Atividade 4

RPC

Grupo:
Ekistoclecio Heleno Duarte de Lima
Ian Karlo Torres dos Santos
Samuel Simões de Souza Filho

Centro de Informática UFPE

Universidade Federal de Pernambuco

O projeto

O objetivo é medir o desempenho da conexão cliente e servidor utilizando o middleware RPC (Remote Procedure Call) em comparação com uma conexão estabelecida usando puramente os sockets UDP e TCP. Para isso foram utilizadas implementações semelhantes de cliente e servidor e medidos os parâmetros de desempenho associados ao RTT.

Backtracking

Implementação

```go
type SudokuSolver struct{}

func (s *SudokuSolver) Run(req common.Request, rep *common.Reply) error {

    board := req.Board

    numRoutines := 5

    matrizChannel := make(chan [][]int)
    signalChannel := make(chan int, numRoutines)

    // fmt.Println("-----------", request)
    rand.Seed(42)
    for i := 0; i < numRoutines; i++ {
        go Solve(&matrizChannel, &signalChannel, board, i)
    }

    matrix := <-matrizChannel

    // PrintBoard(matrix, 9)

    rep.R = matrix

    return nil
}
```

Servidor TCP

```
func BuildTCPServer() {
    sudokuSolver := new(impl.SudokuSolver)

    server := rpc.NewServer()

    server.RegisterName("SudokuSolver", sudokuSolver)

    listener, err := net.Listen("tcp", ":5555")
    if err != nil {…
    }

    server.Accept(listener)
}
```

Cliente TCP

```go
func TcpRpcClient(board []int, iterations int, testing bool, boardNumber int) {
    var reply common.Reply

    client, err := rpc.Dial("tcp", ":5555")
    if err != nil {…
    }

    defer client.Close()

    times := make([]float64, 0)
```

```go
for i := 0; i < iterations; i++ {

    startTime := time.Now()

    args := common.Request{
        Board: board,
    }

    err := client.Call("SudokuSolver.Run", args, &reply)

    if err != nil {…
    }

    duration := time.Since(startTime)

    times = append(times, float64(duration.Microseconds()))

    if testing {…
    }
}
```

Servidor HTTP

```go
func BuildHTTPServer() {
    sudokuSolver := new(impl.SudokuSolver)

    server := rpc.NewServer()

    server.RegisterName("SudokuSolver", sudokuSolver)

    server.HandleHTTP("/", "/debug")

    listener, err := net.Listen("tcp", ":5555")

    if err != nil {
        panic(err)
    }

    http.Serve(listener, nil)
}
```

Cliente HTTP

```go
func HttpRpcClient(board []int, iterations int, testing bool, boardNumber int) {
    var reply common.Reply

    client, err := rpc.DialHTTP("tcp", ":5555")
    if err != nil {
        panic(err)
    }

    defer client.Close()

    times := make([]float64, 0)
```

```go
for i := 0; i < iterations; i++ {

    startTime := time.Now()

    args := common.Request{
        Board: board,
    }

    err := client.Call("SudokuSolver.Run", args, &reply)

    if err != nil {
        panic(err)
    }

    duration := time.Since(startTime)

    times = append(times, float64(duration.Microseconds()))

    if testing {
        common.PrintBoard(reply.R, 9)
    }
}
```

Análise de Desempenho - Especificações do Sistema

- Processador: Intel(R) Core (TM) i7-1165G7 @ 2.80GHz, 4 núcleos, 8 processadores lógicos
- Memória: 16GB
- Sistema operacional: Linux Mint 21.1
- Linguagem de programação: Go (Versão 1.21.1)
- Fonte de alimentação: Rede elétrica
- Processos em execução: Processos essenciais do sistema.

Análise de Desempenho - Métrica

A métrica utilizada é o tempo entre o envio de uma chamada pelo cliente ao servidor e o recebimento da resposta para essa chamada. **Após estabeler uma conexão com o servidor utilizando o GoRPC**, onde o cliente faz chamadas em sequência, enviando um tabuleiro de sudoku e esperando que o servidor o responda com o tabuleiro resolvido.

