Projeto de Middleware para Vanets

Eduardo Maia

Objetivo

- Middleware para IoV
- Implementar um middleware “do zero”
- Middleware Orientado a Mensagem

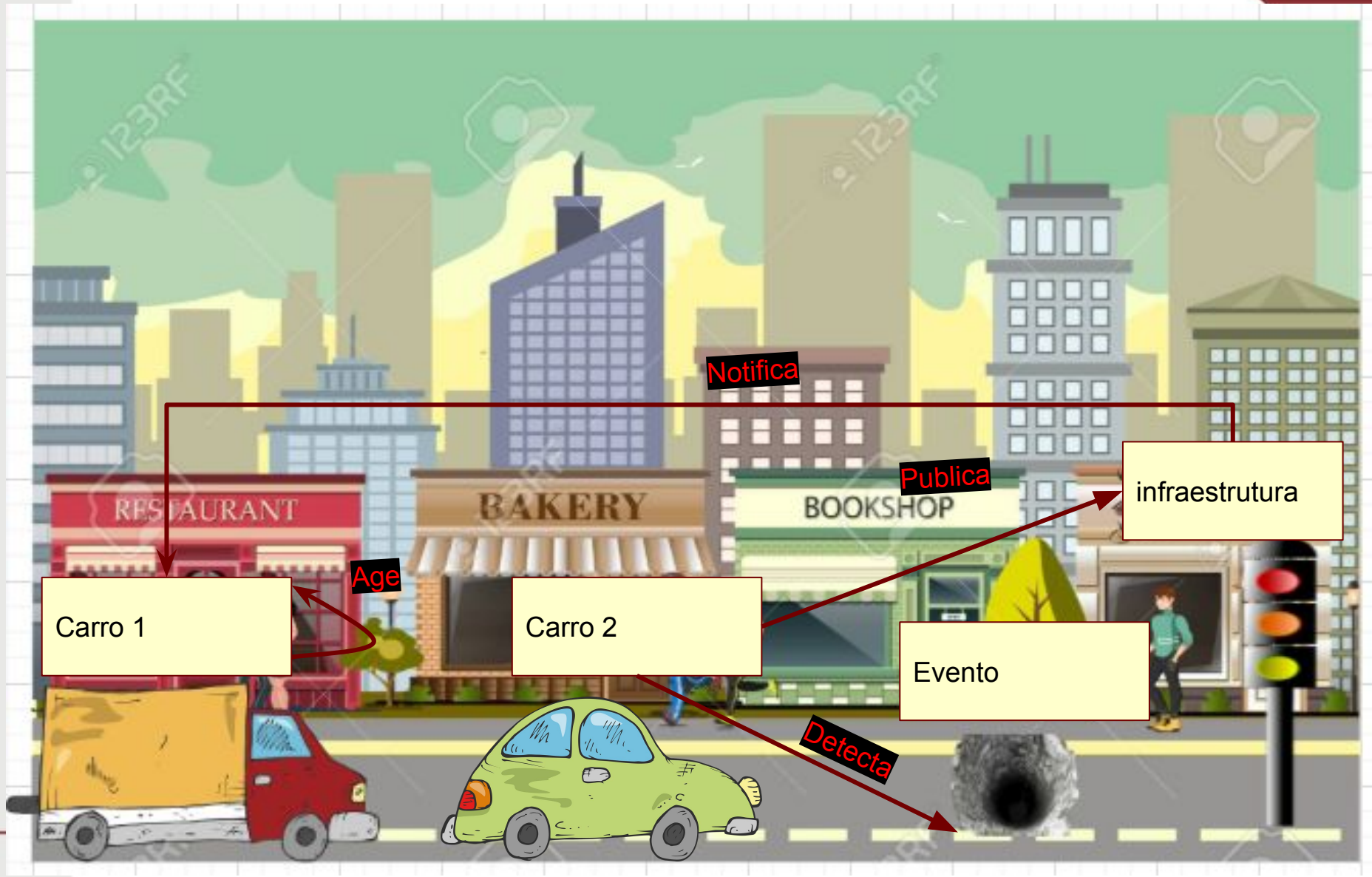
Background

- **IoV - Rede distribuída que conecta carros a motoristas, pedestres, e outras públicas de carros conectados (VANETS)**
  - V2V, V2I, V2P, V2X
- **Carros são equipados com sensores e unidades capazes de processamento**
- **Road-Side Units, são unidades fixas de infraestrutura espalhados pela cidade. Também capazes de processamento**

Background

- **Premissas**
  - Os carros envolvidos no experimento tem um ID (chassi) que o identifica
  - Existe um server que indica se aquele chassi é de um carro roubado, que não fez a revisão no tempo certo, etc.
  - Cada RSU roda um Queue Server

