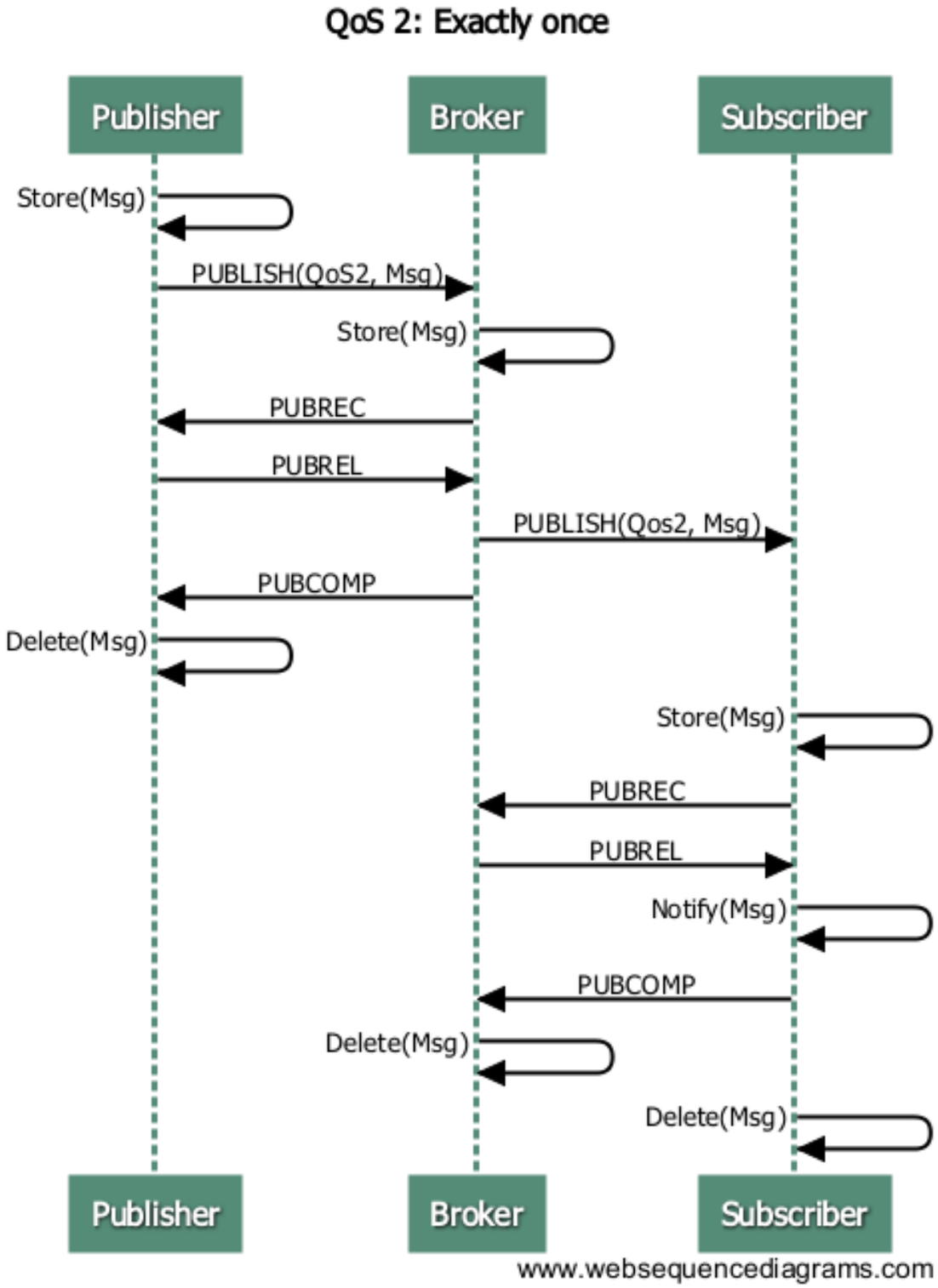
MQTT协议-QoS0


QoS 0: At most once(deliver and forgot)
Publisher
Broker
Subscriber
PUBLISH(QoS0, Msg)
PUBLISH(QoS0, Msg)
Delete(Msg)
Publisher
Broker
Subscriber
www.websequencediagrams.com

