Atividade 3
TCP e UDP

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

- A aplicação do servidor consiste em um solucionador de sudoku utilizando a técnica backtracking e bitmap utilizado nas atividades anteriores.
- Nesse projeto, o cliente envia uma matriz (em forma de vetor) para o servidor e recebe de volta a matriz solucionada.

Backtracking

Implementação - Base do Servidor

```
fmt.Println("Starting Server")

protocolType := os.Args[1]

numRequests, err := strconv.Atoi(os.Args[2])
if err != nil {
    numRequests = 1
    fmt.Println("Invalid value for numRequests")
}

if protocolType == udp {
    fmt.Println("Running UDP server")
    StartUDPServer(numRequests)
} else if protocolType == tcp {
    fmt.Println("Running TCP server")
    StartTCPServer(numRequests)
} else {
    fmt.Println("Please select a valid protocol")
}
```

Implementação - Base do Cliente

```
boards := [][]int{board0, board1, board2}

protocolType := os.Args[1]
var times []int64

numRequests, err := strconv.Atoi(os.Args[2])
if err != nil {
    numRequests = 1
    fmt.Println("Invalid value for numRequests")
}
boardIndex, err := strconv.Atoi(os.Args[3])
if err != nil {
    boardIndex = 1
    fmt.Println("Invalid value for boardIndex")
}
```

```
if protocolType == udp {
    fmt.Println("Running UDP client")
    times = RunUDPClient(numRequests, boards[boardIndex])
} else if protocolType == tcp {
    fmt.Println("Running TCP client")
    times = RunTCPClient(numRequests, boards[boardIndex])
} else {
    fmt.Println("Please select a valid protocol")
    return
}

mean := utils.GetMean(times)
fmt.Println(times)

fmt.Println(mean)
```

Implementação - Servidor TCP

```go
addr, err := net.ResolveTCPAddr("tcp", "localhost:9091")

if err != nil {…
}

listener, err := net.ListenTCP("tcp", addr)

if err != nil {…
}

conn, err := listener.Accept()

if err != nil {…
}
```

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

    req, err := bufio.NewReader(conn).ReadString('\n')

    if err != nil {…
    }

    var matrix [][]int
    err = json.Unmarshal([]byte(req), &matrix)

    res := sudoku.Run(i, matrix)

    jsonData, err := json.Marshal(res)

    if err != nil {…
    }

    jsonData = append(jsonData, '\n')

    _, err = conn.Write([]byte(jsonData))

    if err != nil {…
    }
}
```

Implementação - Cliente TCP

```go
func RunTCPClient(numRequests int, board []int) []int64 {
    times := make([]int64, 0)

    r, err := net.ResolveTCPAddr("tcp", "localhost:9091")

    if err != nil {…
    }

    conn, err := net.DialTCP("tcp", nil, r)

    if err != nil {…
    }

    requests := 0
```

```go
for requests < numRequests {

    start := time.Now()

    jsonData, err := json.Marshal(board)

    if err != nil {…
    }

    jsonData = append(jsonData, '\n')

    _, err = fmt.Fprintf(conn, string(jsonData))

    if err != nil {…
    }

    res, err := bufio.NewReader(conn).ReadString('\n')

    if err != nil {…
    }

    var matrix [][]int
    err = json.Unmarshal([]byte(res), &matrix)
    if err != nil {…
    }

    // utils.PrintBoard(matrix, 9)

    end := time.Now()

    times = append(times, end.Sub(start).Microseconds())

    requests++
```

Implementação - Servidor UDP

```go
func StartUDPServer(maxIter int) {

    req := make([]byte, 1024)

    addr, err := net.ResolveUDPAddr("udp", "localhost:9091")

    if err != nil {…
    }

    conn, err := net.ListenUDP("udp", addr)

    if err != nil {…
    }
```

```go
for i := 0; i < maxIter; i++ {
    _, addr, err := conn.ReadFromUDP(req)

    if err != nil {…
    }

    limitIndex := utils.GetEndOfBuffer(req)

    var matrix []int
    err = json.Unmarshal([]byte(req[:limitIndex]), &matrix)

    res := sudoku.Run(i, matrix)

    jsonData, err := json.Marshal(res)

    if err != nil {…
    }

    jsonData = append(jsonData, '\n')

