

Optimizing IoT Cloud Applications for Scalability:

**Leveraging RPC with Distributed Queues for
Seamless Operations**

Daniel Lombardi

DC - UFSCar, 2025
Orientador: Fredy Valente

Agenda

Introduction

Objectives

Theoretical Basis

Development

Homologation & Results

Conclusions

Introduction

Introduction

IoT & Cloud

- Volume, Velocity, Variety
- Real-Time and Low Latency
- Bidirectional Communication
- Faulty Connections
- Heterogeneous Workloads

Introduction

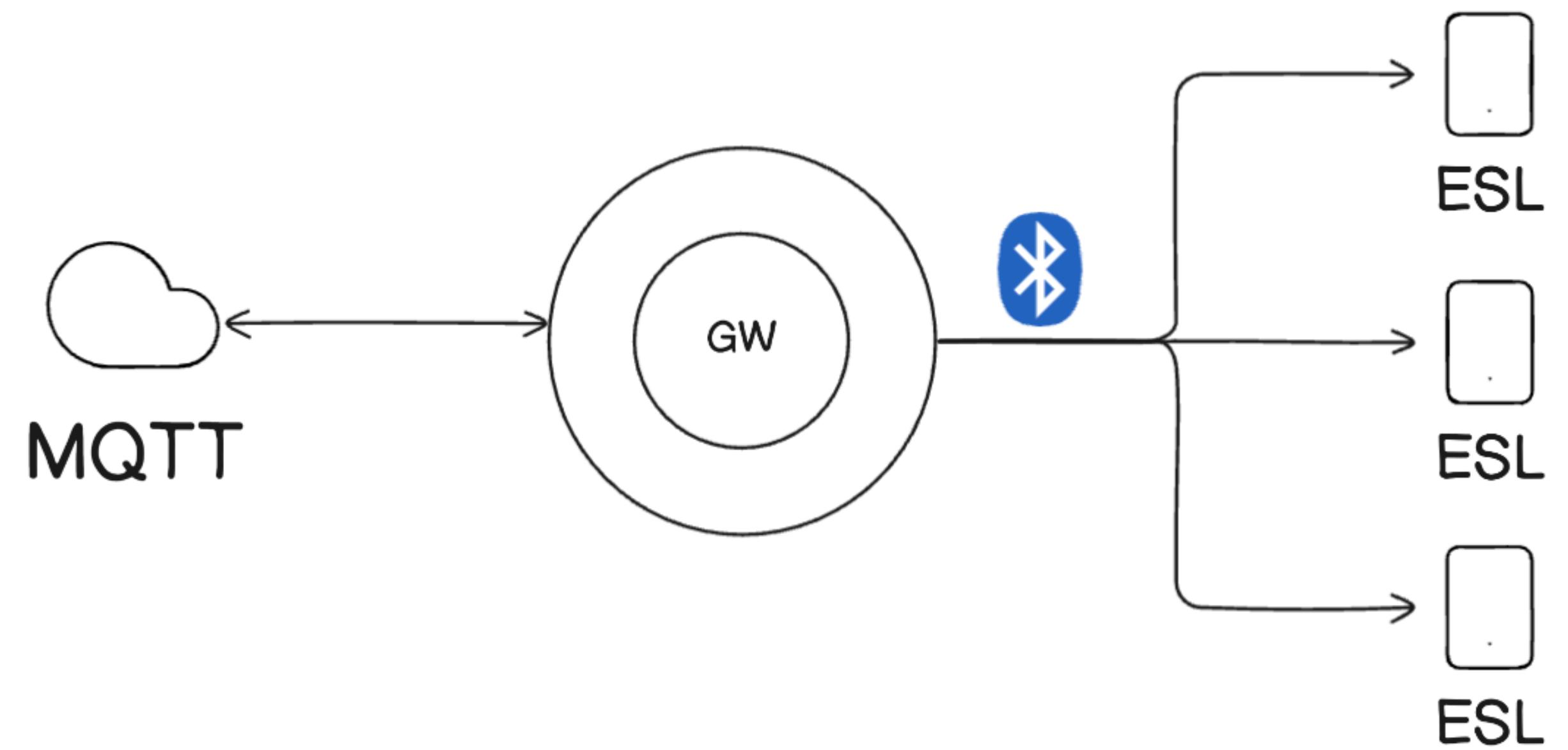
Infrastructure as a Service (IaaS) & Elasticity

- Pay-as-you-go Model
- Managed Components
- Horizontal Scalability

Introduction

The ESL Problem & MQTT Limitations

- Massive Scale
- Traditional Broker Hurdles
 - MQTT struggles with bidirectional communication and the request-reply pattern needed for RPCs when horizontal scaling is applied



Objectives

Objectives

Building a Scalable IoT Communication Backbone

- Building a Scalable IoT Communication Backbone
 - Develop a functional MVP for large-scale IoT device networks tested using ESLs
- Elastic and Scalable Architecture
- Serve as base and reference for future projects requiring a quickly deployable IoT communication backbone with minimal configuration, easily adaptable to other product niches

Theoretical Basis

Theoretical Basis

Theoretical Foundations

- IoT Ecosystems
 - **Sensors** Collect Data; **Actuators** execute physical actions (our focus)
 - **Gateways** act as intermediaries, enabling diverse protocols such as Bluetooth Low Energy (BLE) etc.
- Scalability
 - **Vertical Scaling** (Scale-Up): Increasing single machine capacity (limited, single point of failure)
 - **Horizontal Scaling** (Scale-Out): Distributing workload across multiple machines (virtually limitless, fault-tolerant, cost-effective)

Theoretical Basis

Core Technologies and Concepts

- Cloud Computing
 - **Definition:** On-demand access to shared, configurable computing resources
 - **Elasticity:** The system's ability to automatically adjust resources to match fluctuating workloads, optimizing performance and cost

Theoretical Basis

Core Technologies and Concepts

- Message Queues and Asynchronous Communication
 - **Message Brokers** (Queues): Producers send messages, Consumers receive them, in our case RabbitMQ
 - **Load Balancing** and **Buffering**: Handle traffic bursts, prevents consumer from being overwhelm.
 - **Asynchronous Communication**: Producers don't wait for execution, only broker (queue) confirmation, enables decoupled operation
 - **AMQP**: Open Standard for reliable, asynchronous message transfer
 - **MQTT**: Lightweight, publish-subscribe protocol ideal for IoT

Theoretical Basis

Core Technologies and Concepts

- Remote Procedure Call (RPC)
 - Allows programs to call producers
- In-Memory Data Stores
 - Extremely fast data access (low latency, high throughput)
 - Stores data in RAM, ideal for real-time applications and temporary storage
 - Highly amenable to horizontal scalability

Theoretical Basis

Methodology and Testing Approach

- Flexible Scalability
 - Custom Communication Backbone is designed for Cloud-Native and automatic adaptation to real-time, varying demand
- Comprehensive Testing Strategy
 - Combines quantitative performance measurements and qualitative operational observations
 - Latency, Throughput
 - Ease of setup, configuration overhead