MQTT协议-QoS1


QoS 1: At least once
Publisher
Broker
Subscriber
Store(Msg)
PUBLISH(QoS1, Msg)
Store(Msg)
PUBLISH(Qos1, Msg)
PUBACK
Delete(Msg)
PUBACK
Delete(Msg)
Publisher
Broker
Subscriber
www.websequencediagrams.com

MQTT协议-QoS2


QoS 2: Exactly once
Publisher
Broker
Subscriber
Store(Msg)
PUBLISH(QoS2, Msg)
Store(Msg)
PUBREC
PUBREL
PUBLISH(Qos2, Msg)
PUBCOMP
Delete(Msg)
Store(Msg)
PUBREC
PUBREL
Notify(Msg)
PUBCOMP
Delete(Msg)
Delete(Msg)
Publisher
Broker
Subscriber
www.websequencediagrams.com

MQTT协议-Sessions

- Clean Session Flag
- Transient Session
- Persistent Session
- Offline Message

MQTT协议-KeepAlive

- CONNECT报文KeepAlive参数
- PINGREQ 2字节心跳报文
- XMPP KeepAlive???

MQTT协议-Last Will, Retained Message

- Last Will

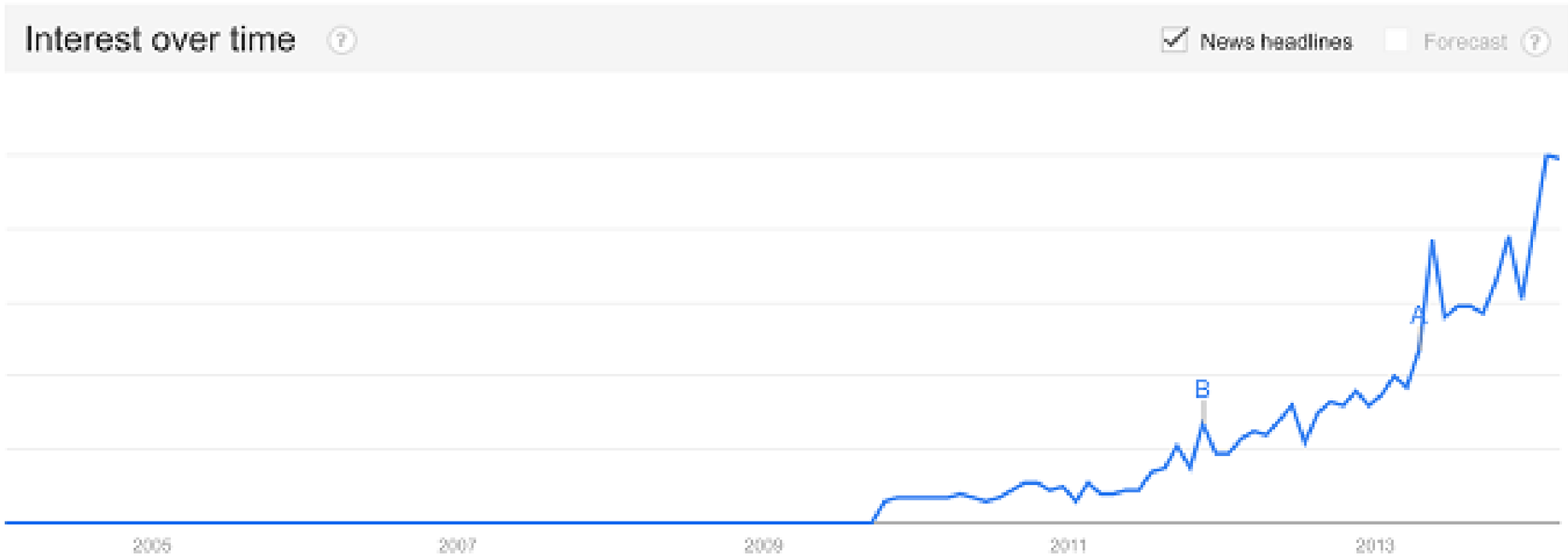
- Retained Message

MQTT协议-WebSocket

- Binary mode frame over WebSocket
- PubSub on Web Browser
- Firefox, Safari, Chrome, Opera…
- IE Sucks?
- Better than Socket.IO?