    _, err = conn.WriteTo([]byte(jsonData), addr)

    if err != nil {…
    }
}
```

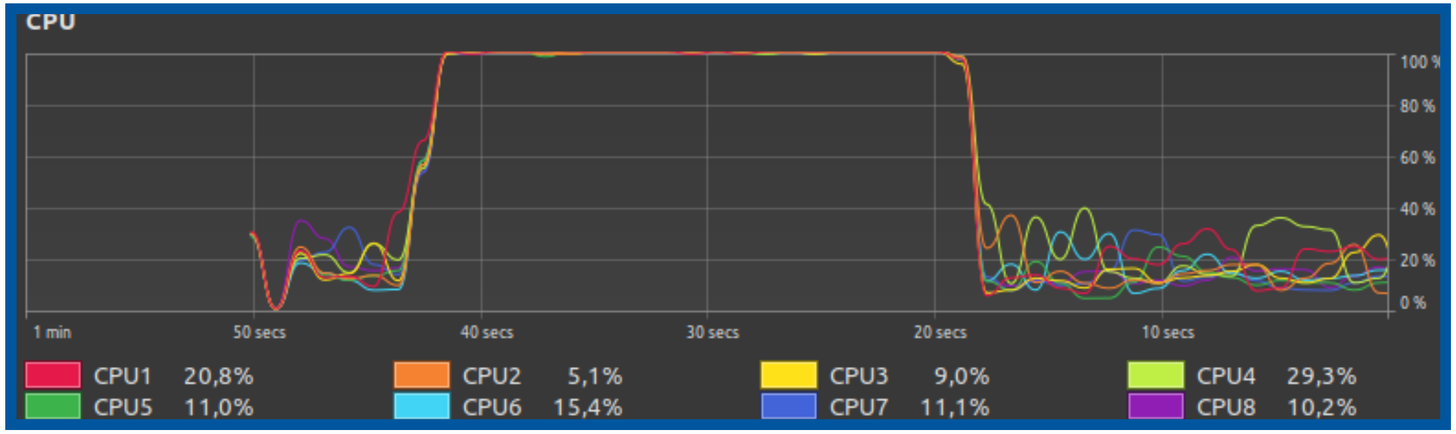
Implementação - Cliente UDP

```go
func RunUDPClient(numRequests int, board []int) []int64 {
    resBuffer := make([]byte, 1024)
    times := make([]int64, 0)

    addr, err := net.ResolveUDPAddr("udp", "localhost:9091")

    if err != nil {…
    }

    conn, err := net.DialUDP("udp", nil, addr)

    if err != nil {…
    }

    requests := 0
```

```go
for requests < numRequests {

    start := time.Now()

    jsonData, err := json.Marshal(board)

    if err != nil {…
    }

    jsonData = append(jsonData, '\n')

    _, err = conn.Write(jsonData)

    if err != nil {…
    }

    _, _, err = conn.ReadFromUDP(resBuffer)

    if err != nil {…
    }

    limitIndex := utils.GetEndOfBuffer(resBuffer)

    var matrix [][]int
    err = json.Unmarshal([]byte(resBuffer[:limitIndex]), &matrix)
    if err != nil {
        fmt.Println(string(resBuffer))
        fmt.Println("Error:", err)
        panic(err)
    }

    // utils.PrintBoard(matrix, 9)

    end := time.Now()

    times = append(times, end.Sub(start).Microseconds())