Development

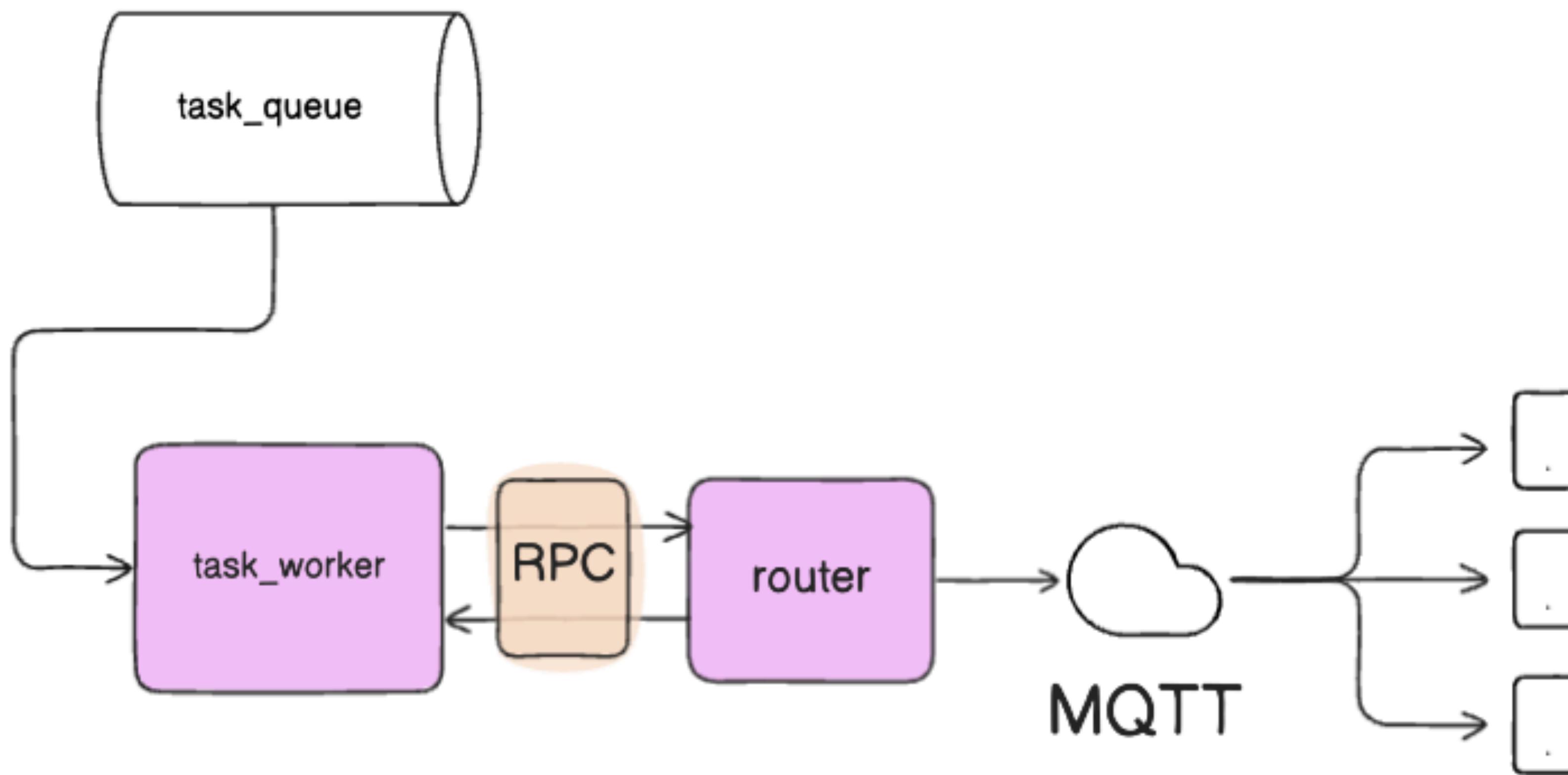
Development

General Architecture

- Async:
 - Long-Running Tasks: ESLs actions can take up to three minutes to complete
 - Decoupling and Resilience
 - Delivery Guarantees: Ensures at-least-once execution semantics

