Monitoramento de radiação utilizando MOM

Equipe:

Arnaldo Morais

Gabriel Gadelha

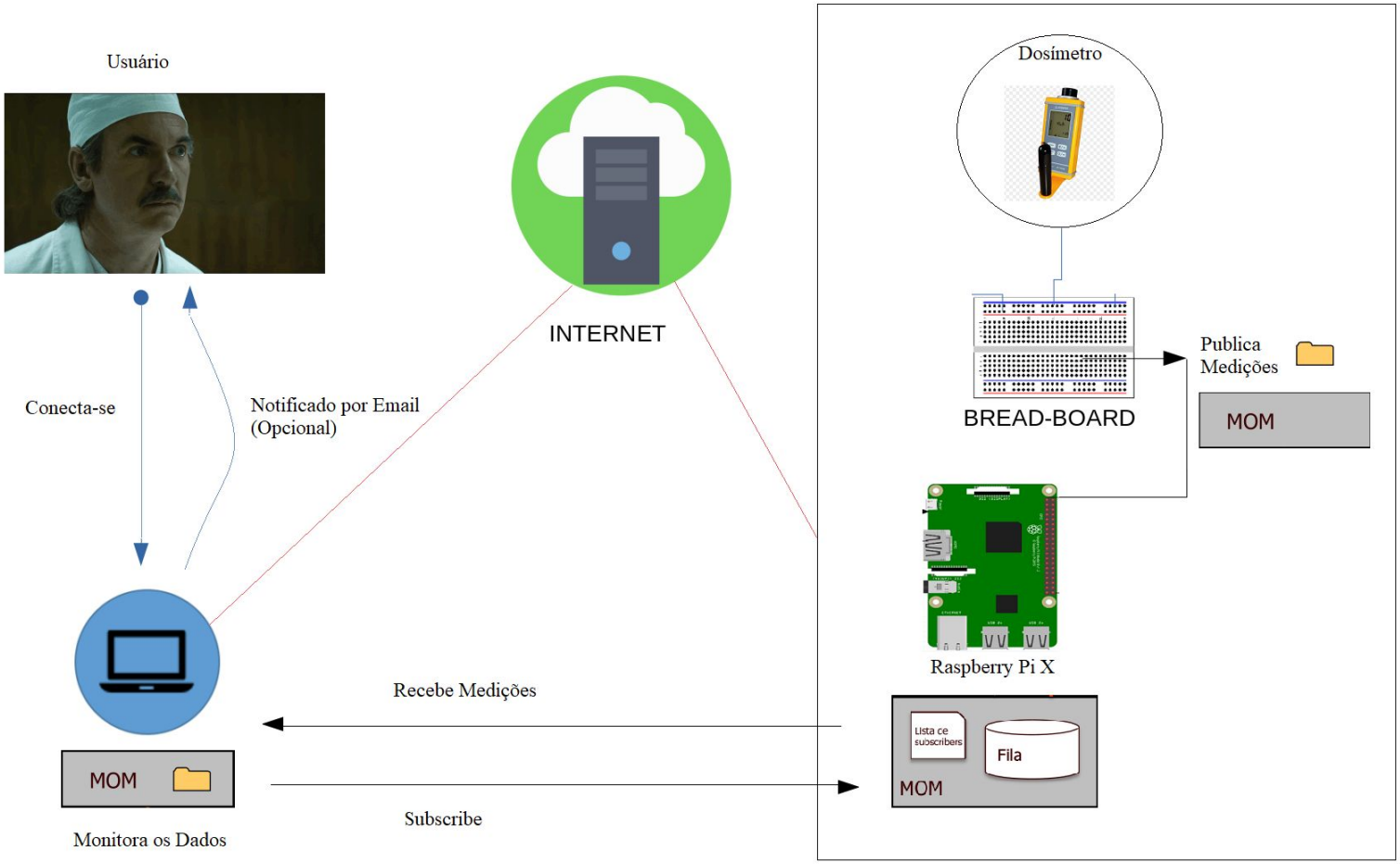
Objetivo

- **Implementar um middleware baseado em MOM para ser usado em uma Raspberry Pi X**