Técnica de avaliação

Foram utilizados dois pares de aplicações, cliente e servidor, **se comunicando utilizando o GoRPC com dois protocolos diferentes, TCP e HTTP respectivamente**. Em seguida realizamos dez mil requisições sequenciais cada uma contendo um tabuleiro. Uma nova requisição só era enviada após receber a resposta do servidor para a requisição anterior. Medindo-se o tempo decorrido entre o envio de uma requisição e o recebimento da resposta. Após todas as requisições, é calculado o tempo médio total de todas elas.

Análise de Desempenho - Carga

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

					5		6	
		5		6				
	7					4		
	8		1				2	
	3			7				
			5		2	1		3
		6			1		4	
		2					7	6
		3			4	5		1

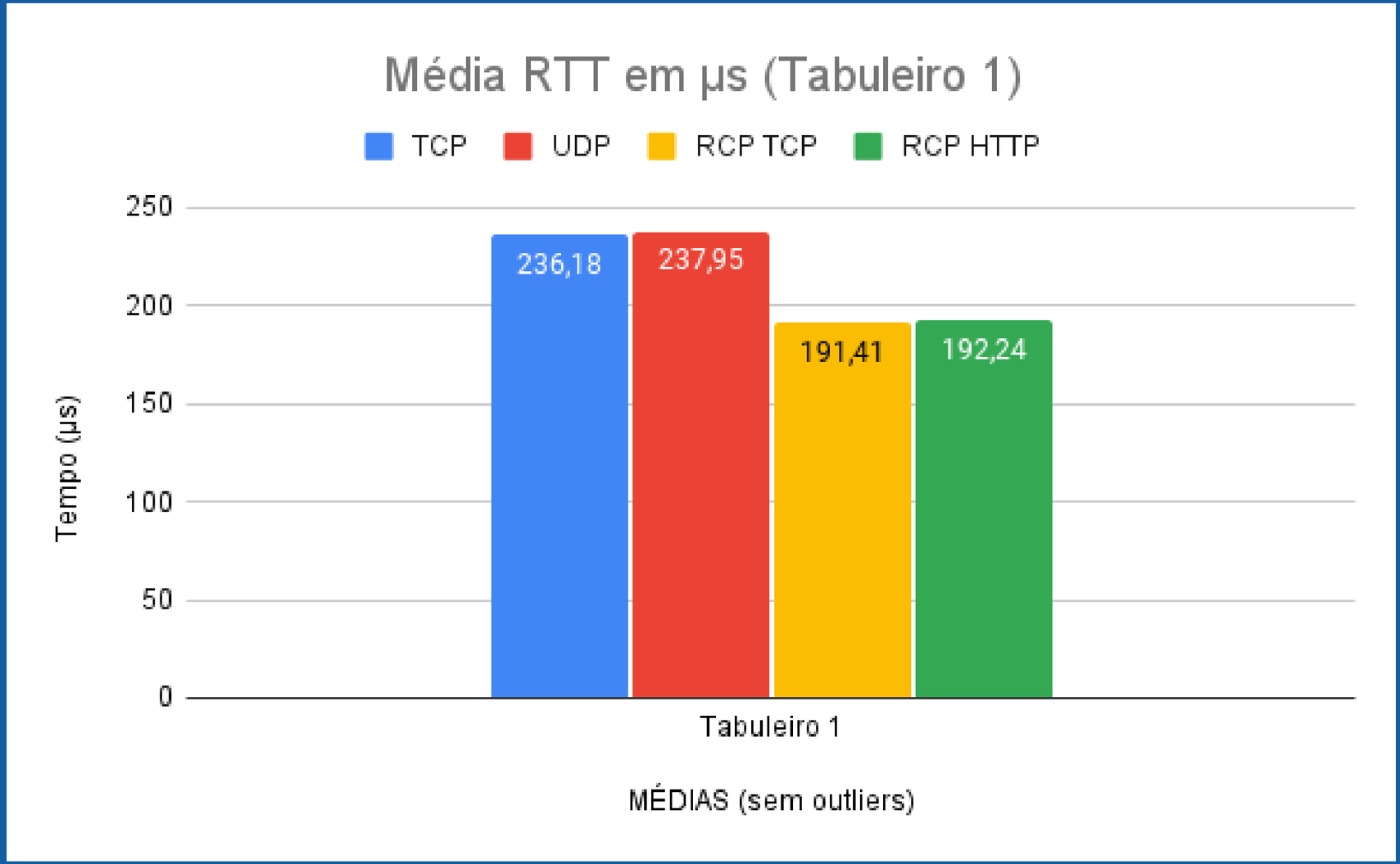
	9		2					
						5		
4	8			5		6	9	2
					9			
			4	2			8	
			8		7			5
	6	1	7		5	9		4
	4			6			5	1
2			1					6

Para evitar que configurações diferentes de tabuleiro tivessem impacto nos testes foram utilizados 3 modelos diferentes para o processo de análise.