Development

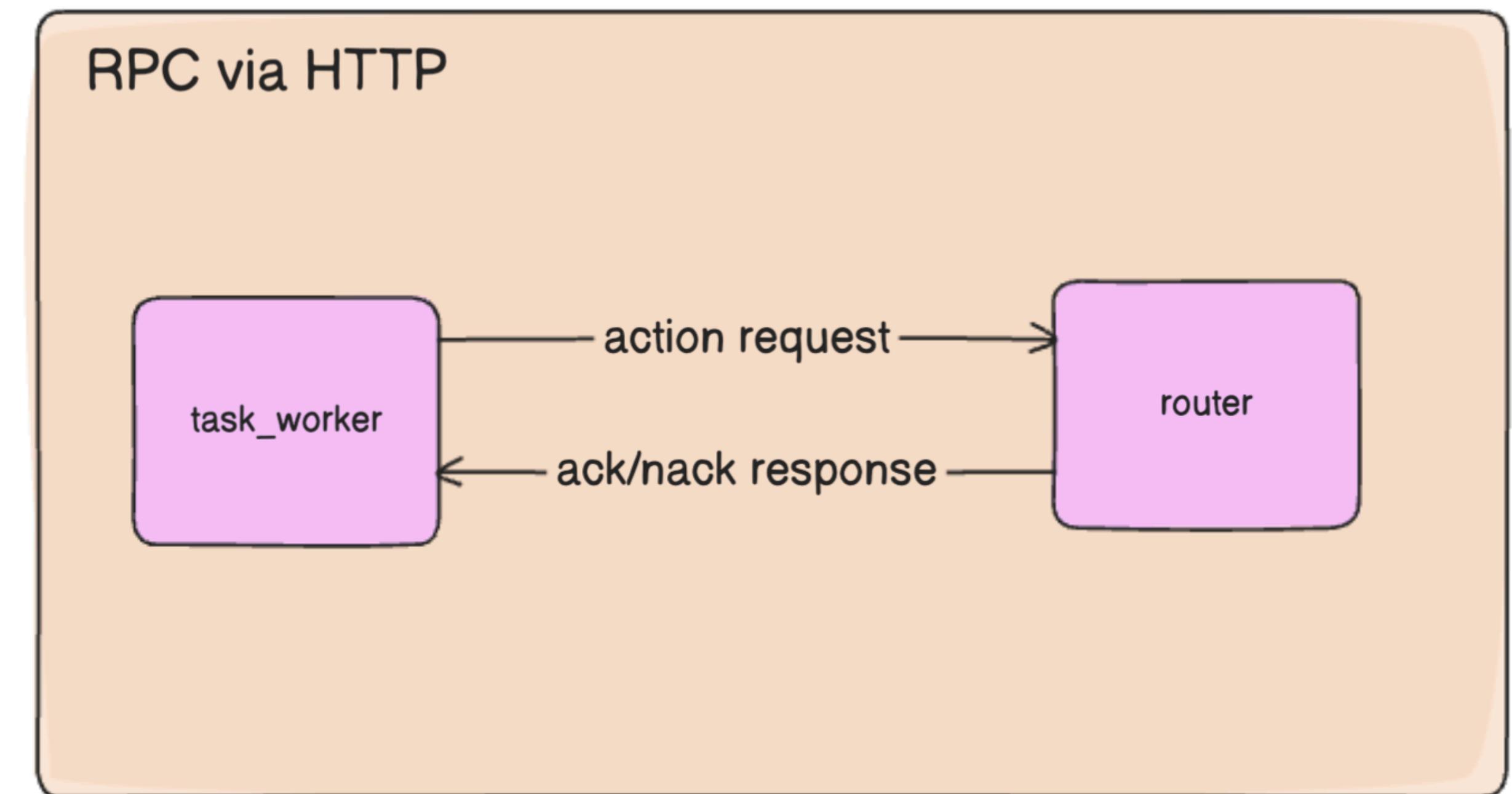
General Architecture



Development

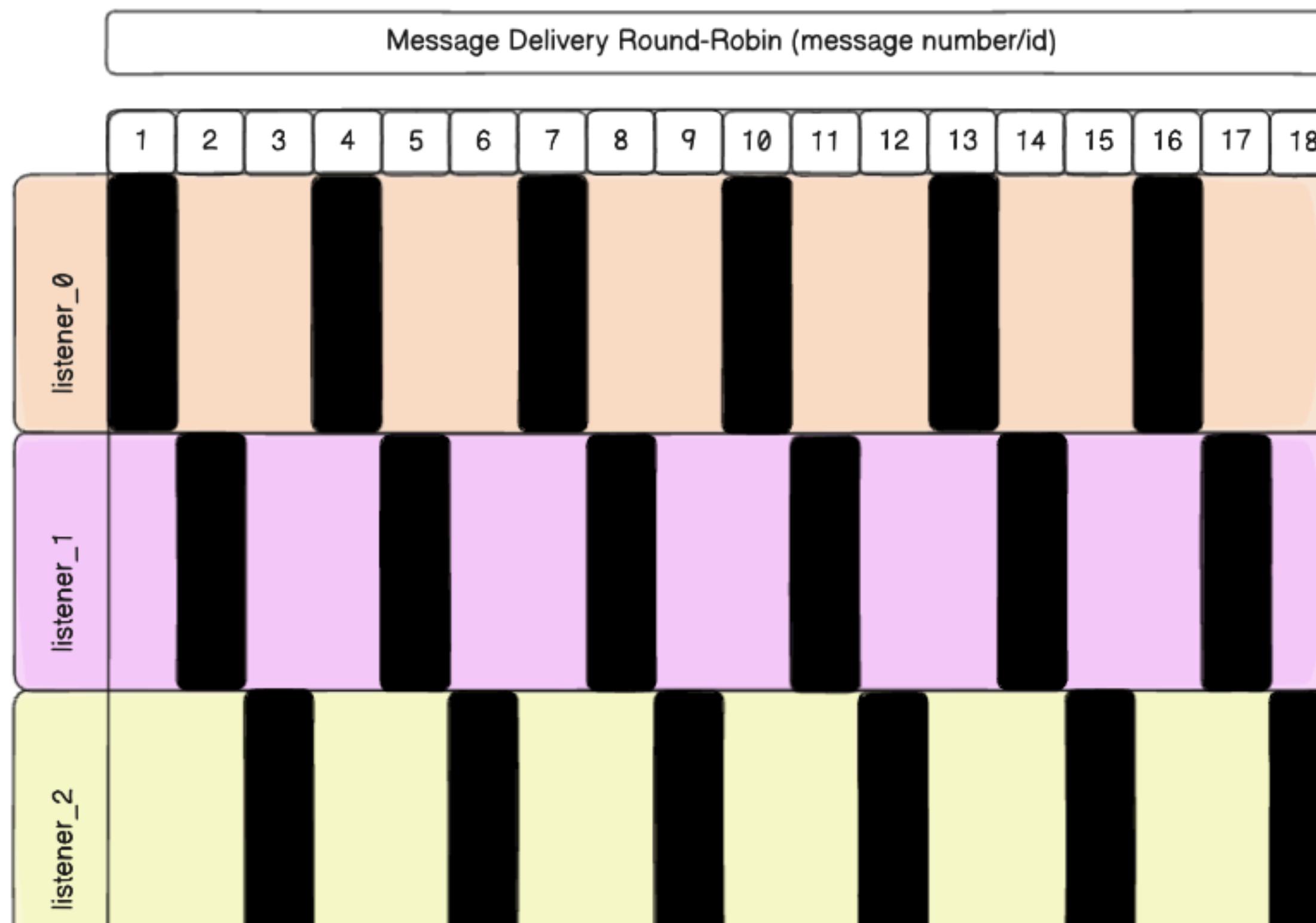
Baseline Backbone

- RPC via HTTP
- Limitation: The MQTT protocol's round-robin load balancing
 - Leads to misrouted requests, breaking the request-reply pattern
 - Forces static binding **1:1** between each **router** and **gateway**