MQTT应用-Mobile, IoT, M2M…

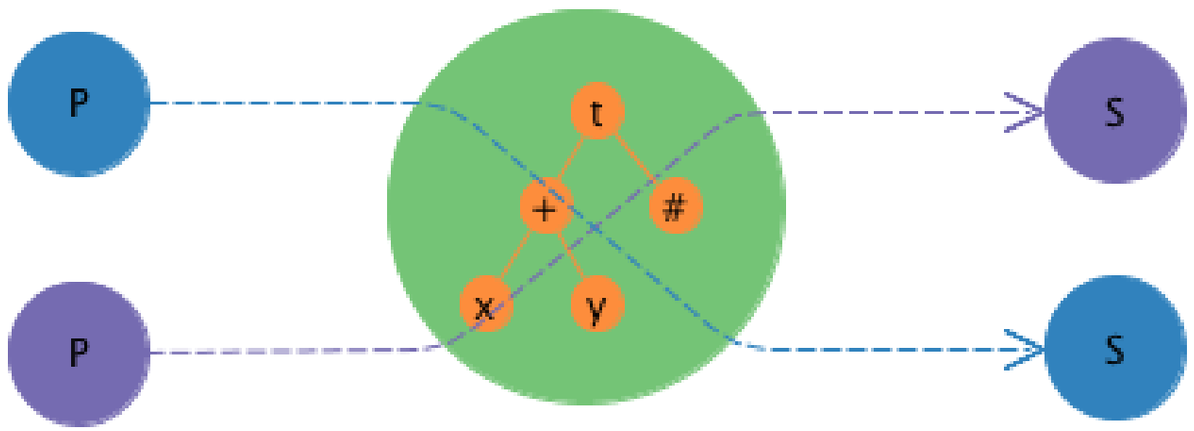
- Android Push
- Mobile Chat(Facebook Messenger)
- 物联网(IoT, M2M)、智能硬件、车联网...
- 行业市场(电力、石油、能源…)