    requests++
}
```

Problemas de escalabilidade

Com 10000 requisições, cada requisição disparando 5 threads, o computador começou a travar muito, chegando a 100% de uso da CPU.

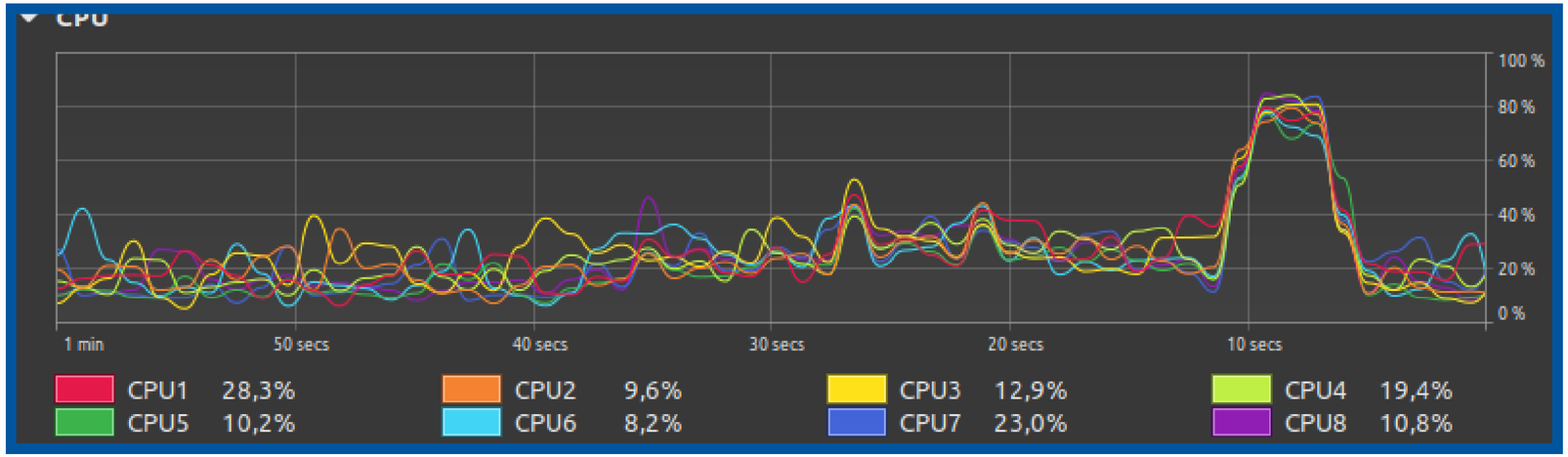


Uso de CPU em 500 requisições

```
---------- 0
Sucesso 4
---------- 1
Sucesso 0
Sucesso 2
Sucesso 1
---------- 2
Sucesso 2
Sucesso 1
Sucesso 4
Sucesso 4
```

Problemas de escalabilidade

Mudando a estretégia, encerrando as outras threads quando encontrar uma solução.



Uso de CPU em 500 requisições

```
---------- 0
Sucesso 3
Encerrou sem sucesso 2
Encerrou sem sucesso 4
Encerrou sem sucesso 0
Encerrou sem sucesso 1
---------- 1
Sucesso 0
Encerrou sem sucesso 4
Encerrou sem sucesso 3
Encerrou sem sucesso 1
Encerrou sem sucesso 2
---------- 2
Sucesso 0
Encerrou sem sucesso 4
Encerrou sem sucesso 3
Encerrou sem sucesso 2
Encerrou sem sucesso 1
```

Solução

Utilizamos um canal para indicar para as outras threads (go routines) que uma já alcançou objetivo e não é mais necessário continuar a operação.

```
func SolveRecursive(row, col int, board [][]int, size int, lines, columns, sectors []int, signalChannel
    // tenta ler do canal bufferizado, se ler algo ele se encerra, se não ele continua
    if len(*signalChannel) != 0 {
        return false
    }
```