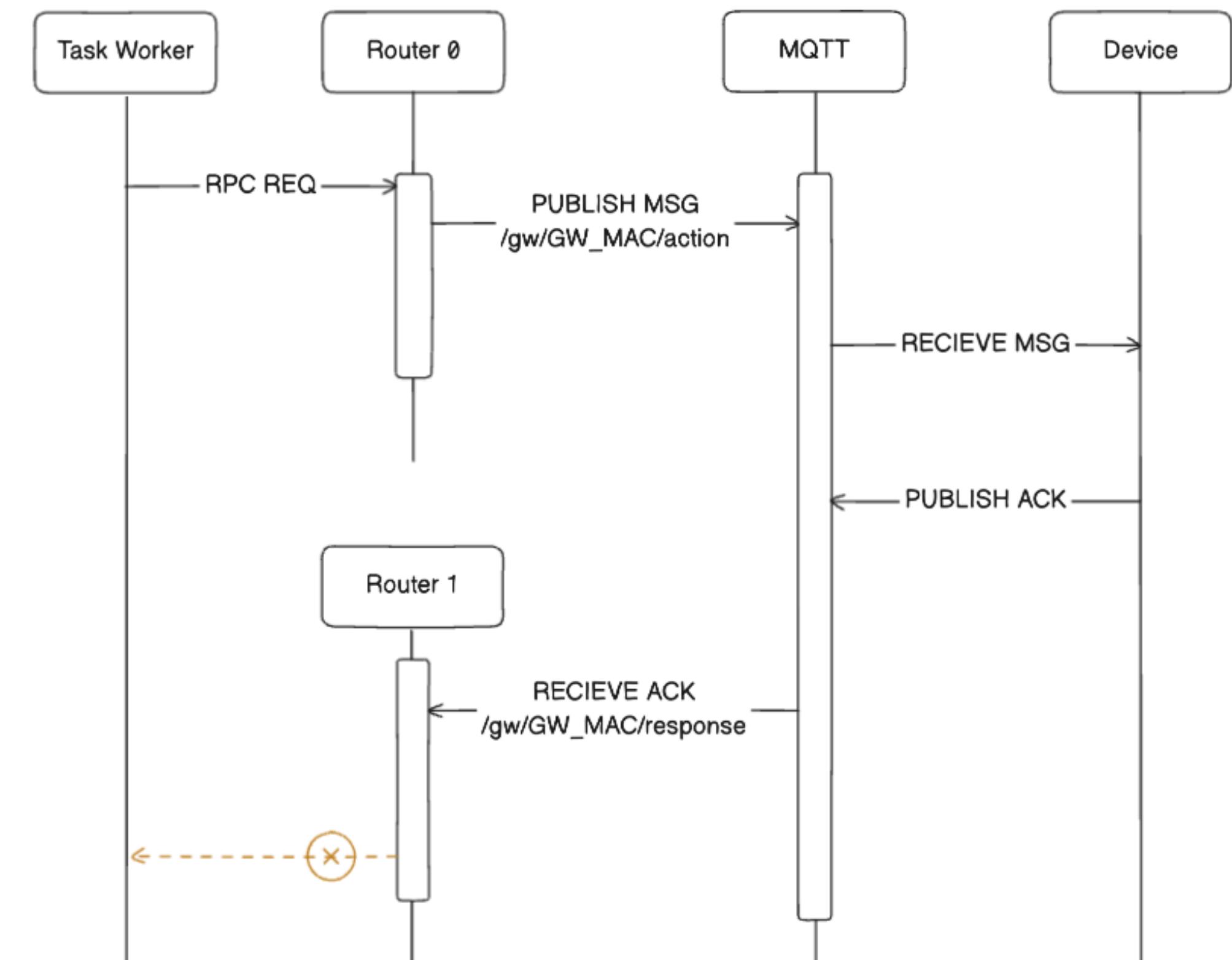


Development

Baseline Backbone



■ Message Delivered to Node

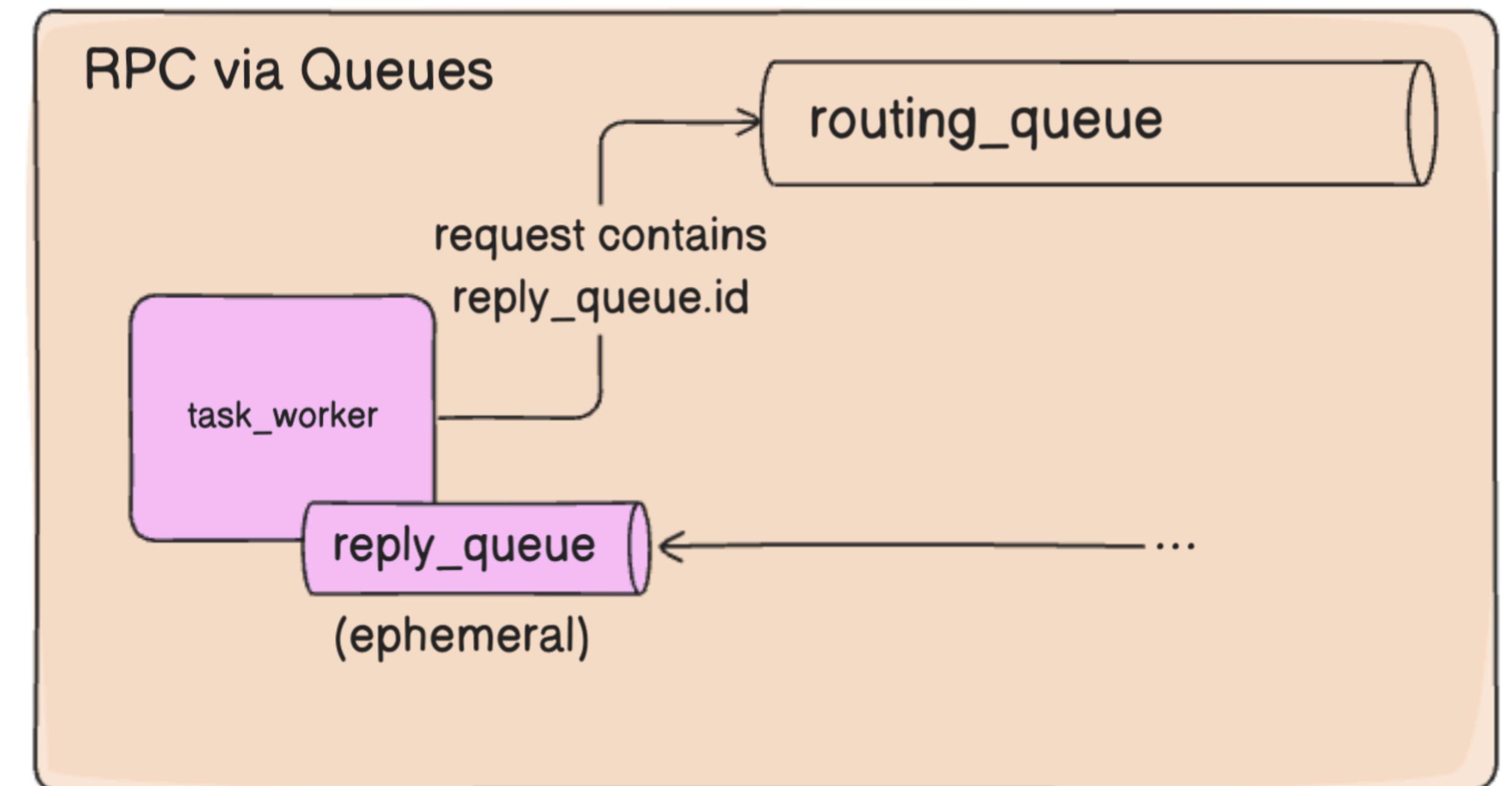


Router 0 and Router 1 are sharing a subscription (round-robin) to the topic /gw/GW_MAC/response

Development

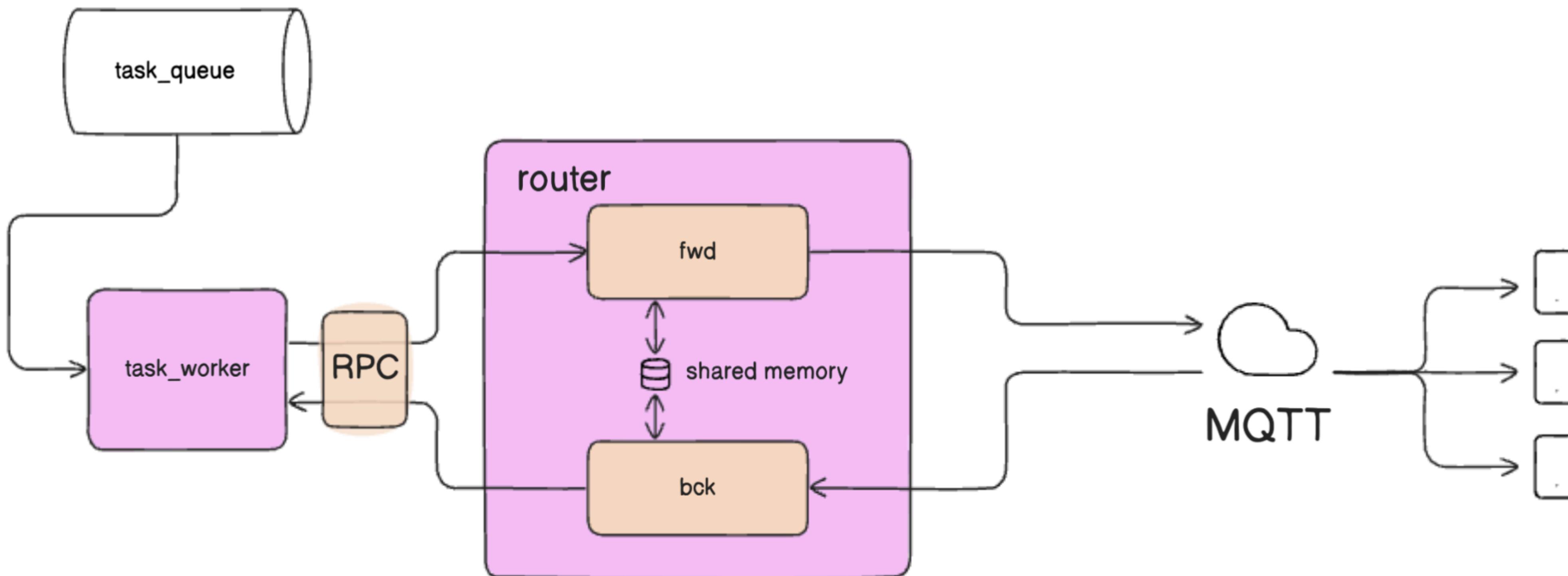
Custom Backbone

- Replies are directed to a specific reply queue using RequestId
- Eliminates sticky sessions, allowing multiple routers to share single gateways load
- Shared Memory via redis



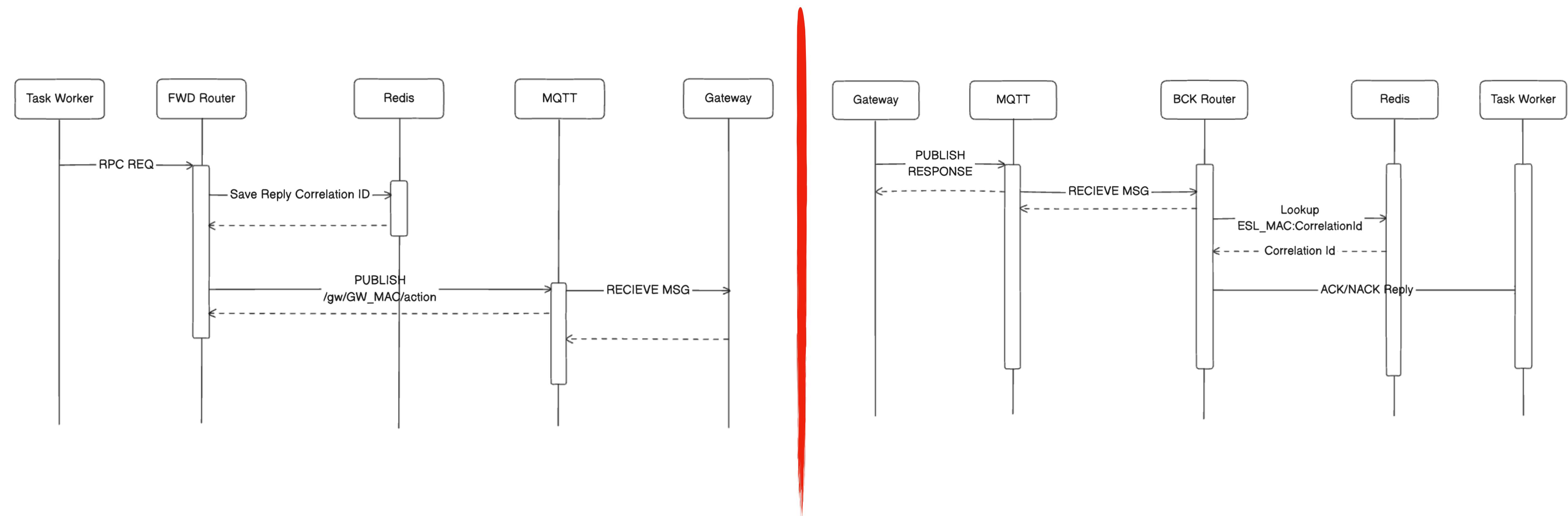
Development

Custom Backbone



Development

Custom Backbone



Development

Pseudo Code

```
● ● ●

// task_worker
func main() {
    for task := range rabbitMqMsgsChan {
        esls := dbService.BusinessLogic(tasks)

        var msgs []models.RoutingMessage
        for esl := range esls {
            m := models.RoutingMessage{}.FromEsl(esl)
            msgs = append(msgs, m)
        }

        reps := commsBackbone.Forward(msgs) // This method waits for replies
        if len(reps) != len(macs) {
            return errors.New("not all devices replied")
        }
    }
}
```