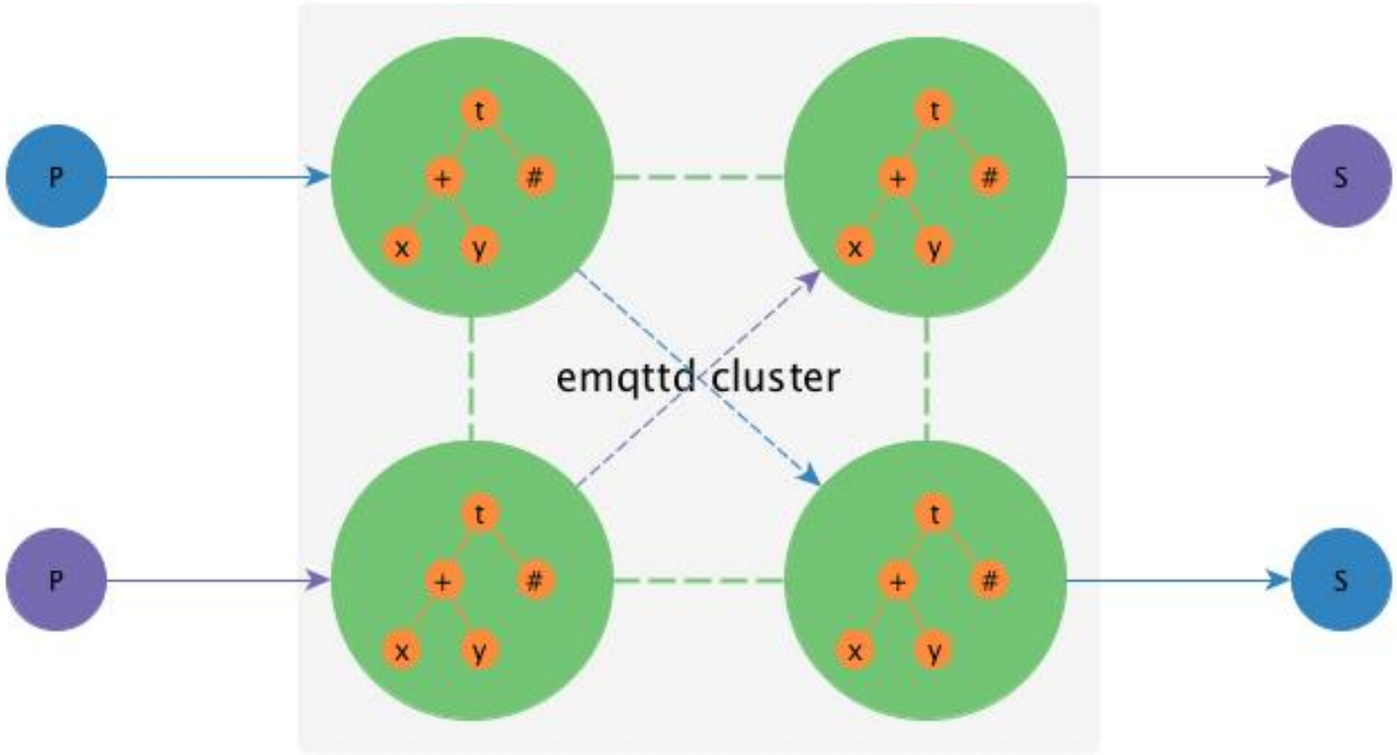


eMQTT设计与应用

自由 • 开放 • 分享

架构-概念模型





设计-分层与模块

- 连接层(Socket, Client, Protocol)
- 会话层(Global Session)
- 路由层(Router, PubSub)
- 分布层(Trie树, Topic表)
- 认证与访问控制(ACL)
- 钩子(Hooks)与插件(Plugins)
- Erlang相关的设计建议?

设计-连接层(Socket, Client, Protocol)

- eSockd - General Non-blocking TCP/SSL Socket Server
- Acceptor Pool and Asynchronous TCP Accept
- Max Connection Management
- Leaky Bucket Rate Liming
- KeepAlive Timer
- Parser and Serializer
- Protocol Packets Procecess

设计-连接层(Socket, Client, Protocol)

- TCP/SSL Connection Support
- MQTT Over WebSocket(SSL) Support
- HTTP Publish API Support
- Stomp, SockJS Support
- Private TCP Protocol

设计-连接层(Socket, Client, Protocol)

- 全异步TCP收发

```
handle_info({inet_async, _Sock, _Ref, {ok, Data}}, State) ->
    Size = size(Data),
    ?LOG(debug, "RECV ~p", [Data], State),
    emqttd_metrics:inc('bytes/received', Size),
    received(Data, rate_limit(Size, State#client_state{await_recv = false}));

handle_info({inet_async, _Sock, _Ref, {error, Reason}}, State) ->
    shutdown(Reason, State);

handle_info({inet_reply, _Sock, ok}, State) ->
    hibernate(State);

handle_info({inet_reply, _Sock, {error, Reason}}, State) ->
    shutdown(Reason, State);
```

设计-连接层(Socket, Client, Protocol)

- Parser Fun

```
-spec parse(binary(), {none, [option()]} | fun()) -> {ok, mqtt_packet()} | {error, any()} | {more, fun()}.
parse(<<>>, {none, Limit}) ->
    {more, fun(Bin) -> parse(Bin, {none, Limit}) end};
parse(<<PacketType:4, Dup:1, QoS:2, Retain:1, Rest/binary>>, {none, Limit}) ->
    parse_remaining_len(Rest, #mqtt_packet_header{type   = PacketType,
                                                  dup    = bool(Dup),
                                                  qos    = QoS,
                                                  retain = bool(Retain)}, Limit);
parse(Bin, Cont) -> Cont(Bin).
```

- Serializer Fun

```
-spec serialize(mqtt_packet()) -> binary().
serialize(#mqtt_packet{header = Header = #mqtt_packet_header{type = Type},
                       variable = Variable,
                       payload  = Payload}) ->
    serialize_header(Header,
        serialize_variable(Type, Variable,
            serialize_payload(Payload))).
```

设计-会话层(Session)

- 会话层处理MQTT协议PUBLISH/SUBSCRIBE消息交互流程
- Qos0/1/2消息接收与下发，消息超时重传，离线消息保存
- 飞行窗口(Inflight Window)，下发消息的顺序保证
- 缓存MQTT客户端的全部订阅(Subscription)，并终结QoS
- 服务器发送到客户端的，已发送未确认的Qos1/2消息
- 客户端发送到服务端，未接收到PUBREL的QoS2消息
- 客户端离线时，持久会话保存离线的Qos1/2消息

设计-会话层(Session)

```
       |<------------------ Max Len ------------------>|
       ------------------------------------------------
IN -> |       Pending Messages    | Inflight Window   | -> Out
       ------------------------------------------------
                                   |<--- Win Size --->|
```

- 消息队列(Message Queue)和飞行窗口(Inflight Window)