Cenário de uso



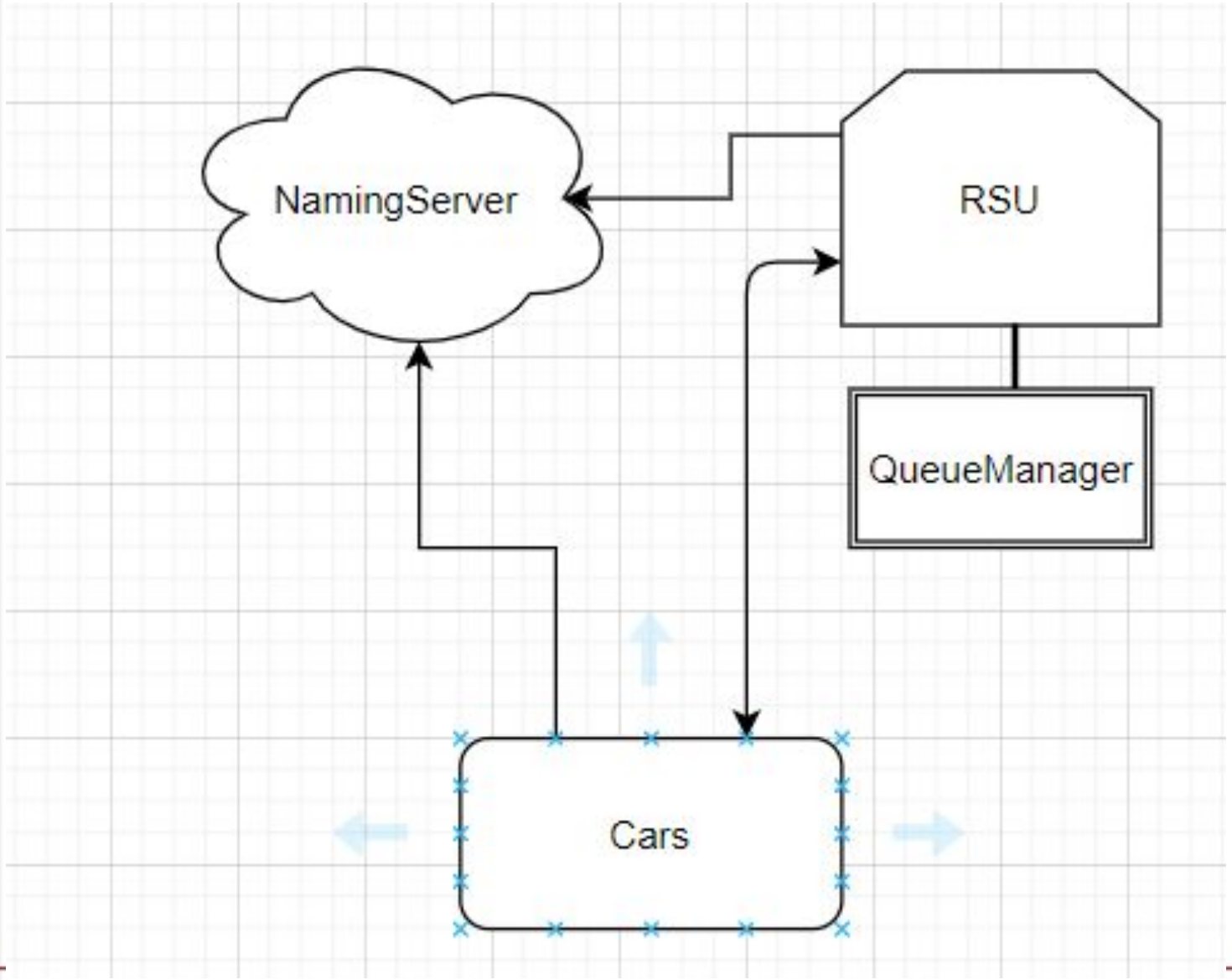
Requisitos

#	Descrição do Requisito Funcional
RF01	**Transparência de acesso:** Às chamadas dos métodos do objeto da fila de eventos são feitas de maneira semelhante a uma chamada local. Acesso Local e Remoto são programados de maneira similar
RF02	**Transparência de localização**: O cliente não precisará saber informações quanto à localização dos servidores.
RF03	**Suporte a múltiplos clientes/Transparência de concorrência**: O RSU deve ser capaz de manter conexão e atender a múltiplos clientes
RF04	**Transparência de replicação:** O recurso da fila de mensagens está replicado em cada RSU
RF05	**Serialização das mensagens**: Os dados serão estruturados em JSON e transformados em streams de bytes antes de serem enviados a rede
RF06	**Criptografia:** Todas as mensagens devem utilizar PKI para criptografar as mensagens