Development

Pseudo Code



```
// fwd_router
func main() {
    for routingMsg := range rabbitMqMsgsChan {
        sharedMem.Save(routingMsg.EslMac, routingMsg.CorrelationId)

        gwMac := dbService.GetRoute(routingMsg.EslMac)
        topic := "/gw/" + gwMac + "/action"

        mqtt.Publish(topic, routingMsg)
    }
}
```

Development

Pseudo Code



```
// bck_router
func onMsgCallback(rep models.RoutingReply) {
    correlationId := sharedMemService.RepKey(rep.EslMac)
    messagingService.Reply(correlationId, rep)
}

func main() {
    topic := "/gw/+response"
    mqtt.SetCallback("$share/router-bck-group/", onMsgCallback)
    mqtt.Start() // Hangs
}
```

Development

Pseudo Code



```
// task_worker
func main() {
    for task := range rabbitMqMsgsChan {
        esls := dbService.BusinessLogic(tasks)

        var msgs []models.RoutingMessage
        for esl := range esls {
            m := models.RoutingMessage{}.FromEsl(esl)
            msgs = append(msgs, m)
        }

        reps := commsBackbone.Forward(msgs) // This method waits for replies
        if len(reps) != len(macs) {
            return errors.New("not all devices replied")
        }
    }
}
```

Development

Pseudo Code



```
// fwd_router
func main() {
    for routingMsg := range rabbitMqMsgsChan {
        sharedMem.Save(routingMsg.EslMac, routingMsg.CorrelationId)

        gwMac := dbService.GetRoute(routingMsg.EslMac)
        topic := "/gw/" + gwMac + "/action"

        mqtt.Publish(topic, routingMsg)
    }
}
```

Development

Pseudo Code



```
// bck_router
func onMsgCallback(rep models.RoutingReply) {
    correlationId := sharedMemService.RepKey(rep.EslMac)
    messagingService.Reply(correlationId, rep)
}

func main() {
    topic := "/gw/+response"
    mqtt.SetCallback("$share/router-bck-group/", onMsgCallback)
    mqtt.Start() // Hangs
}
```

Homologation & Results

Homologation and Results

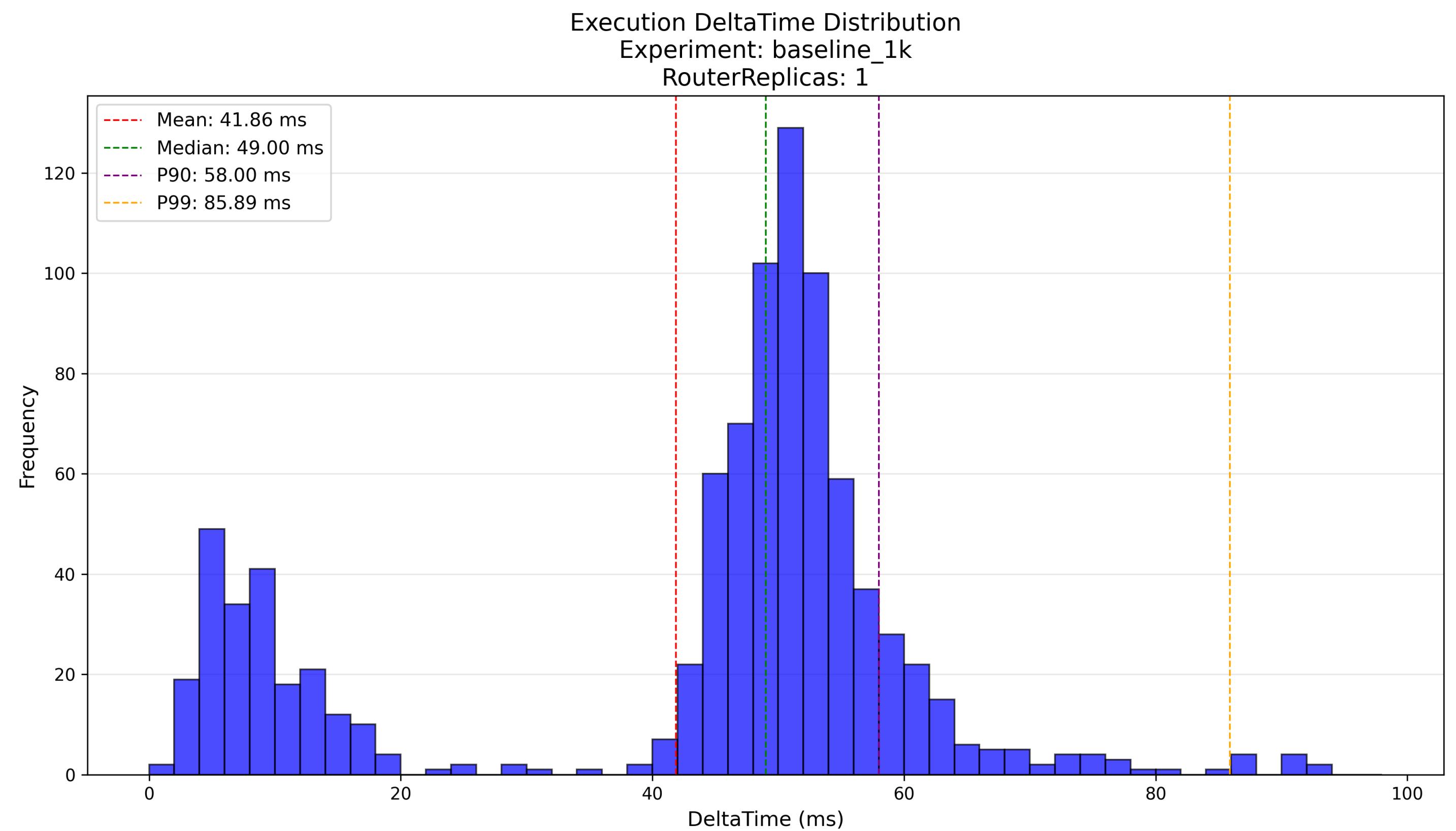
Testing Setup & Methodology

- Test Environment
 - 12-core machine, 16GB RAM
 - **Router Elements:** CPU capped at 100mCPU per replica
 - Other systems had no resource limits
- Load Generation
 - 50 concurrent workers to load routers; 60s timeout
- Data Collection
 - Telemetry on worker-side to collect end-to-end latency

