



Ex02 Middleware



Setup - Server

- Processador i7
- 8GB de RAM
- Windows 10 64 bits
- SSD
- Wi-fi: ligada
- Apenas a IDE (Goland) aberta



Setup - Cliente(s)

- MacBook Pro
- Processador i7
- 16GB de RAM
- SSD
- Wi-fi: ligada
- Apenas a IDE (VsCode) aberta



Aplicação

- Chat
- Servidor aceita múltiplos usuários
- Uma mensagem enviada por um usuário deve ser distribuída para todos os outros usuários logados no servidor



Experimento - Definição

- Não abro e fecho a conexão para cada request
- Envio de uma mensagem



Talk is cheap. Show me the code.


— *Linus Torvalds* —

AZ QUOTES




 main.go


 chat_message.proto ×

 udpclient.go

 tcpclient.go


protocol > rpc >  chat_message.proto

```
1  syntax = "proto3";
2  package protocol_rpc;
3
4  service rpcChat {
5      rpc sendMessages(stream messages) returns (stream messages) {}
6  }
7
8  message messages {
9      string name = 1;
10     string message = 2;
11 }
```




```
func (c *RpcClient) Dial(address string) error {
    var opts []grpc.DialOption
    opts = append(opts, grpc.WithInsecure())
    opts = append(opts, grpc.WithBlock())
    println("Dialing ", address)
    conn, err := grpc.Dial(address, opts...)
    if err == nil {
        println("Dialed successfully")
        c.conn = conn
        c.client = pb.NewRpcChatClient(conn)
    } else {
        log.Printf("dial error=%s", err.Error())
    }

    return err
}
```

```
func (c *RpcClient) SendMessage(message string) error {  
    if c.stream != nil {  
        serializedMessage := c.serialize(c.name, message)  
        err := c.stream.Send(&serializedMessage)  
        return err  
    }  
    return errors.New("no stream")  
}
```


```
func (c *RpcClient) serialize(name string, message string) pb.Messages {  
    return pb.Messages{  
        Name:      name,  
        Message:   message,  
    }  
}
```



```
func (c *RpcClient) Start() {
    println("Sending stuff")
    ctx, cancel := context.WithTimeout(context.Background(), 30000*time.Second)
    c.stream, _ = c.client.SendMessages(ctx)
    for {
        in, err := c.stream.Recv()
        if err == io.EOF {
            break
        } else if err != nil {
            log.Printf("Read error %v", err)
            break
        }
        if in != nil {
            c.incoming <- c.messageToCommand(in)
        }
    }
    defer cancel()
}
```


```
func (s *RpcServer) SendMessages(stream pb.RpcChat_SendMessagesServer) error {
    for {
        in, err := stream.Recv()
        if err == io.EOF {
            return nil
        }
        if err != nil {
            return err
        }

        client := client{
            stream: stream,
        }
        s.mutex.Lock()
        s.clients[in.Name] = &client
        println("server received:", in.Name, in.Message)
        cmd := protocol.MessageCommand{
            Name:    in.Name,
            Message: in.Message,
        }
        s.Broadcast(cmd)
        s.mutex.Unlock()
    }
}
```



```
func (s *RpcServer) Start() {
    grpcServer := grpc.NewServer()
    pb.RegisterRpcChatServer(grpcServer, s)
    grpcServer.Serve(s.listener)
}

func (s *RpcServer) Listen(address string) error {
    flag.Parse()
    lis, err := net.Listen("tcp", address)
    s.listener = lis
    if err != nil {
        log.Fatalf("failed to listen: %v", err)
    }
    return err
}
```



```
func (s *RpcServer) Broadcast(command interface{}) error {
    var in pb.Messages
    switch v := command.(type) {
    case protocol.MessageCommand:
        in = pb.Messages{
            Name:          v.Name,
            Message:        v.Message,
        }
    }
    for _, client := range s.clients {
        err := client.stream.Send(&in)
        if (err != nil) {
            return err
        }
    }
    return nil
}
```