- 飞行窗口(Inflight Window)保存当前正在发送未确认的Qos1/2消息。窗口值越大，吞吐越高；窗口值越小，消息顺序越严格

- 当客户端离线或者飞行窗口(Inflight Window)满时，消息缓存到队列

- 如果消息队列满，先丢弃Qos0消息，或者丢弃最早进入队列的消息

设计-会话层(Session)

- PacketId 客户端到服务端的Packet收发与确认
- MessageId 全局唯一的、时间序列的消息ID，分配给每一条Qos1/2消息，用于端到端的消息处理

```
PktId <-- --> MsgId <-- --> MsgId <-- --> PktId
     |<--- Qos --->|<---PubSub--->|<-- Qos -->|
```

设计-会话层(Session)

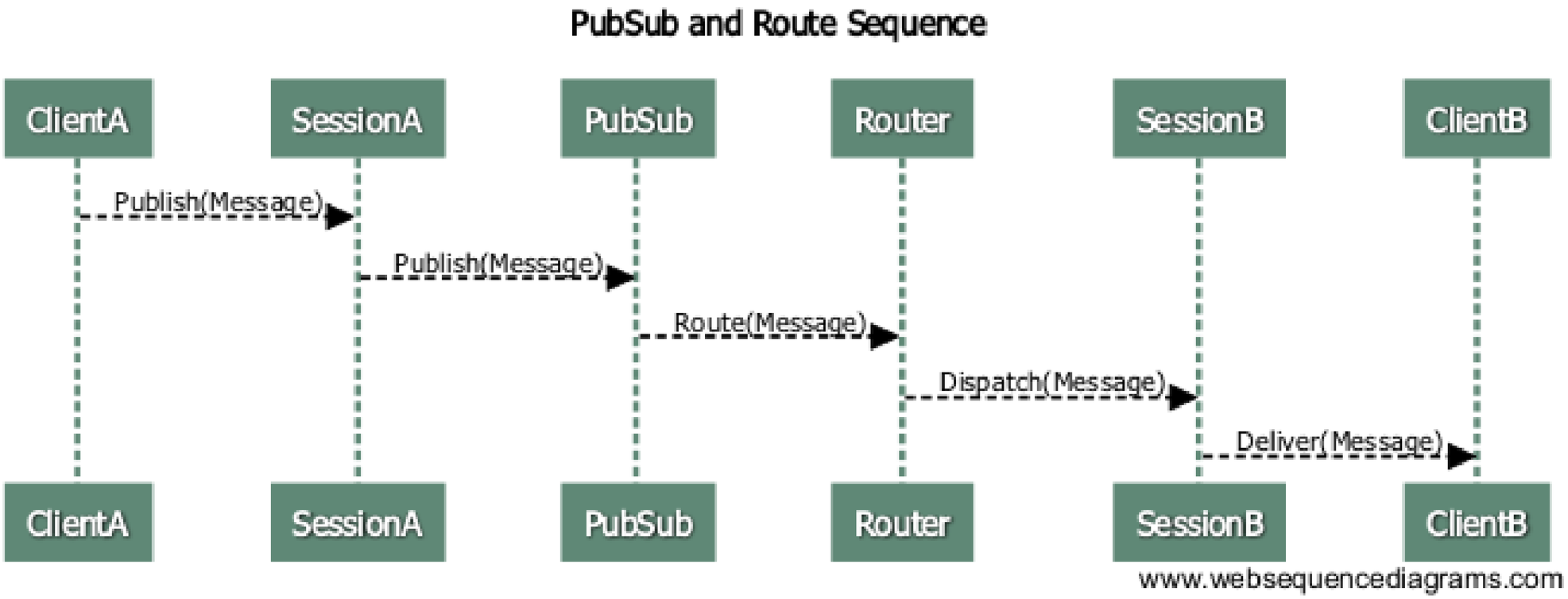
```
------------------------------------------------------------
|        Timestamp        |  NodeID + PID  |  Sequence  |
|<------- 64bits ------->|<--- 48bits --->|<- 16bits ->|
------------------------------------------------------------
```

- 全局唯一消息ID结构:
  - 64bits时间戳: erlang:system_time if Erlang >= R18, otherwise os:timestamp
- Erlang节点ID: 编码为2字节
- Erlang进程PID: 编码为4字节
- 进程内部序列号: 2字节的进程内部序列号

设计-路由层(Router, PubSub)

- 字典树(Trie)匹配路由
- Topic表读取分布节点
- Router进行消息路由分发
- Session消息送达与重传

设计-路由层(Router, PubSub)