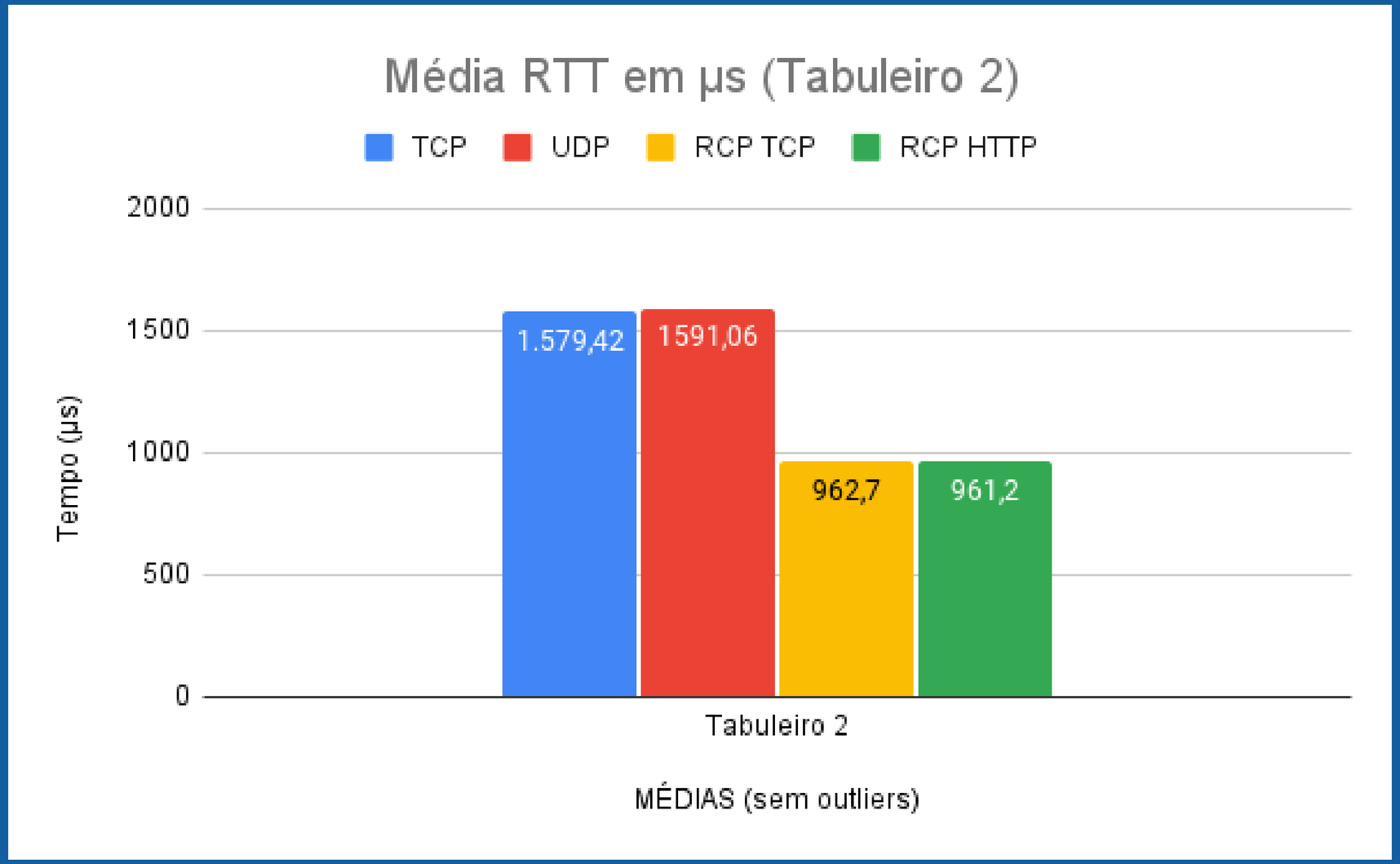
Análise

Análise estatística dos dados obtidos