Gráfico de barras mostrando a média em nanosegundos do tcp, udp e rpc variando o número de clientes simultaneos no server

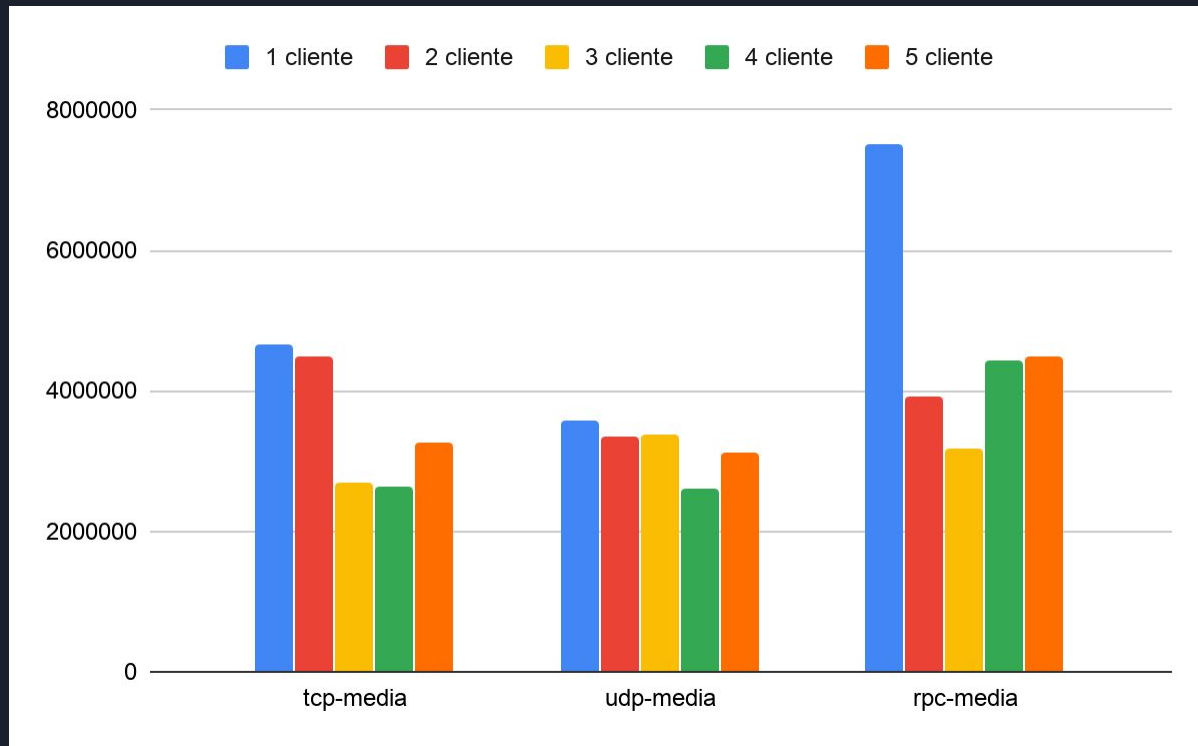




Tabela com média e desvio padrão

	10K	20K	17K	24K	24K
tech	1 cliente	2 cliente	3 cliente	4 cliente	5 cliente
tcp-media	4666579,874	4482366,399	2689208,496	2625793,523	3270637,43
udp-media	3576550,45	3336877,423	3390203,885	2621711,998	3112051,849
rpc-media	7525160,534	3931476,376	3191801,979	4439429,763	4490949,309
udp-sd	1627584,07	1316768,204	1387847,743	1405537,8	1437354
tcp-sd	14666275,66	3092133,34	2345034,041	1539143,79	2443028,513
rpc-sd	44915005,85	2665456,707	2790359,136	3311240,624	3334939,268