设计-分布层(Distributed)

- 集群(Cluster)
  - Mnesia数据库复制实现集群：一个disco_copies节点，多个ram_copies节点
  - 订阅关系(Subscriptions)、本地路由表分别保存在各自节点
  - Topic Trie树、Topic->Node映射表多节点复制
- 桥接(Bridge)
  - Pub --> Broker1 --- Bridge Forward--> Broker2 -- Bridge Forward --> Broker3 --> Sub
  - 桥接节点间只消息转发，不复制Mnesia数据库

设计—认证与ACL

- 认证方式
  - 用户名、密码认证
  - ClientID认证
  - 匿名认证(anonymous)
  - 浏览器Cookie认证
- 插件认证
  - LDAP
  - MySQL
  - PostgreSQL

设计—认证与ACL

- ACL访问控制设计
  (https://github.com/emqtt/emqttd/wiki/ACL)

- {allow | deny, Who, Access, TopicFilters}.

- Who :: all | ClientId | {client, ClientId} |
  {ipaddr, IpAddr} | {user, Username}

```
{allow, {user, "dashboard"}, subscribe, ["$SYS/#"]}.

{allow, {ipaddr, "127.0.0.1"}, pubsub, ["$SYS/#", "#"]}.

{deny, all, subscribe, ["$SYS/#", {eq, "#"}]}.

{allow, all}.
```

设计—认证与ACL

- ACL访问控制插件：
  - Internel: etc/acl.config
  - MySQL
  - PostgreSQL
  - Redis(TODO)

设计—钩子(Hooks)

Hooks设计
(https://github.com/emqtt/emqttd/wiki/Hooks%

Name	Type	Description
client.connected	foreach	Run when client connected successfully
client.subscribe	foldl	Run before client subscribe topics
client.subscribe.after	foreach	Run After client subscribe topics
client.unsubscribe	foldl	Run when client unsubscribe topics
message.publish	foldl	Run when message is published
message.acked	foreach	Run when message is acked
client.disconnected	foreach	Run when client is disconnnected

设计—插件(Plugins)

- emqttd_plugin_template - Plugin template and demo
- emqttd_dashboard - Web Dashboard
- emqttd_plugin_mysql - Authentication with MySQL
- emqttd_plugin_pgsql - Authentication with PostgreSQL
- emqttd_plugin_redis - Redis Plugin
- emqttd_stomp - Stomp Protocol Plugin
- emqttd_sockjs - SockJS(Stomp) Plugin
- emqttd_recon - Recon Plugin

设计—Erlang相关

- 使用Pool, Pool, Pool… and GProc(github.com/uwiger/gproc)
- 异步，异步，异步消息...同步用于负载保护
- 避免进程Mailbox累积消息，负载高的进程可以使用gen_server2
- 避免过度使用gen_server2, erlang:demonitor(MRef, [flush])不能工作, RabbitMQ 3.5.x之前hibernate有问题(https://github.com/rabbitmq/rabbitmq-server/pull/269)
- 服务器Socket连接、会话进程必须Hibernate
- 多使用Binary数据，避免进程间内存复制

设计—Erlang相关

- 使用ETS, ETS, ETS…Message Passing Vs ETS
- 避免ETS select, match without key
- 避免大量数据读写ETS, 使用lookup_element, update_counter…
- 适当开启ETS表{write_concurrency, true}
- 保护Mnesia Transaction，避免overload
- 避免Mnesia index_read, match, select