Cenário de uso

- **Um sensor de radiação envia suas medições ao Middleware, implementado na placa. Um host se inscreve no serviço de mensageria e passa a receber o conteúdo desejado. A aplicação do host monitora medições e poderá alertar ao usuário, via email, de circunstâncias críticas (radiação elevada).**

Cenário de uso



Requisitos

- **Modelo**
  - Middleware orientado a mensagens
- **Transparências**
  - Acesso e localização

Requisitos

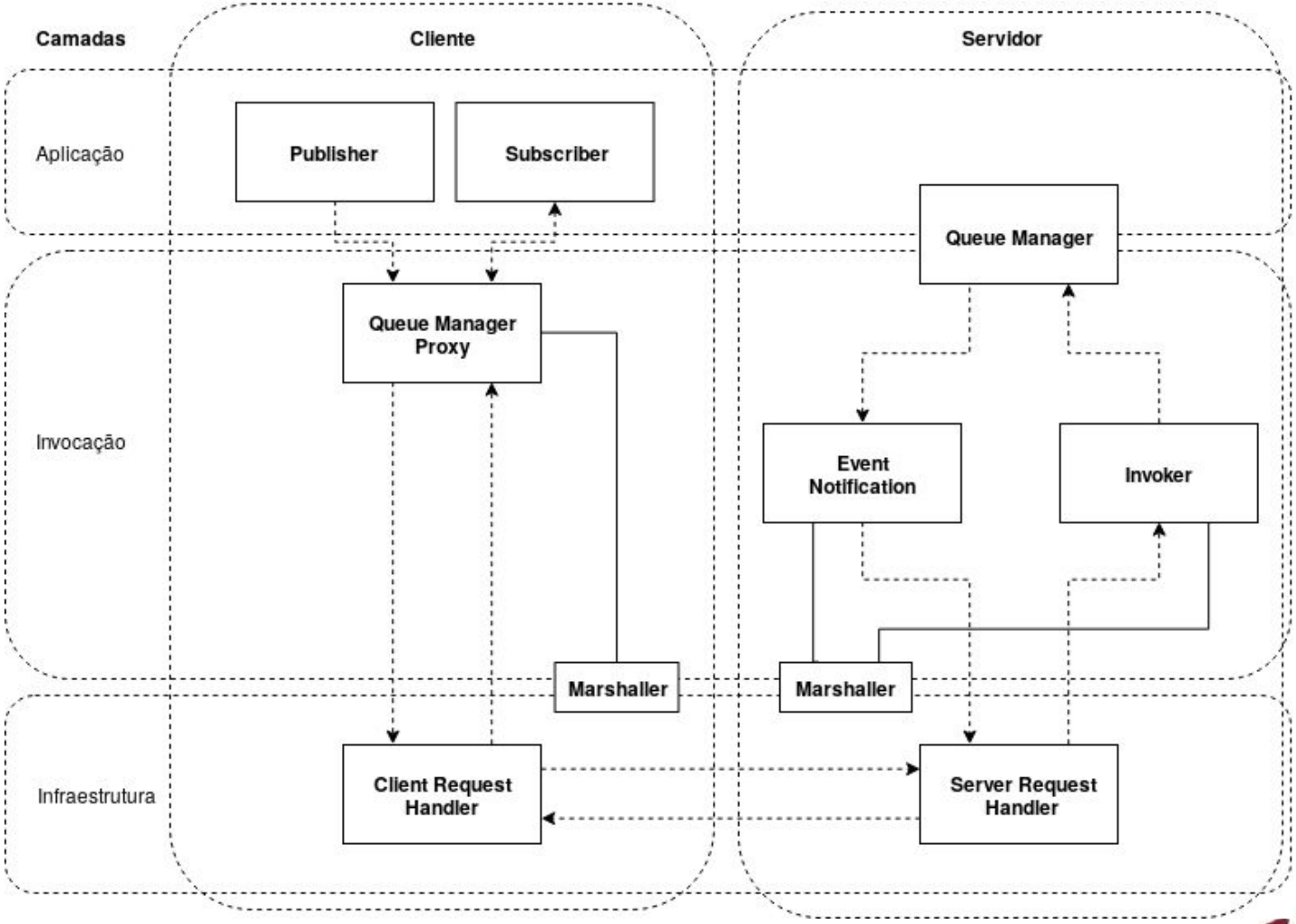
#	Descrição do Requisito Funcional
RF01	Gerenciar Filas
RF02	Serializar Dados
RF03	Definir o protocolo de Middleware
RF04	Gerenciar publishers/subscribers
RF05	Gerenciar perda de mensagens