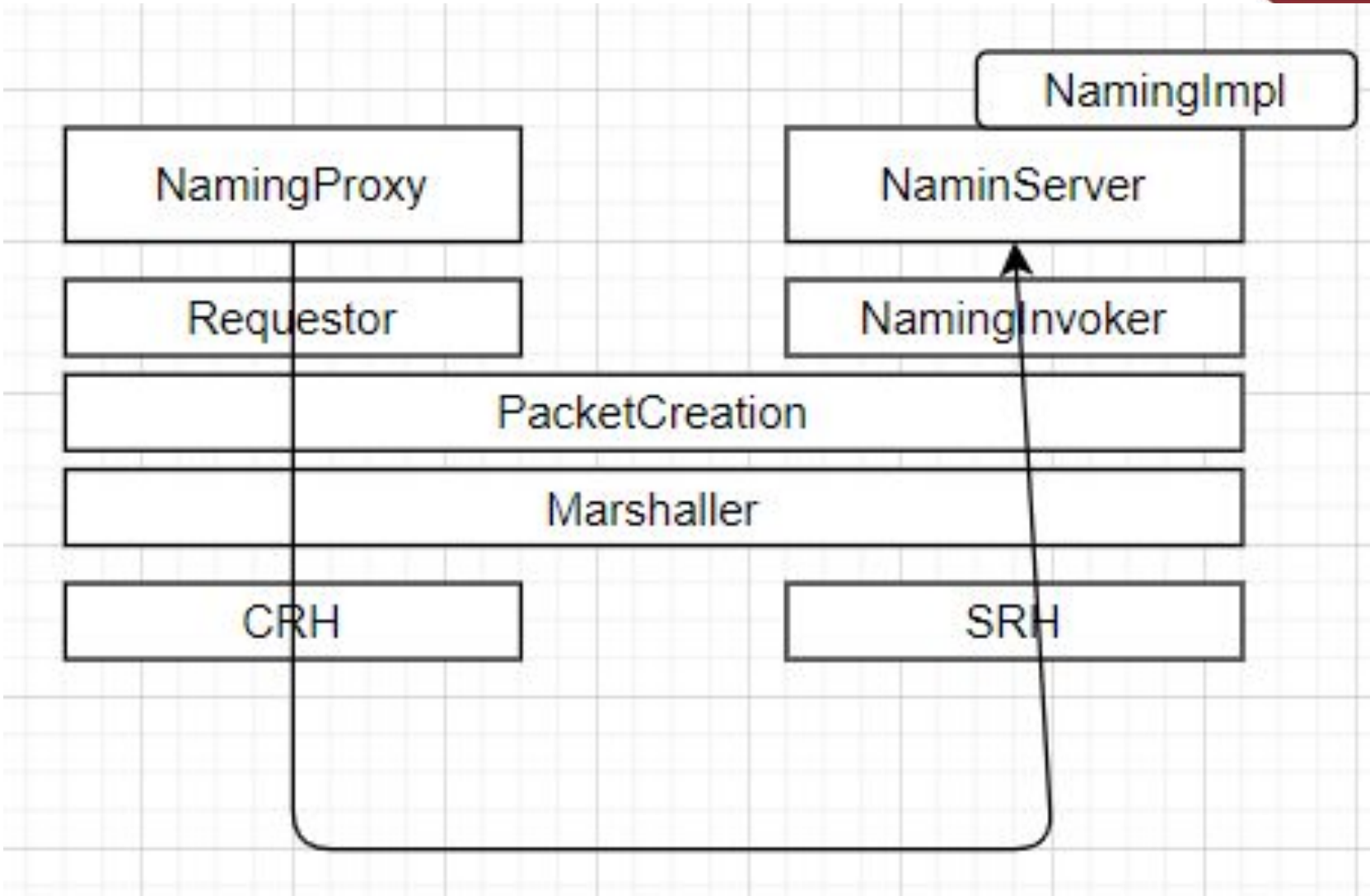
Requisitos

#	Descrição do Requisito Não-Funcional
RNF01	**Uso de Docker**
RNF02	**Padrões de projetos** (Remoting Patterns): Estes padrões serão usados para modularizar o middleware.
RNF03	**Homogeneidade**. Todos os componentes serão desenvolvidos na mesma tecnologia por questões de praticidade. No caso, a tecnologia em questão é Go
RNF04	**Segurança:** Todas as mensagens são criptografadas usando PKI