设计—Erlang相关

- erlang:system_monitor监控long_schedule, long_gc, busy_port, busy_dis_port

- etop查看msg_q, memory, reductions, runtime…

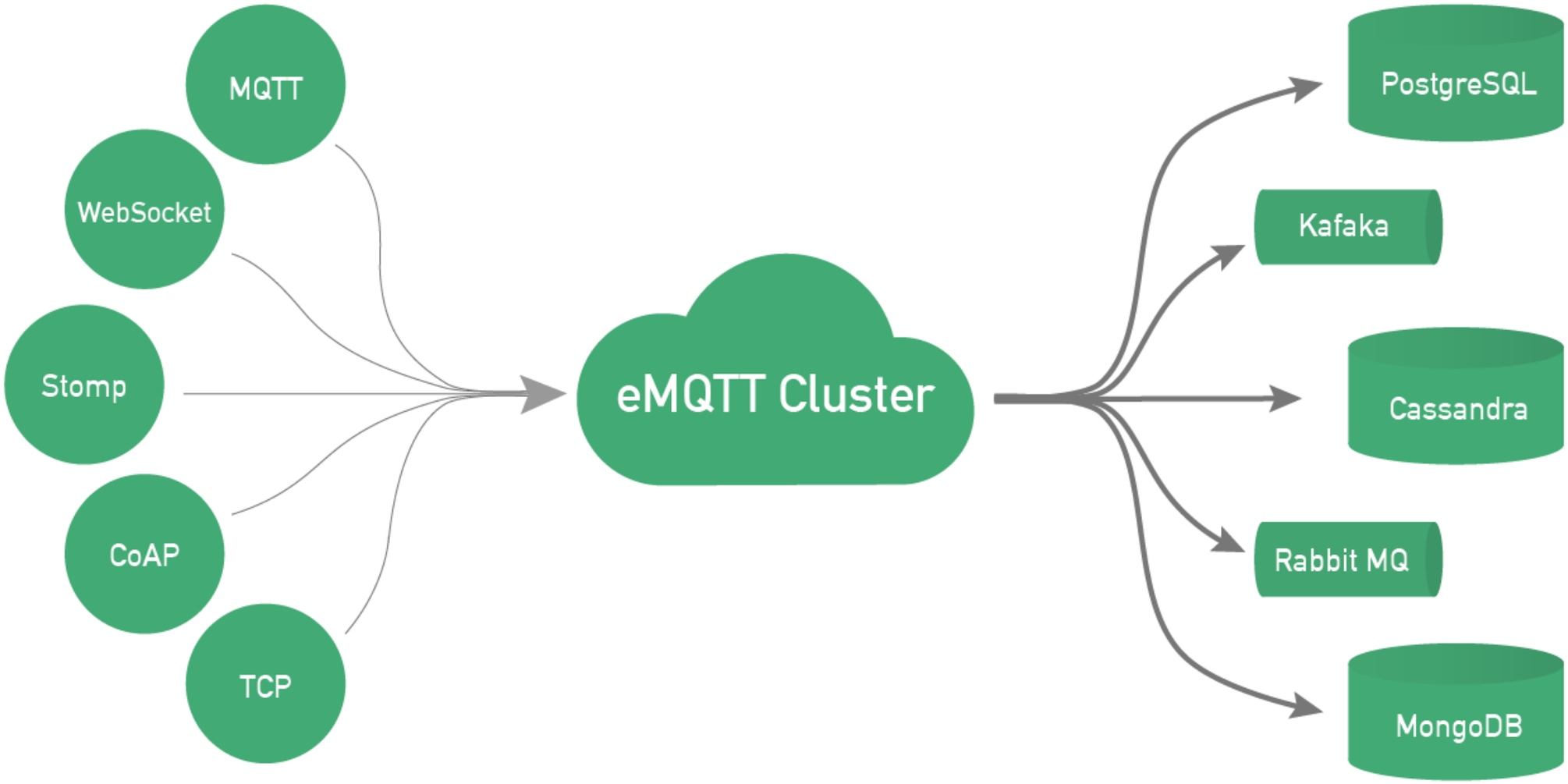
eMQTT-Benchmark

- 3G内存, 50%CPU/核心 (8核, 32G内存CentOS节点)
  - 250+K Connections,
  - 50K Topics,
  - 250K Subscribers,
  - 4K Qos1 Messages/Sec In,
  - 20K Qos1 Messages/Sec Out,
  - 12M+(bps) In, 56M+(bps) Out Traffic
- 产品环境：500K+手机连接
- 压力测试：900K+测试连接

eMQTT应用

- IoT, M2M PubSub
- Mobile Push
- Mobile Chat
- Web Push/Chat
- Hardware, Terminal, Raspberry Pi…
- Enterprise MQTT Server