Homologation and Results

Functional Tests - Baseline System

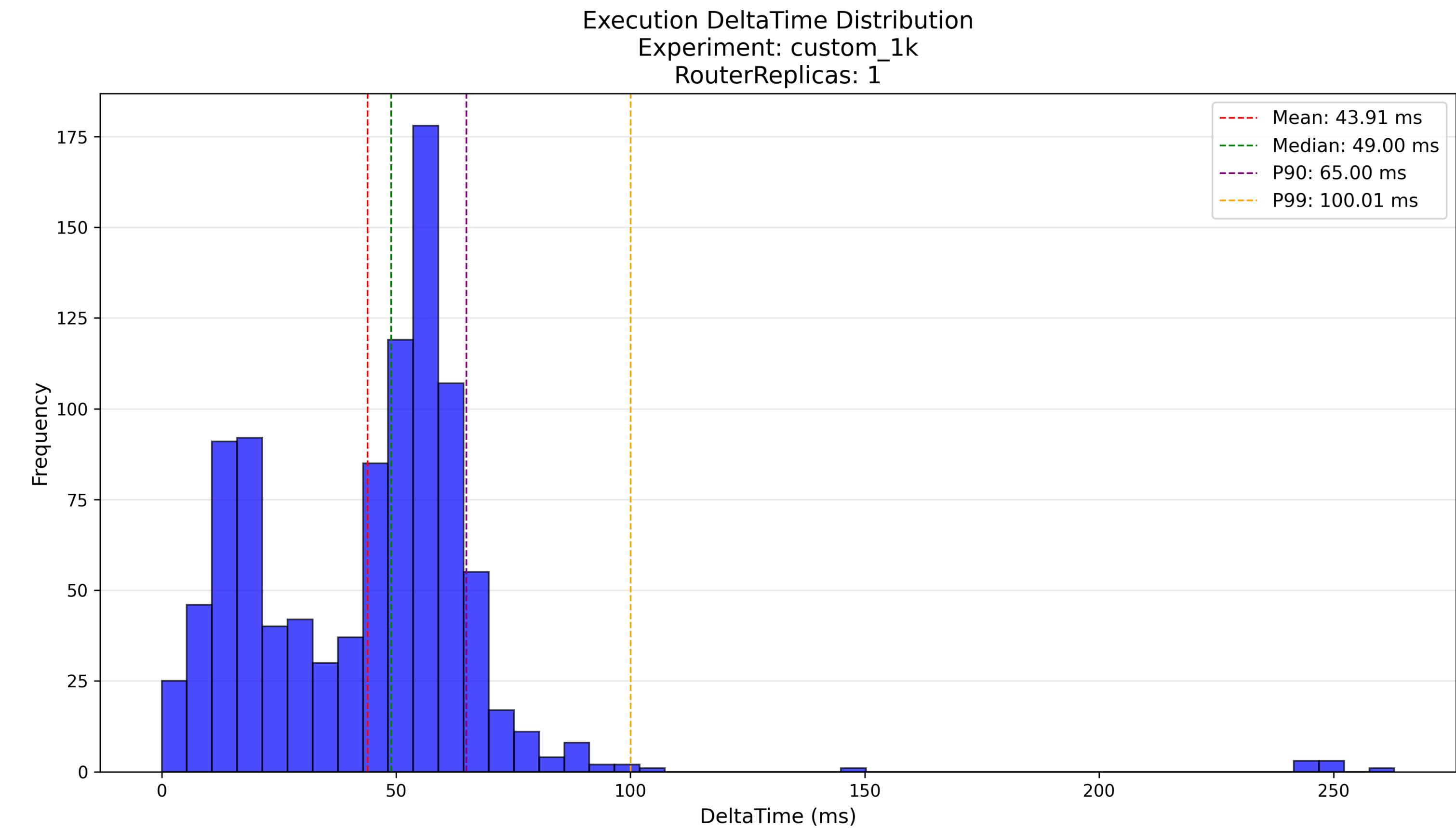
- Verifying Core Operations
(1k tasks, 1 Router Replica)
 - Confirm the systems correctly follow the architectural sequence diagram



Homologation and Results

Functional Tests - Custom System

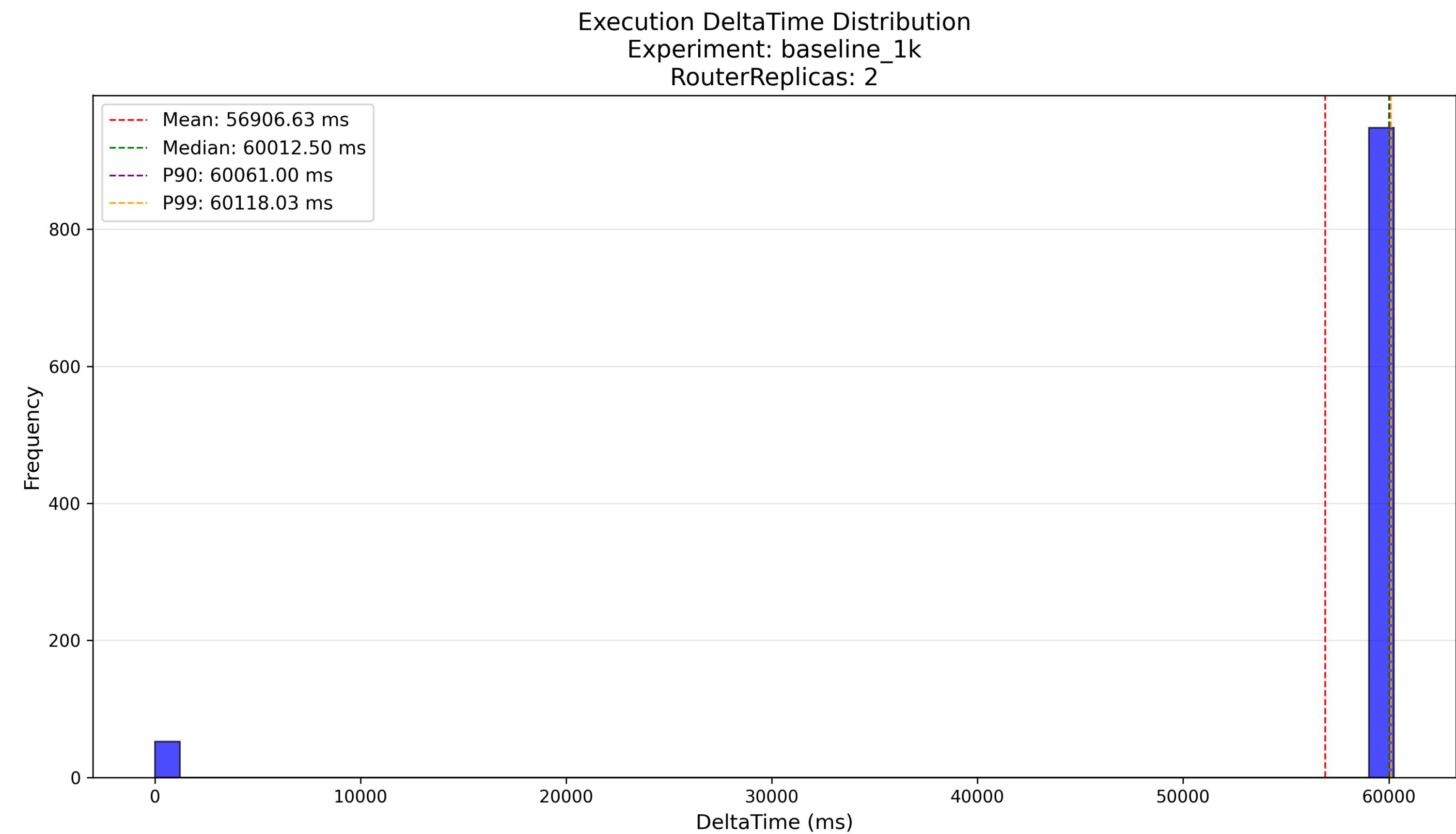
- Verifying Core Operations
(1k tasks, 1 Router Replica)
 - Confirm the systems correctly follow the architectural sequence diagram