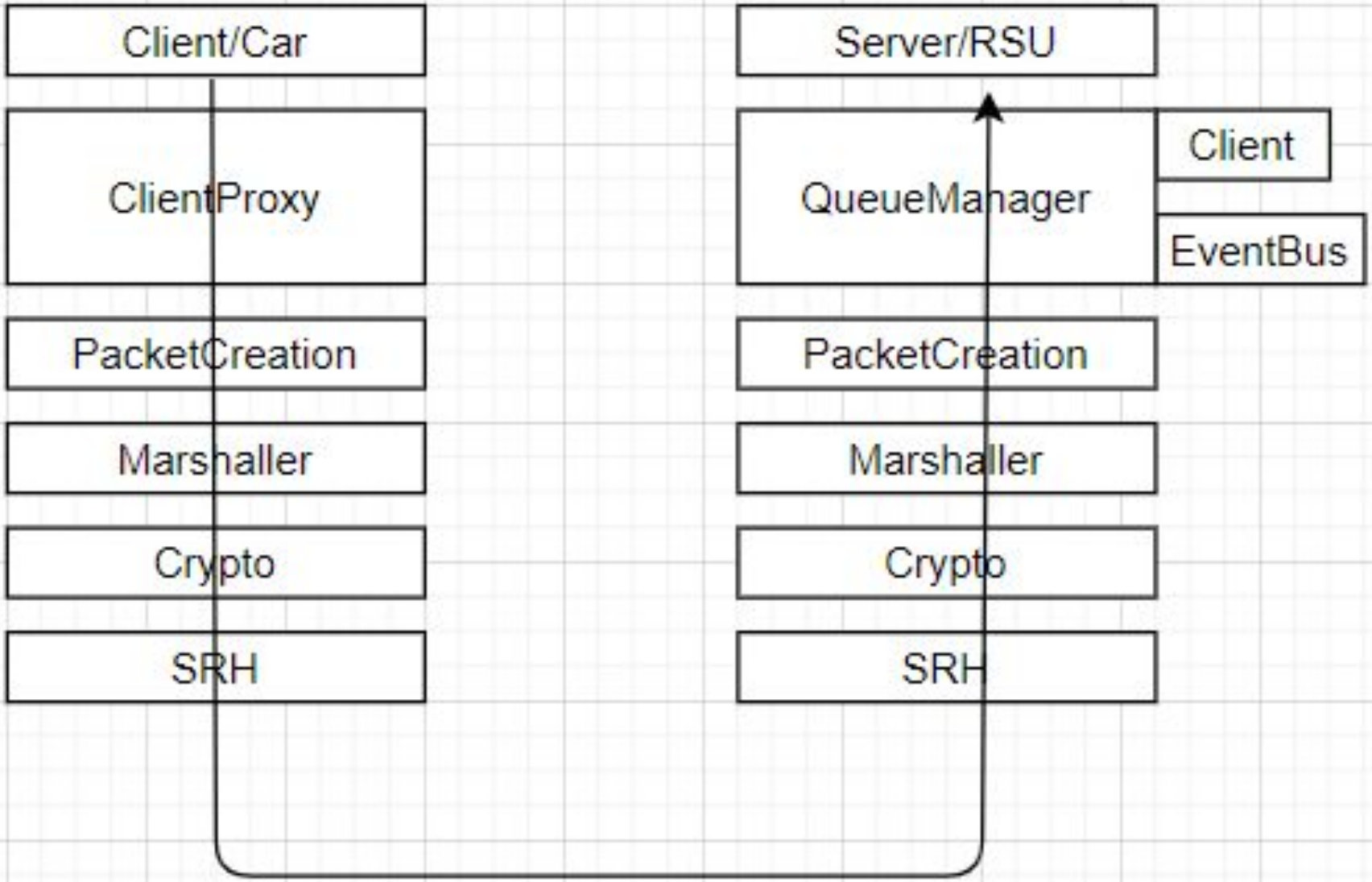
Arquitetura



Arquitetura - NamingServer



Arquitetura



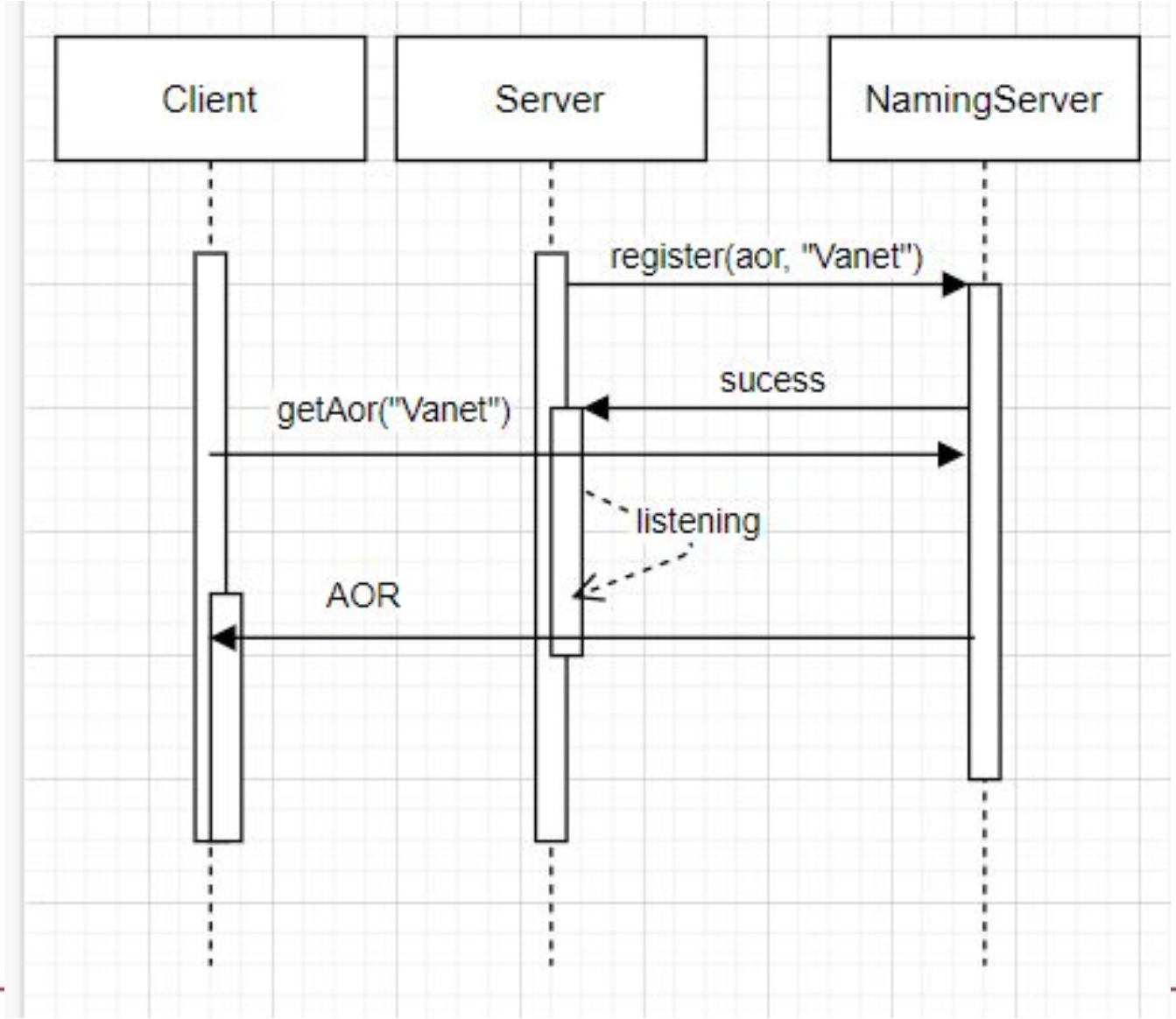
Arquitetura

- **Naming (Transparência de Localização)**
  - Usa invoker e proxies para se comunicar com a lookUpTable.
- **Client Proxy (Transparência de Acesso)**
  - Permite que objetos remotos sejam acessados da mesma maneira que objetos locais
- **Requestor & Invoker**
  - É usado apenas para o NamingServer para acessar o objeto remoto NamingImpl
- **Marshaller (Serialização das Mensagens)**
  - Transforma as mensagens em array de bytes
- **Cripto**
  - Encripta as mensagens
- **QueueManager (Transparencia de concorrencia)**
  - Gerencia a lista de clientes e as filas de eventos que eles disparam.