eMQTT应用-Kafka, DB, NoSQL集成



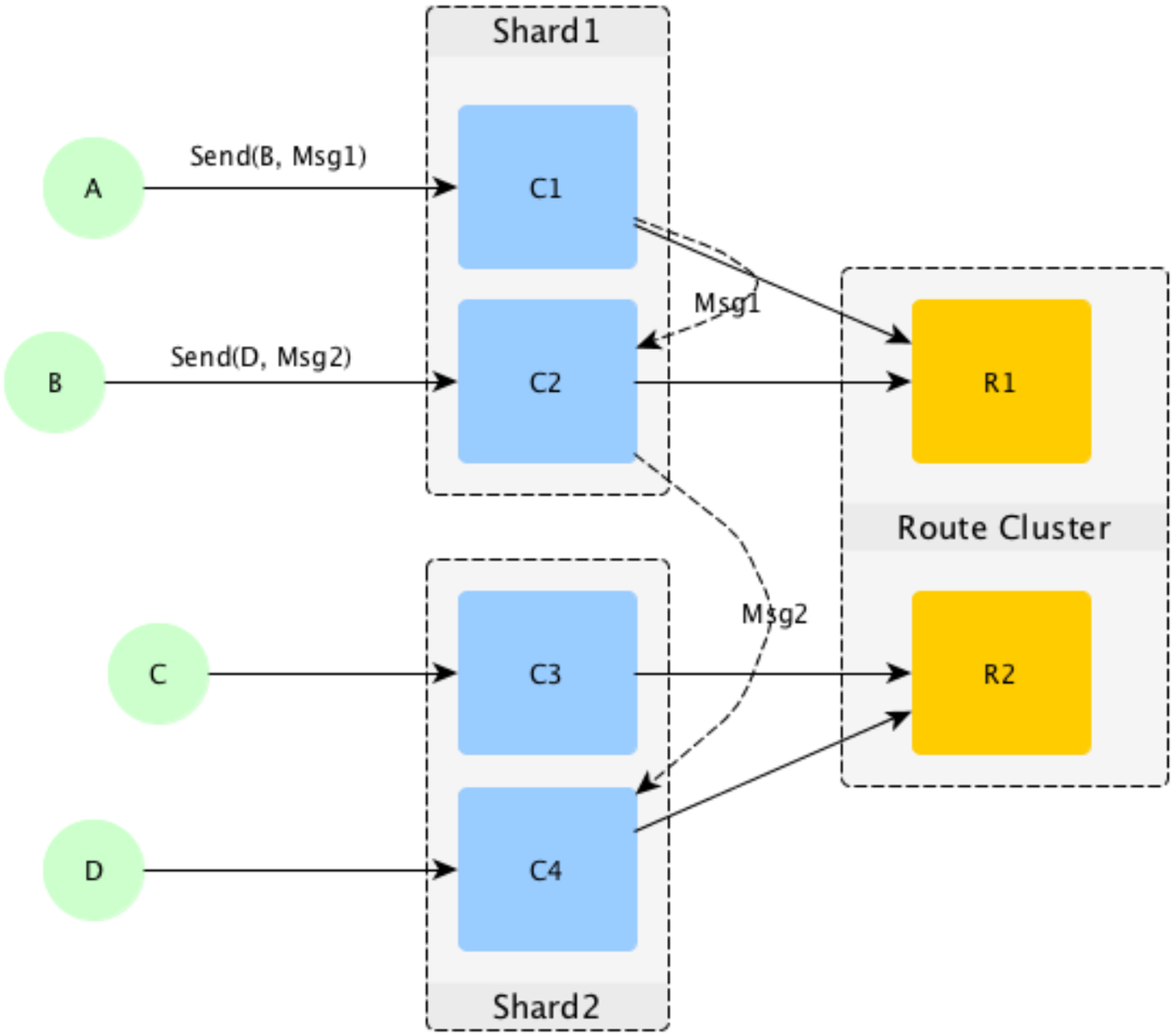
eMQTT应用—IoT, M2M PubSub

eMQTT应用—Mobile Push

- 单节点100万连接，最高1Gbps下行，路由分片流控
- 通过Topic订阅，按地域、行业、内容推送分类消息
- 通过持久化Session处理移动终端频繁上下线问题
- 通过DNS，Shard等部署方式支持到1000万线？

eMQTT应用—Mobile Chat

Write WhatsApp + Facebook Messenger?



致谢

@joaohf @callbay @hejin1026 @desoulter @turtleDeng @Hades32 @huangdan @phanimahesh @dvliman @kevsmith @CrazyWisdom @wuming123057

开源中国　　美式咖啡　　喜力啤酒

他们说忘了摇滚有问题　　万能青年旅店　　The Seven Mile Journey

《公路之光》　　《杀死那个石家庄人》　　《Passenger’s Log, The Unity Fractions》

联系

Feng Lee <feng@emqtt.io>

GitHub: github.com/emqtt

Twitter: @emqtt