Requisitos

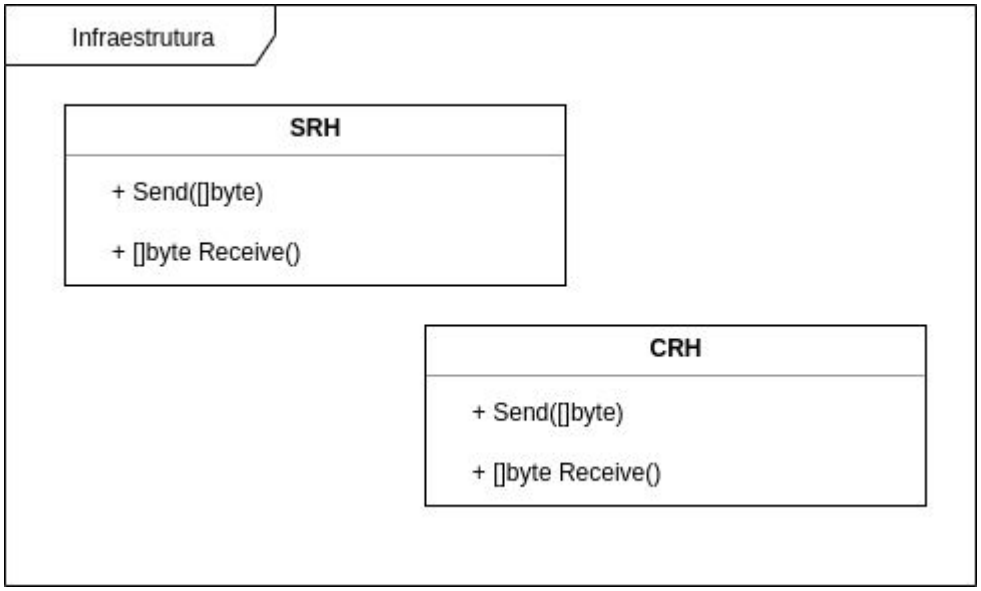
#	Descrição do Requisito Não-Funcional
RNF01	A fim de reduzir o consumo de energia, as requisições do sensor terão uma taxa máxima de 6 mensagens por minuto (ou 0.1/s)
RNF02	*Offline First* - As mensagens do sensor que não forem enviadas devido a erros de conexão, serão armazenadas em uma fila com capacidade de 100 mensagens.