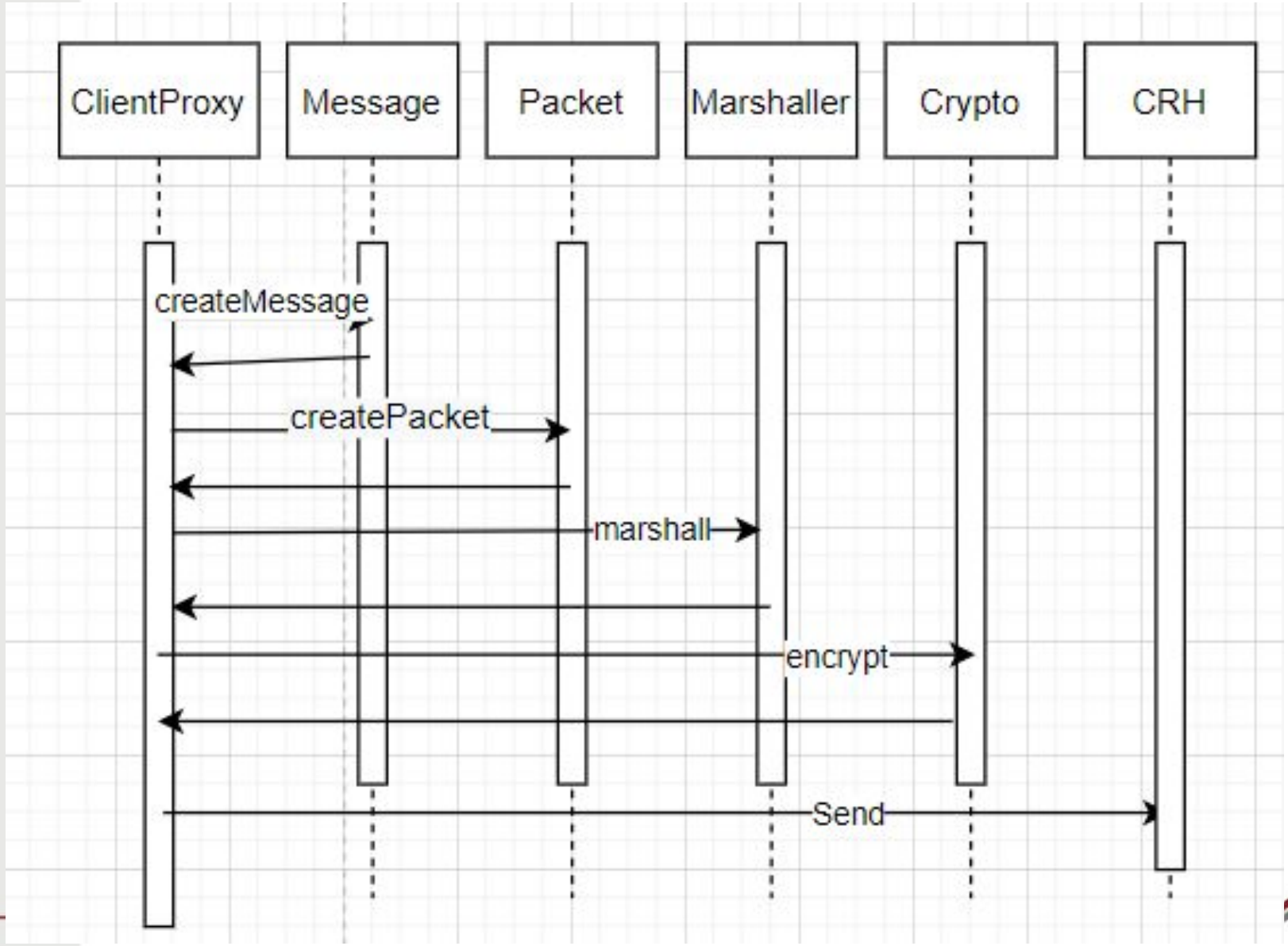
Arquitetura

- **Client & Server Request Handler**
  - Envia e recebe informações

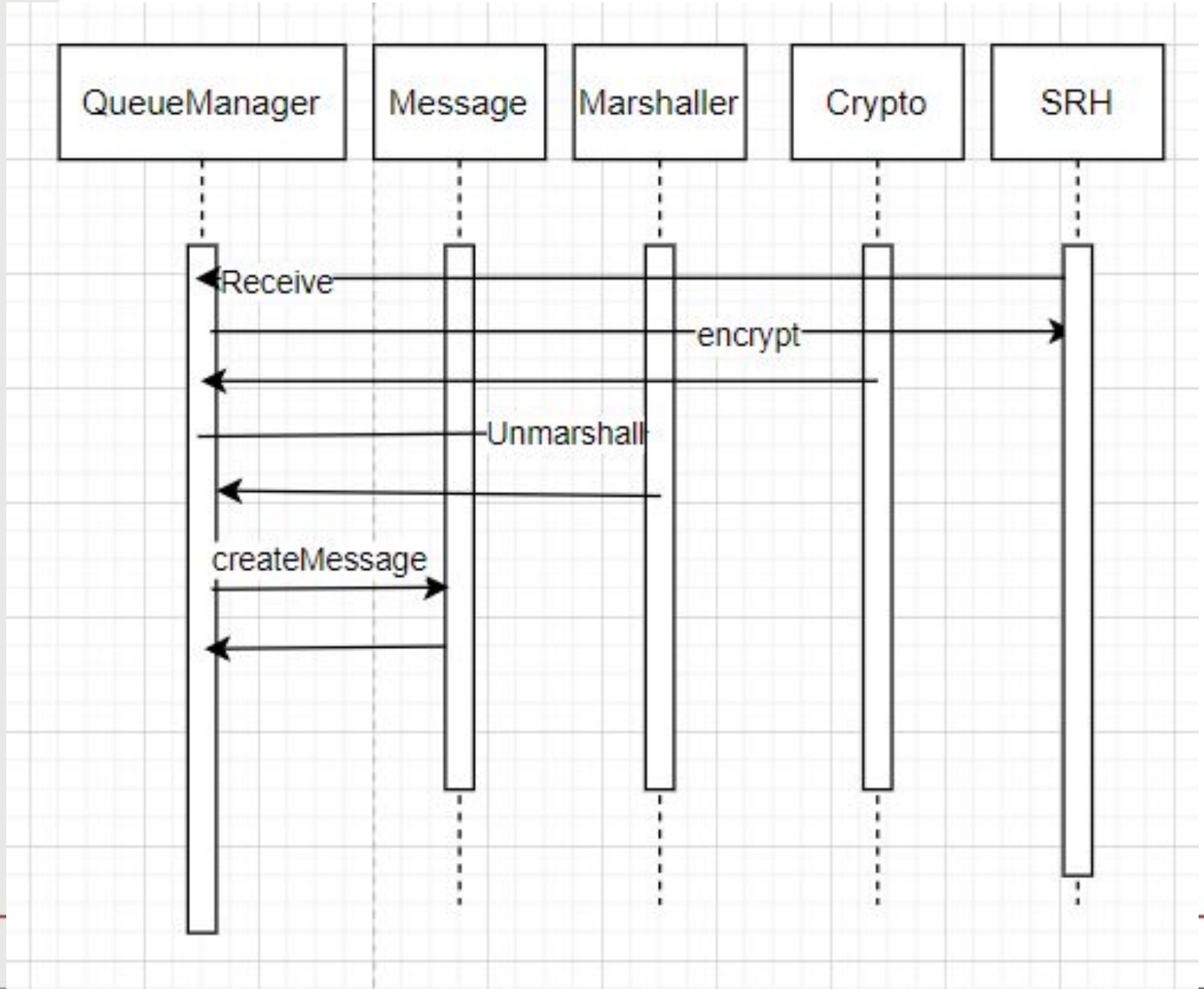
Projeto


Client
Server
NamingServer
register(aor, "Vanet")
sucess
getAor("Vanet")
listening
AOR

Projeto



Projeto


QueueManager
Message
Marshaller
Crypto
SRH
Receive
encrypt
Unmarshall
createMessage

Implementação

```
type AOR struct {
    Address  string `json:"Address"`
    Protocol string `json:"Protocol"`
    ObjectId string `json:"ObjectId"`
    Pub      AORPUB `json:"Pub"`
}
```

Implementação

```
type Message struct {
  Operation string      `json:"operation"`
  Topic     interface{} `json:"topic"`
  AOR       interface{} `json:"AOR"`
  ClientId  int
}
```

Implementação

```
func startClient(id int) *d.ClientProxy {
  namingProxy := n.NewNamingProxy("172.17.0.2:1243")

  aor := namingProxy.LookUp("Vanet")
  fmt.Println("Received aor:")
  fmt.Println(aor)
  var c = d.NewClientProxy(aor, id)
  go c.Start()
  return c
}
```

Implementação

```
func NewClientProxy(aor *common.AOR, id int) *ClientProxy {
    N := big.Int{}
    N.SetString(aor.Pub.N, 10)
    srvPub := &rsa.PublicKey{
        N: &N,
        E: aor.Pub.E,
    }
    return &ClientProxy{
        protocol:   aor.Protocol,
        srvAddress: aor.Address,
        id:         aor.ObjectId,
        Kp:         common.GenerateKeypair(true),
        srvPub:     srvPub,
        ClientId:   id,
    }
}
```