```
    if SolveRecursive(i, j, board, size, lines, columns, sectors, signalChannel) {
        // fmt.Println("Sucesso :)", id)
        *signalChannel <- 1
        *channel <- board
        //manda informações de termino pro canal bufferizado
```

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. Apos a conexão com o servidor o cliente faz chamadas em sequencia enviando um tabuleiro 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 via protocolo TCP e UDP respectivamente, foram realizadas dez mil requisições sequenciais cada uma contendo um tabuleiro e uma nova requisição so era enviada apos 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 e em seguida calculado o tempo medio decorrido entre cada requisição e resposta.

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

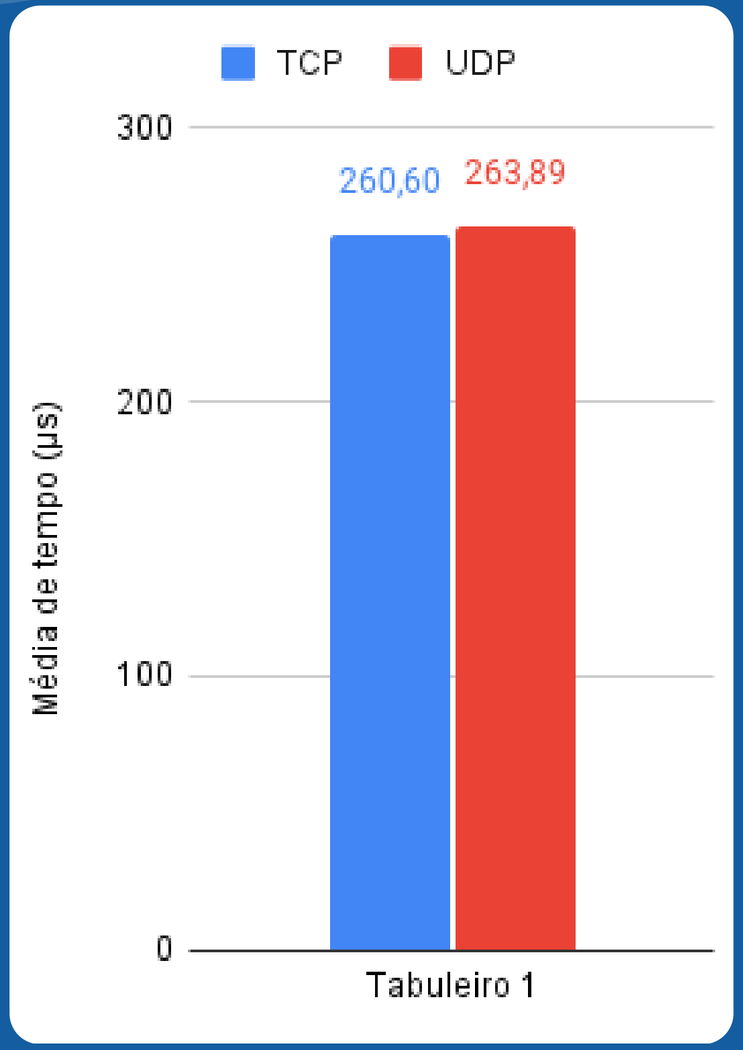
	9		2					
						5		
4	8			5		6	9	2
					9			
			4	2			8	
			8		7			5
	6	1	7		5	9		4