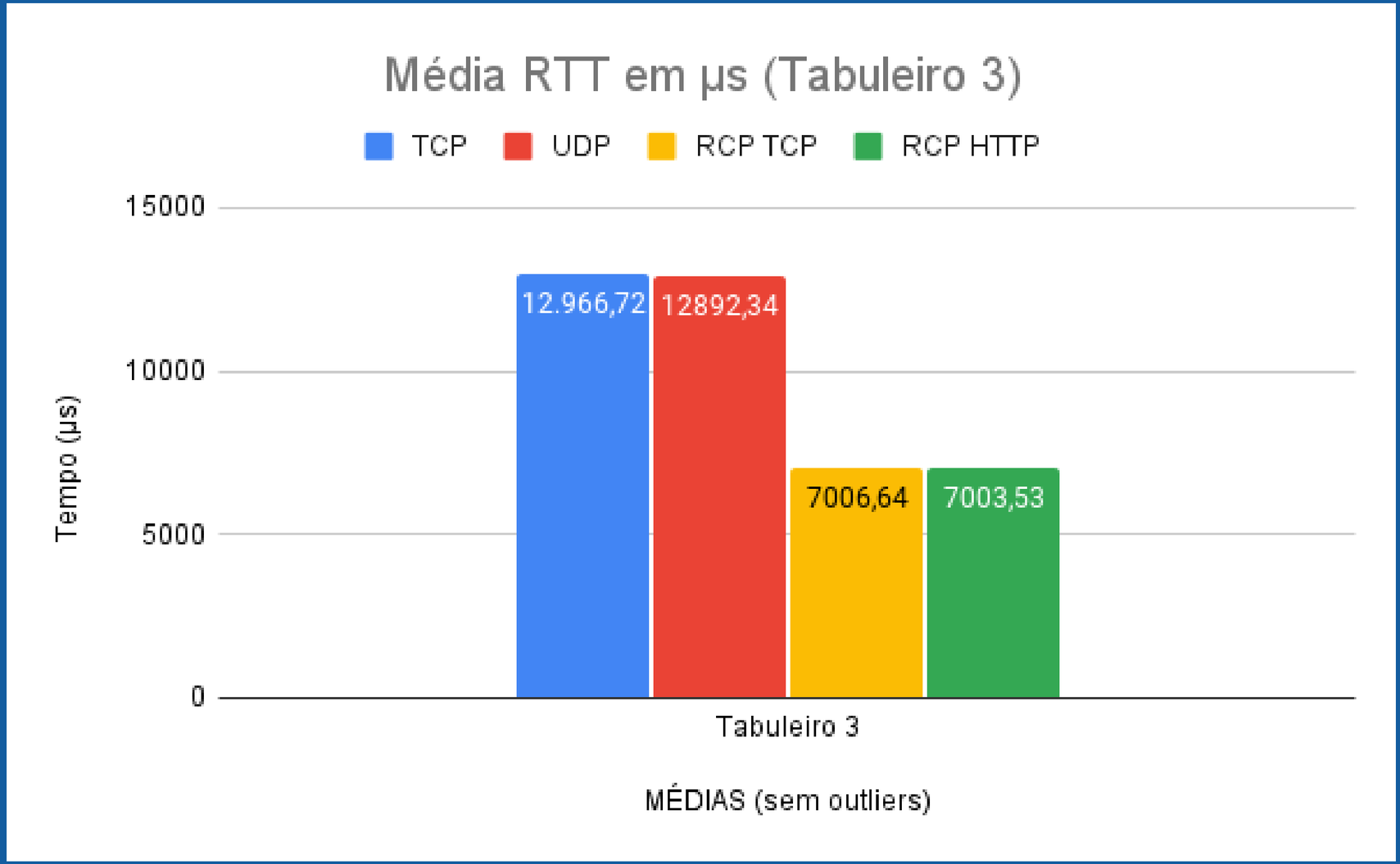
Análise de Desempenho (Tabuleiro 1)



Análise de Desempenho (Tabuleiro 2)