Implementacão

```go
func (cp *ClientProxy) ChangeLane(newLane string) {
  cp.invokeCommand("ChangeLane", newLane)
}

func (cp *ClientProxy) BroadcastEvent(lane string) {
  cp.invokeCommand("BroadcastEvent", lane)
}

func (cp *ClientProxy) RegisterOnLane(lane string) {
  cp.invokeCommand("Register", lane)
}

func (cp *ClientProxy) RegisterKey() {
  pub := &common.AORPUB{
    N: (*(cp.Kp.Pub.N)).String(),
    E: cp.Kp.Pub.E,
  }
  cp.invokeCommand("RegisterKey", pub)
}
```

Implementação

```
func (cp *ClientProxy) Start() {
  for {
    var data []byte
    if cp.crh != nil {
      if cp.protocol == "tcp" {
        data = cp.crh.ReceiveTcp()
      } else {
        data = cp.crh.ReceiveUDP()
      }
      decryptedData := common.Decrypt(data, cp.Kp.Priv, true)
      ter := unpack(decryptedData)
      result := ter.Result
      fmt.Println("Start-result")
      if result != nil {
        fmt.Println(result)
      }
    }
  }
```

Implementação

```go
func GenerateKeypair(client bool) *Keypair {
  var err error
  var size = rsaKeySize
  if client {
    size = rsaClientSize
  }
  fmt.Println("Size: ", size)
  priv, err := rsa.GenerateKey(rand.Reader, size)
  if err != nil {
    return nil
  }
  var pub = &priv.PublicKey
  var kp = &Keypair{
    Priv: priv,
    Pub:  pub,
  }
  return kp
}
```

```go
const (
  rsaKeySize     = 8192
  rsaClientSize = 2048
)
```

Implementação

```
unc Encrypt(message []byte, pub *rsa.PublicKey, client bool) []byte {
 if message == nil {···
 } else if pub == nil {···
 }
 var err error
 fmt.Println(len(message))
 ciphertext, err :=
   rsa.EncryptOAEP(getHash(client), rand.Reader, pub, message, nil)
 if err != nil {···
 }
 // Since encryption is a randomized function, ciphertext will be
 // different each time.
 return ciphertext
```

Implementação

```
unc Decrypt(cipherText []byte, priv *rsa.PrivateKey, client bool) []byte {
 if cipherText == nil {···
 } else if priv == nil {···
 }
 message, err :=
    rsa.DecryptOAEP(getHash(client), rand.Reader, priv, cipherText, nil)
 if err != nil {···
 }
 return message
```

Implementação

```
unc Decrypt(cipherText []byte, priv *rsa.PrivateKey, client bool) []byte {
 if cipherText == nil {···
 } else if priv == nil {···
 }
 message, err :=
    rsa.DecryptOAEP(getHash(client), rand.Reader, priv, cipherText, nil)
 if err != nil {···
 }
 return message
```

Implementação

```
func NewRequestPacket(message *Message) *Packet {
  fmt.Println("Creating package")
  reqHeader := &ReqHeader{ ···
  }
  var reqReqBodyArray = make([]interface{}, 2)
  reqReqBodyArray[0] = message.Topic
  reqReqBodyArray[1] = message.AOR
  reqRepBody := &ReqRepBody{ ···
  }
  packet := &Packet{
    Header: Header{
      Version:  "1.0",
      ClientId: message.ClientId,
    },
    Body: Body{ ···
    },
  }
  return packet
}
```

Centro de Informática

Implementação

```
} else if message.Operation == "RegisterKey" {
  pubMap := message.Topic.(map[string]interface{})
  Nstring := pubMap["N"].(string)
  Eint := int(pubMap["E"].(float64))
  fmt.Println(strconv.Itoa(Eint))
  fmt.Println(Nstring)
  N := big.Int{}
  N.SetString(Nstring, 10)
  clientPub := &rsa.PublicKey{
    N: &N,
    E: Eint,
  }
  c.Pub = clientPub
```

Implementação

```go
func (qm *QueueManager) findPubWithClientId(id int) *rsa.PublicKey {
    qm.mutex.Lock()
    fmt.Println("findPubWithClientId. Locked")
    var pub *rsa.PublicKey
    for _, client := range qm.clients {
        if client.Id == id {
            pub = client.Pub
        }
    }
    qm.mutex.Unlock()
    fmt.Println("findPubWithClientId. Unlocked")
    return pub
}
```

Implementação

- **Carro**
  - Comandos: changeLane, Register, RegisterKey, BroadcastEvent
- **Criptografia**
  - RSA
    - Chave do servidor 8192 bits
    - Chave do cliente 2048 bits
  - Chave pública do servidor salva no NamingServer
  - Chave pública dos clientes enviadas criptografadas pelas chaves do servidor durante fase de registro
- **Docker**