Fim Parte 1

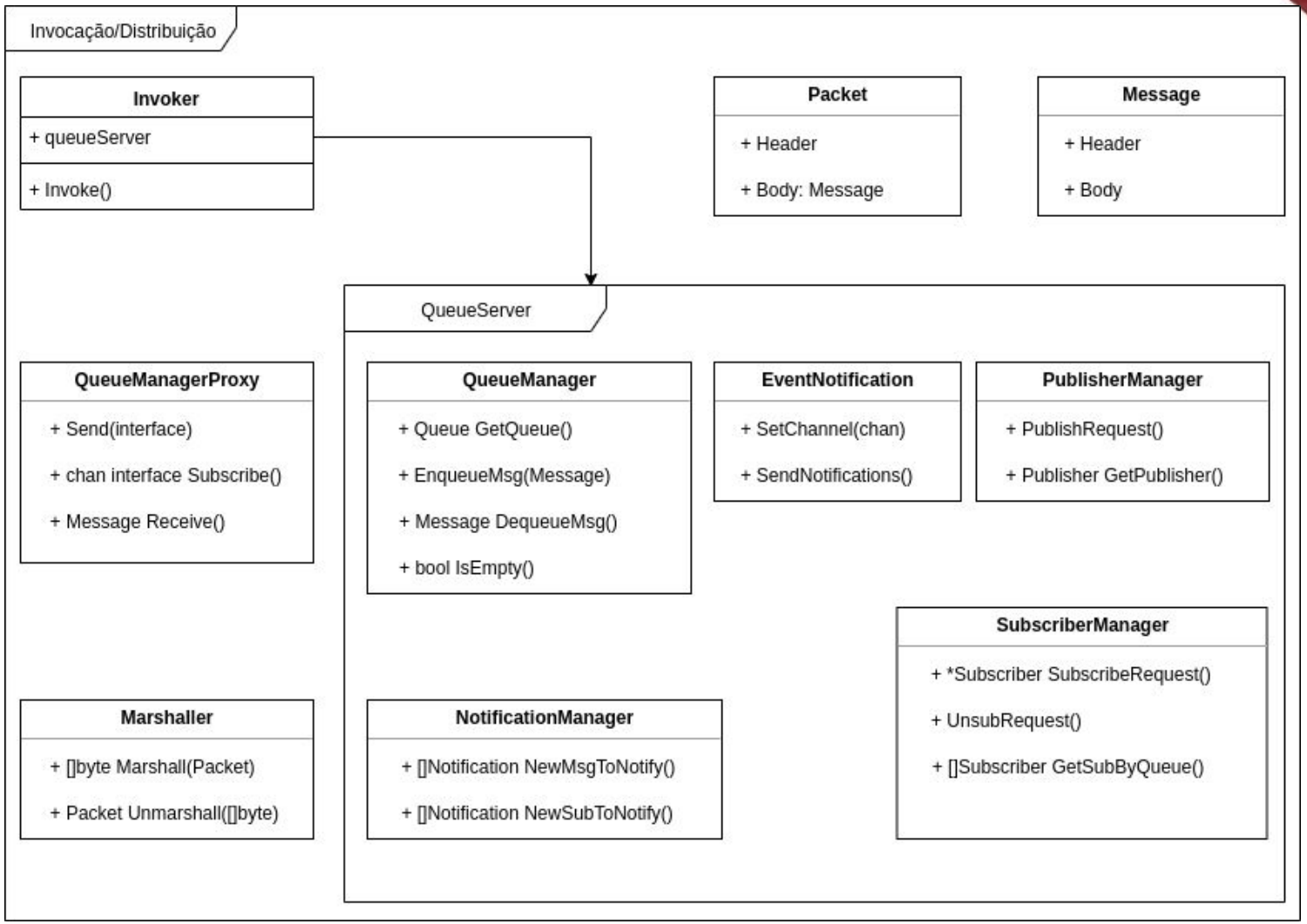
Arquitetura



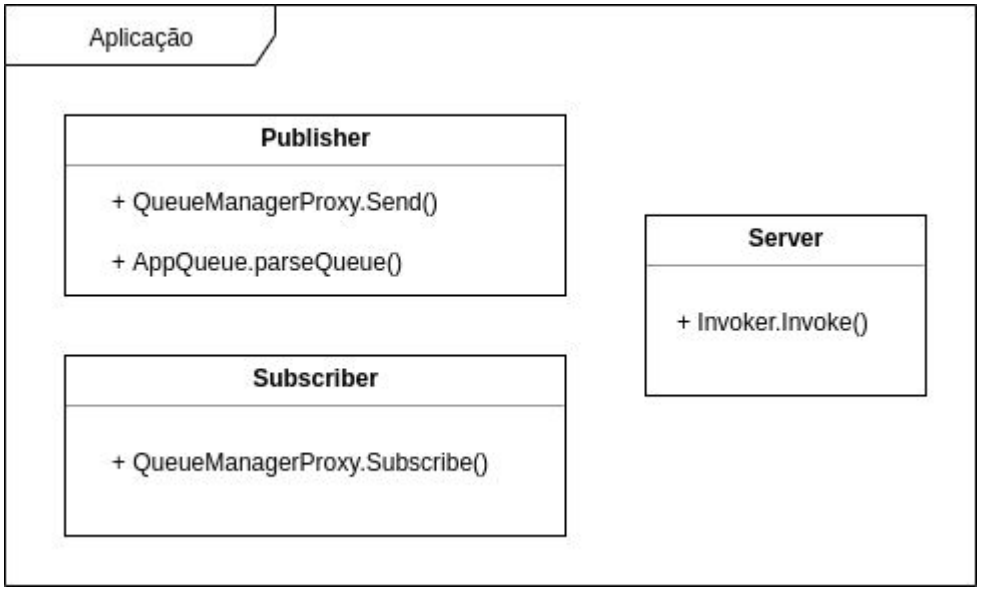
Projeto - Infraestrutura



Projeto - Invocação/Distribuição



Projeto - Aplicação



Implementação

- **Tecnologia utilizada**
  - Raspberry Pi 3
  - Golang 1.13
  - Kit arduino

Implementação

- **Operador**
  - Criando Proxy e fazendo Subscribe

```
// Rad Queue
radQueueProxy := queue.QueueManagerProxy{
    Host: OPERATOR_HOST,
    Port: OPERATOR_PORT,
    QueueName: RADIATION_QUEUE
}
radChannel := radQueueProxy.Subscribe()
go GetRadiation(radChannel, dangerRadChannel)
```

Implementação

- **Sensor**
  - Criando Proxy para Publicar

```
// Radiation Publish Queue
radQueueProxy := queue.QueueManagerProxy{
    Host: SENSOR_HOST,
    Port: SENSOR_PORT,
    QueueName: RADIATION_QUEUE
}
radQueueProxy.Send("publishRequest", nil)
go PublishRadiation(radQueueProxy)
```

  - Publicando

```
sensorqueue.Enqueue(radValue)
if hasConnection(proxy) {
    parseQueue(proxy)
} else {
    fmt.Println("Sem conexão")
}
```

```
// Garantir taxa máxima
time.Sleep(shared.REAL_TIME)
```

Implementação

- **QueueManagerProxy - Subscribe**
  - Subscribe

```
func (proxy QueueManagerProxy) Subscribe() chan interface{}
```

```
crh := crh.CRH{
    ServerHost: shared.QUEUE_SERVER_HOST,
    ServerPort: shared.QUEUE_SERVER_PORT
}
marshaller := marshaller.Marshaller{}
```

Implementação

- **QueueManagerProxy - Subscribe**
  - Criando Mensagem e Pacote

```
operation := "subscribe"
```

```
msgHeader := message.MessageHeader{
    Host: proxy.Host,
    Port: proxy.Port,
    Destination: proxy.QueueName,
    ExpirationDate: msgTimeValidation,
}
msg := message.Message{Header: msgHeader}
```

```
pkt := packet.Packet{}
pkt.Header = packet.PacketHeader{Operation: operation}