Homologation and Results

Performance/Scalability Tests - Baseline System

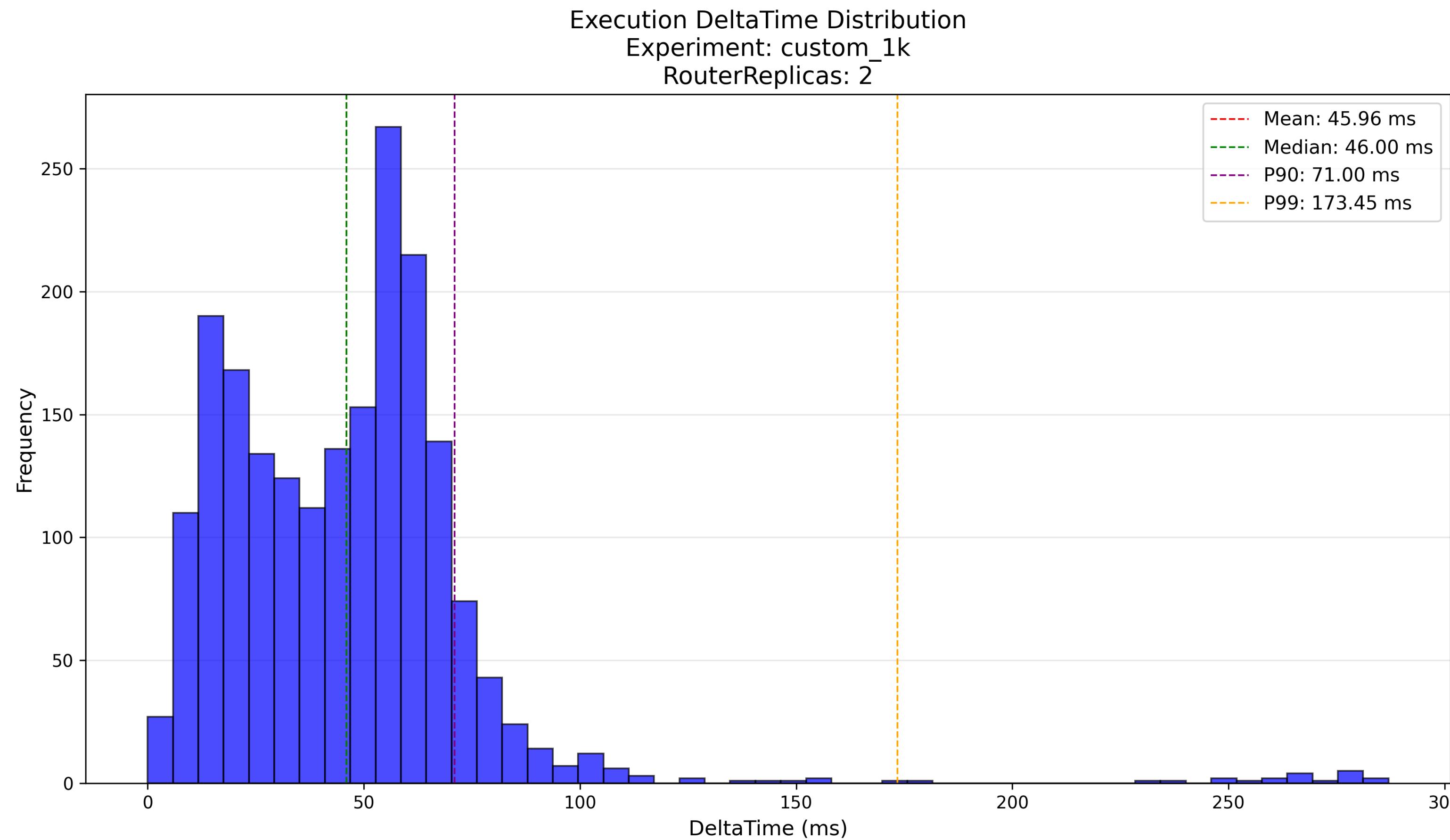
- Verifying Scalability (1k tasks, 2 Router Replicas)
 - **Misrouting Pitfall:** Requests timed out



Homologation and Results

Performance/Scalability Tests - Custom System

- Verifying Scalability (1k tasks, 2 Router Replicas)
 - Correctly routes replies to the requesting worker