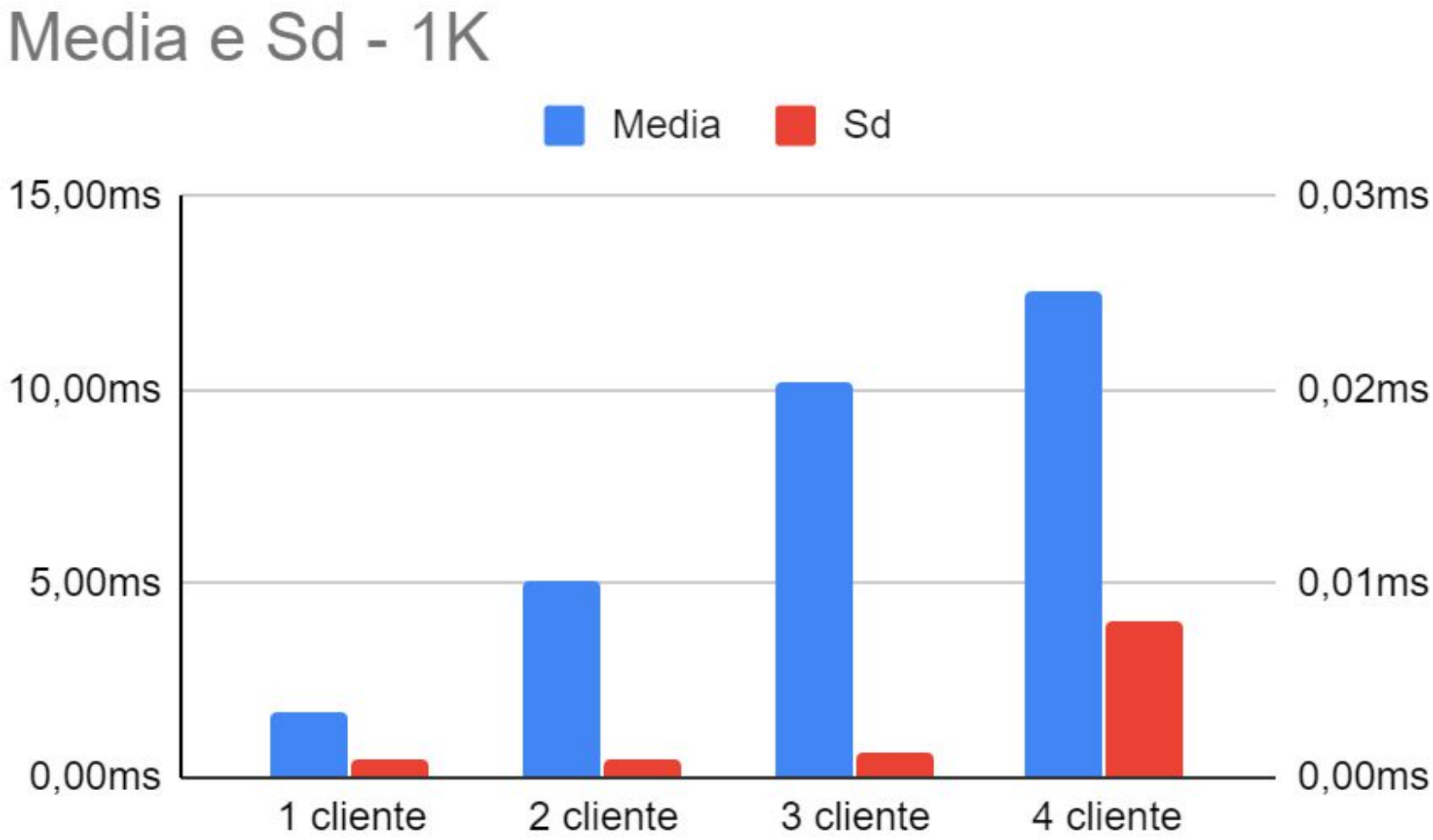
- **Código todo em Go**

Avaliação de Desempenho

- **Setup**
  - Programas executando:
    - Apenas o terminal com o docker executando o middleware
  - Rede Wi-fi Conectada
  - Carregador Conectado

- **Avaliação**
  - A aplicação realiza operações de Registro de chave, e de área. Em seguida efetua operação de mudança de faixa. Foram executados dois batches um com mil mensagens outro com 10 mil mensagens. Ambos os batches com 1, 2, 3 e 4 clientes simultaneos

Avaliação de Desempenho



Avaliação de Desempenho

Medida	1 cliente	2 cliente	3 cliente	4 cliente
Média	1,64ms	5,07ms	10,21ms	12,56ms
Sd	0,001ms	0,00093ms	0,0012ms	0,008ms

Avaliação de Desempenho

Avaliação de Desempenho

Medida	1 cliente	2 cliente	3 cliente	4 cliente
Media	4,23ms	8,86ms	12,34ms	16,95ms
Sd	0,00138ms	0,00096ms	0,0013ms	0,003ms

Conclusão

- [coloque aqui as características mais importantes do seu middleware]
- [coloque aqui as limitações do seu middleware]
- [lições aprendidas com o projeto]

Observações

- **As equipes precisarão apresentar e discutir a proposta de projeto antes do início da implementação**
- **Projeto precisa ser implementado em Java, C, C++, C#, Go, Erlang, Elixir**
  - Outras linguagens poderão ser aceitas se discutidas com o professor
- **Equipes com até 03 (três) integrantes**
- **Data de Entrega**
  - Como definido no site da disciplina
- **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