pkt.Body = packet.PacketBody{Message: msg}
```

```
err := crh.Send(marshaller.Marshal(pkt))
```

Implementação

- **QueueManagerProxy - Subscribe**
  - Ouvindo notificações

```
// Notification Channel
contentChannel := make(chan interface{}, 100)
```

```
go func() {
    for {
        msgReceived := proxy.Receive()
        content := msgReceived.Body.Content
        contentChannel <- content
    }
}()
```

```
return contentChannel
```

Implementação

- **QueueManagerProxy - Send**
  - Idêntico ao processo de Subscribe, porém sem a criação de canal.

Implementação

- **Invoker**

```
srhImpl := srh.SRH{
    ServerHost: invoker.Host,
    ServerPort: invoker.Port
}
marshallerImpl := marshaller.Marshaller{}

queueServer := queue.QueueServer{}
```

```
// Queue Server Managers
subManager := &queueServer.SubManager
pubManager := &queueServer.PubManager
queueManager := &queueServer.QueueManager
notifManager := &queueServer.NotifManager
eventNotifManager := &queueServer.EventNotifManager
```

```
// control loop
for {
    // receive request packet
    rcvPktBytes, err := srhImpl.Receive()
```

Implementação

- **Invoker - Publish Request**

```
case "publishRequest":
    // Publish to Queue
    pubManager.PublishRequest(host, port, dest)
```

```
type PublisherManager struct {
    PubList []Publisher
}
```

```
type Publisher struct {
    Host string
    Port int
    PublishQueues map[string]bool
}
```

Implementação

- **Invoker - Subscribe**

```
case "subscribe":
    // Subscribe to Queue
    sub := subManager.SubscribeRequest(host, port, dest)
```

```
// Create Notifications for New Subscriber
queue := queueManager.GetQueue(dest)
notifList := notifManager.NewSubscriberToNotify(*sub, *queue)

for _, notif := range notifList {
    notifChannel <- notif
}
```