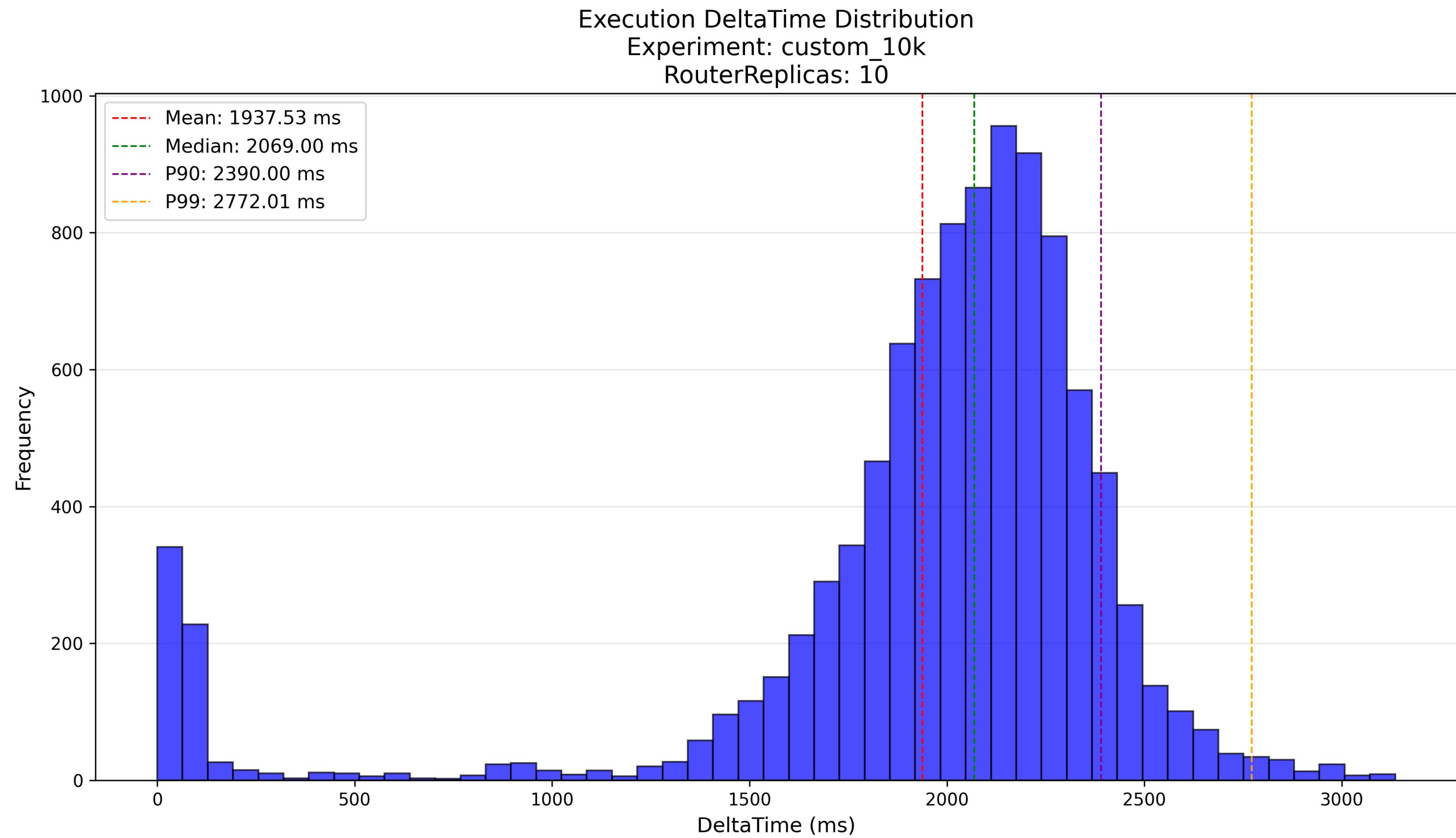
**WE WILL NOT CONTINUE PERFORMANCE TESTS
ON THE BASELINE SYSTEM**



Homologation and Results

Performance/Scalability Tests - Baseline System

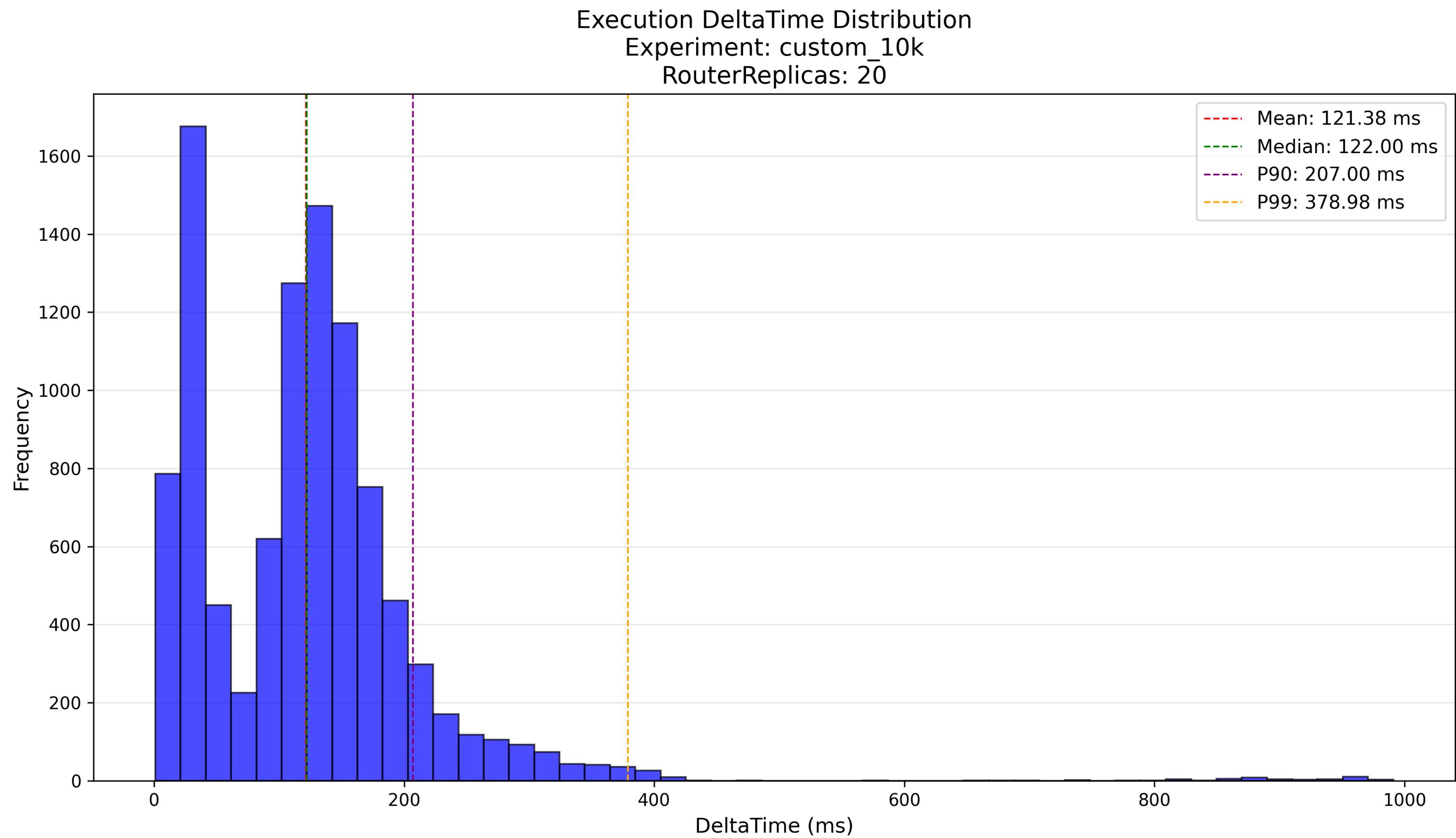
- Verifying Performance
(10k tasks, 2 Router
Replicas)
 - Increase in latency,
system starting to
bottleneck



Homologation and Results

Performance/Scalability Tests - Baseline System

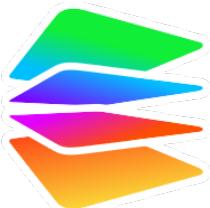
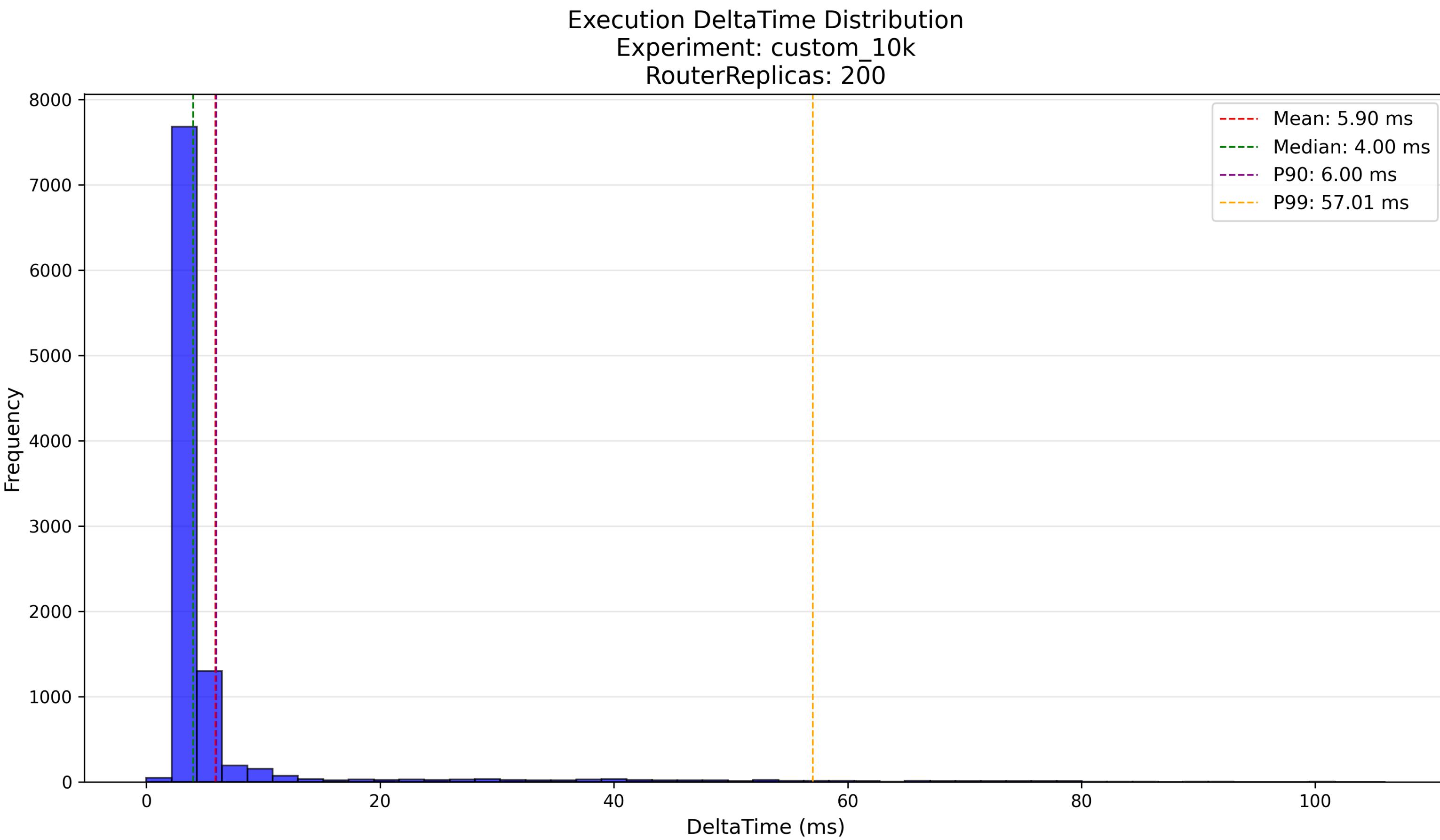
- Verifying Performance
(10k tasks, 20 Router
Replicas)
 - Latency reduced back
to acceptable values



Homologation and Results

Performance/Scalability Tests - Baseline System

- For Fun Experiment,
Stretch (10k tasks, 200
Router Replicas)
 - Latency greatly reduced



Conclusions

Conclusions

Achieving a Scalable and Cloud-Native IoT Communication Backbone

- MVP Success
 - Developed System fully executes all architectural sequences
 - Exceptional Scalability, true Elasticity
- Cloud-Native Design
 - Developed using modern, well-established technologies (Go, RabbitMQ, Redis, MQTT, Docker)
 - Strategic choice makes the system Cloud-Native by design

Thank You

Questions?

IoT Communication Backbone

Demo

github.com/LombardiDaniel/TCC