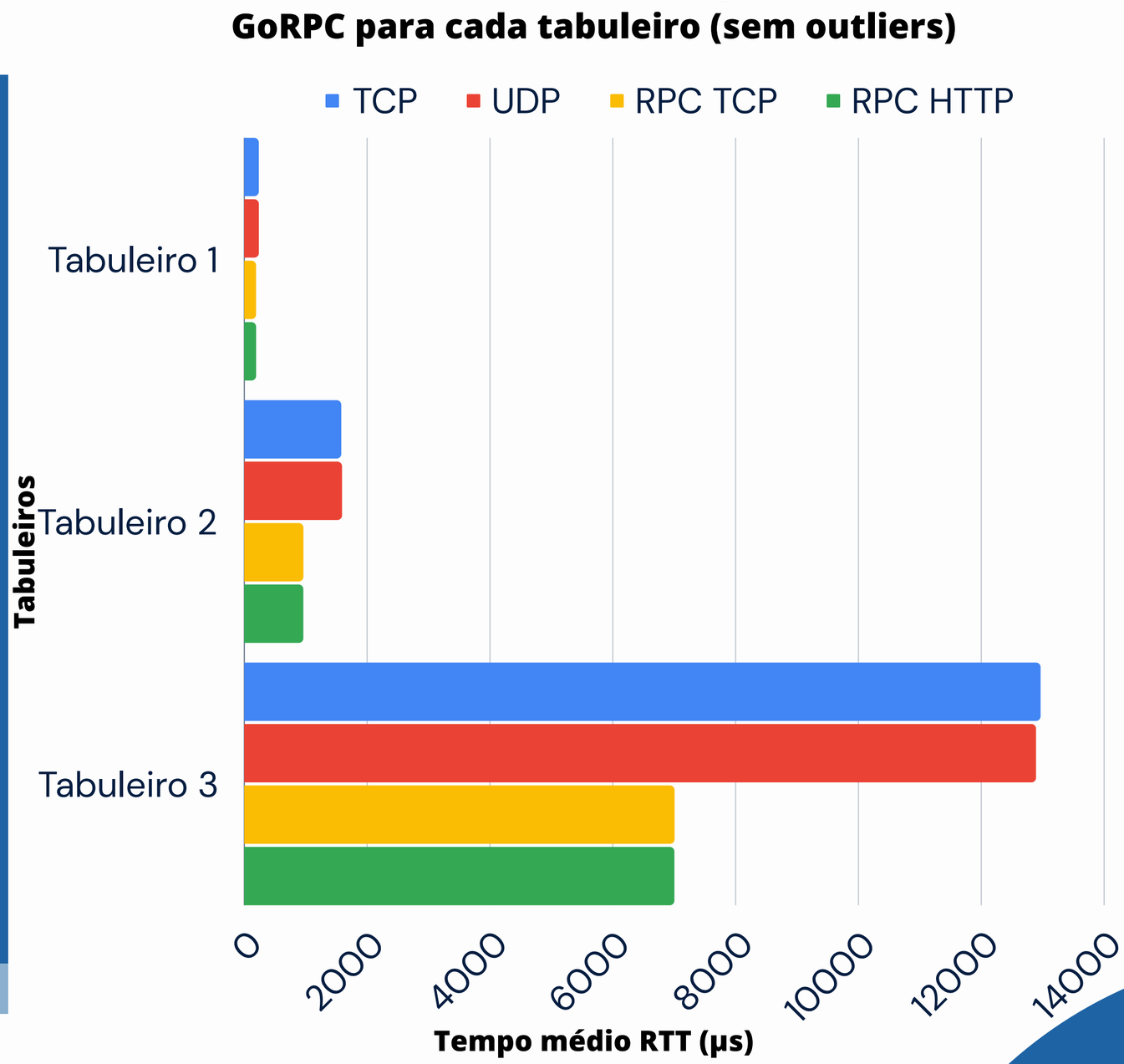


Análise de Desempenho (Tabuleiro 3)



Resultados

Partindo das análises realizadas, é possível concluir que, embora o GoRPC realize alguns processamentos adicionais, sua implementação conta com mecanismos de otimização para as requisições, refletidos no melhor desempenho em comparação com implementações que utilizam puramente os sockets UDP e TCP. Comparando os protocolos HTTP e TCP, implementados no GoRPC, apresentam um tempo consideravelmente próximo, resultado também das otimizações que tornam a diferença entre os protocolos insignificante para o tempo médio das requisições.



Obrigado