Implementação

- **Invoker - Subscribe**

```
// Execute Nofitication Event Sender
notifChannel := make(chan queue.Notification, 1000)
eventNotifManager.SetChannel(notifChannel)
go eventNotifManager.SendNotifications()
```

Implementação

- **Invoker - Publish**

```
case "publish":
```

```
//Check if Publisher is authorized
publisher, exists := pubManager.GetPublisher(host, port)
```

```
// Publish to Queue
queueManager.EnqueueMsg(msgReceived)
```

```
subList := subManager.GetSubscribersByQueue(dest)
notifList := notifManager.NewMessageToNotify(msgReceived, subList)

for _, notif := range notifList {
    notifChannel <- notif
}
```

Implementação

- **EventNotification**

```
func (manager *EventNotification) SendNotifications()
```

```
subList := subManager.GetSubscribersByQueue(dest)
notifList := notifManager.NewMessageToNotify(msgReceived, subList)

for _, notif := range notifList {
    notifChannel <- notif
}
```

```
pktBytes := marshallerImpl.Marshal(pkt)
err := subSRH.Send(pktBytes)
if err != nil {
    manager.NotifChannel <- notif
}
```

Implementação

- **QueueServer**

```
// Listen for publish/subscribe requests
go startQueueServerInstance()
```

```
func startQueueServerInstance() {
    queueInvoker := invoker.QueueInvoker{
        Host:shared.QUEUE_SERVER_HOST,
        Port:shared.QUEUE_SERVER_PORT
    }
    queueInvoker.Invoke()
}
```

Avaliação de Desempenho

- **Comparação do nosso middleware com o RabbitMQ**
  - **Métrica**: Tempo de lógica do middleware
  - A efeito de teste, o operador, sensor e servidor de filas, foram executados na Raspberry e o limite da taxa foi retirado.
  - **Funcionamento**: O cliente irá publicar sucessivamente no servidor de filas. Nas mensagens publicadas, haverá o timestamp de envio. Assim que o servidor for notificado, ele marcará o tempo. Portanto, o tempo em questão será a subtração do tempo do servidor menos o timestamp do cliente.

Avaliação de Desempenho

Parâmetros do Sistema	Valor
Processador	4× ARM Cortex-A53, 1.2GHz
Memória	1GB LPDDR2 (900 MHz)
WiFi	Desligado
Bluetooth	Desligado
Sistema Operacional	Raspbian Buster Lite

Parâmetros da Carga de Trabalho	Níveis
Operação remota	Publish
Número de invocações	1000

Avaliação de Desempenho - Resultados

Conclusão

- **Pontos positivos: Abstrai em muito a complexidade da implementação de uma aplicação distribuída; Assincronismo; Open source.**
- **Pontos negativos: Não é resiliente; Não é rápido (Gargalos ainda não identificados).**
- **Aprendizagem: Entregar assincronismo é complicado. O middleware orientado à mensagem possui mais liberdade para o uso de padrões de projeto. Isso dificulta no desenvolvimento, a respeito das decisões tomadas.**

Observações

- As equipes precisarão apresentar e discutir a proposta de projeto antes do início da implementação
- Projeto precisa ser implementado em Go
- Equipes com até 03 (três) integrantes
- Data de Entrega
  - 13/11
- Entregáveis
  - Slides
  - Código
- Estes slides devem ser preenchidos de forma incremental, funcionarão como documentação do projeto e deverão ser utilizados para o acompanhamento de cada uma das etapas de desenvolvimento do projeto
- No dia da apresentação do projeto, estes slides devem ser usados na apresentação
- Cada equipe terá até 20 minutos para apresentar o projeto e demonstrar o middleware executando

Observações

- **Critérios de Avaliação**
  - Qualidade técnica (40%)
  - Alinhamento com os conceitos da disciplina (30%)
  - Qualidade/clareza da apresentação (20%)
  - Avaliação de Desempenho (10%)
  - **Inovação (10%) [Bônus]**
    - **Equipes que fizerem projetos na área de Sensores, IoT, SDN e Nuvem ganham este bônus automaticamente.**

Fim dos Slides