	4			6			5	1
2			1					6

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

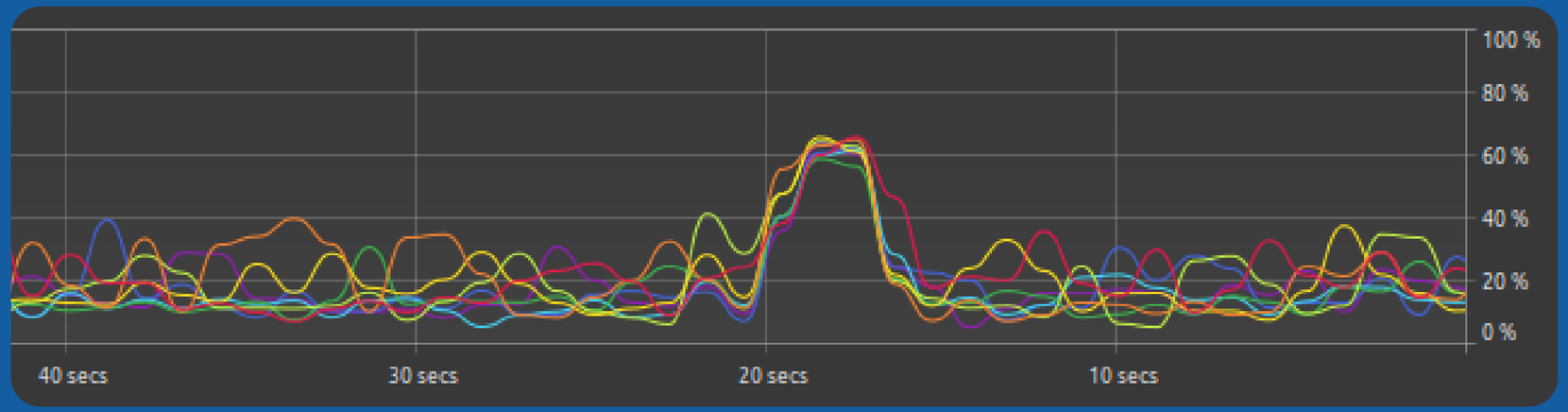
Análise

Análise estatística dos dados obtidos

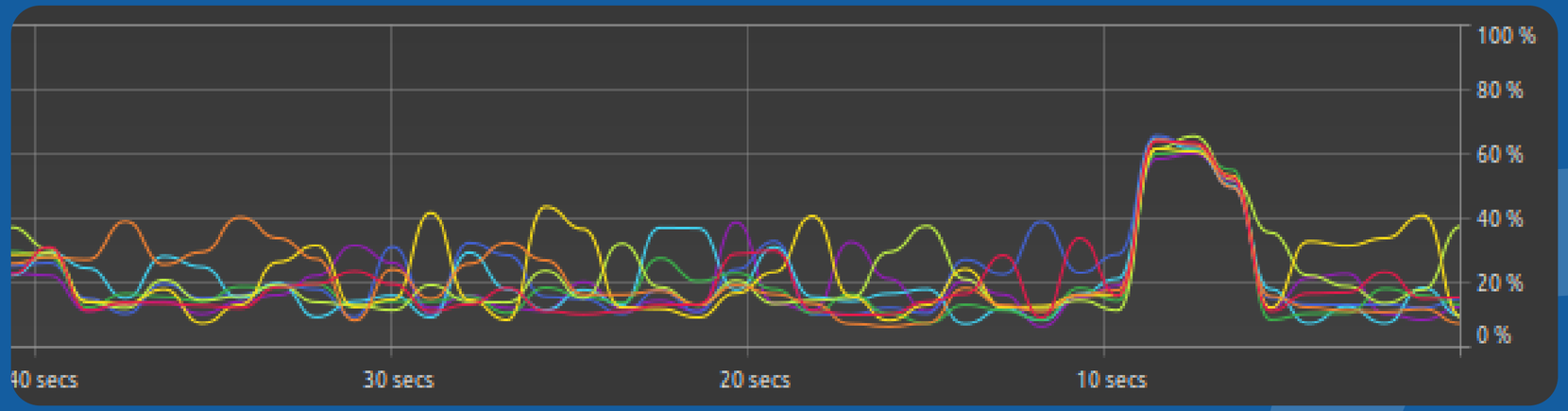
Análise de Desempenho (Tabuleiro 1)



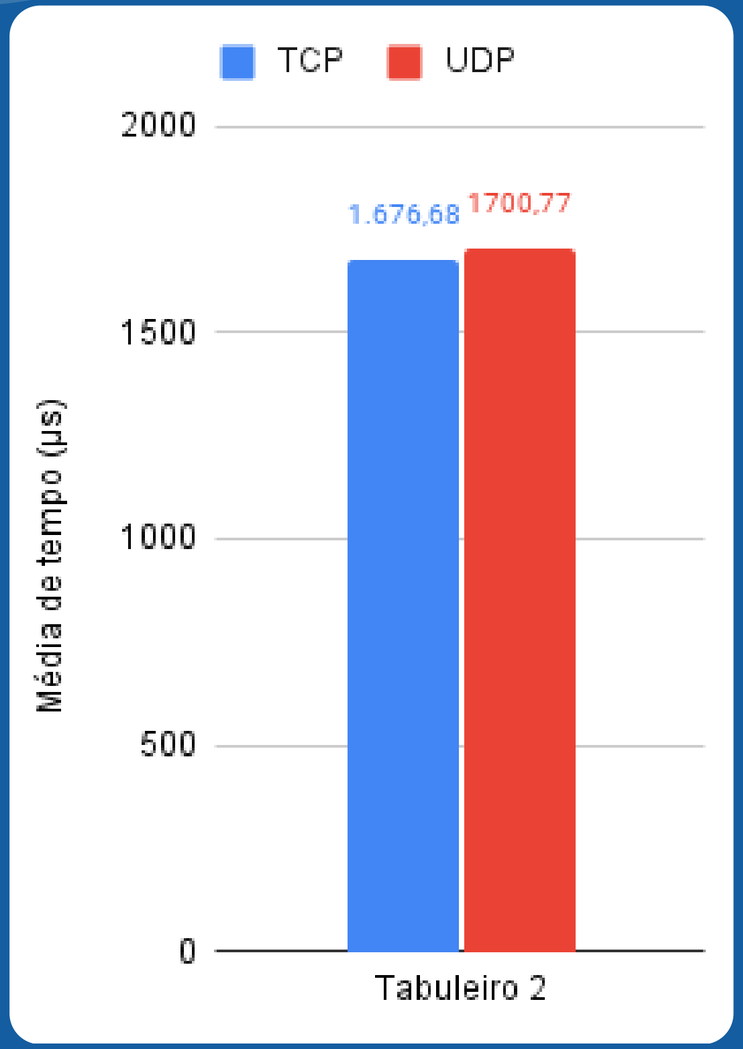
Uso de CPU - TCP



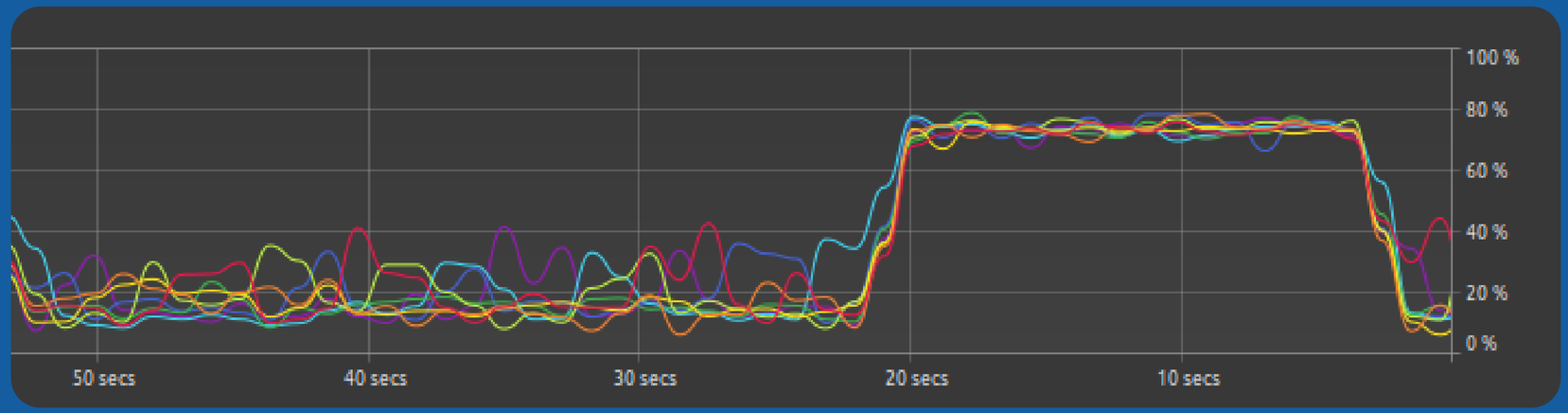
Uso de CPU - UDP



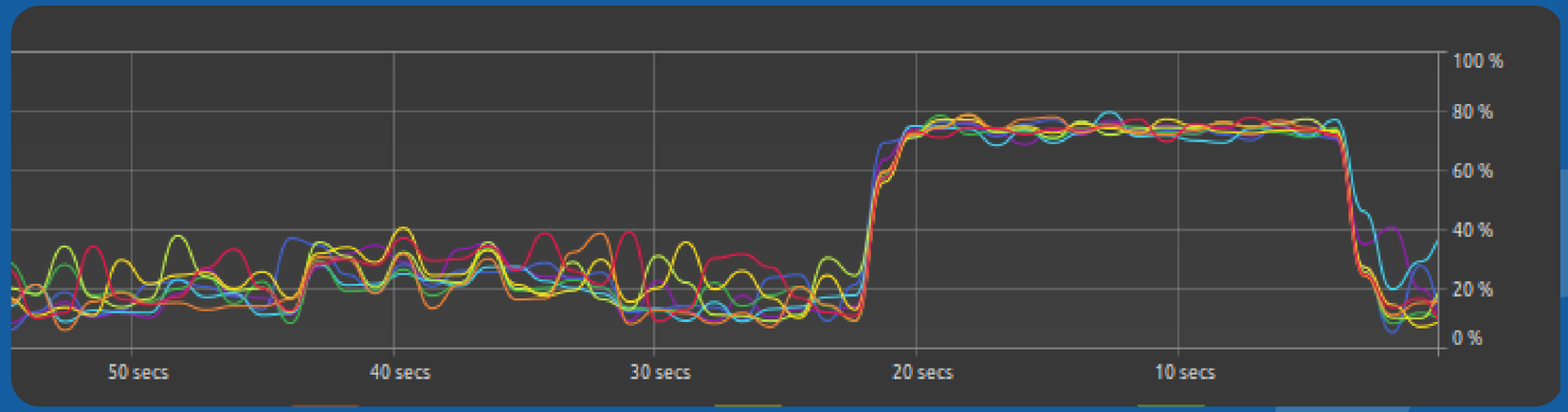
Análise de Desempenho (Tabuleiro 2)



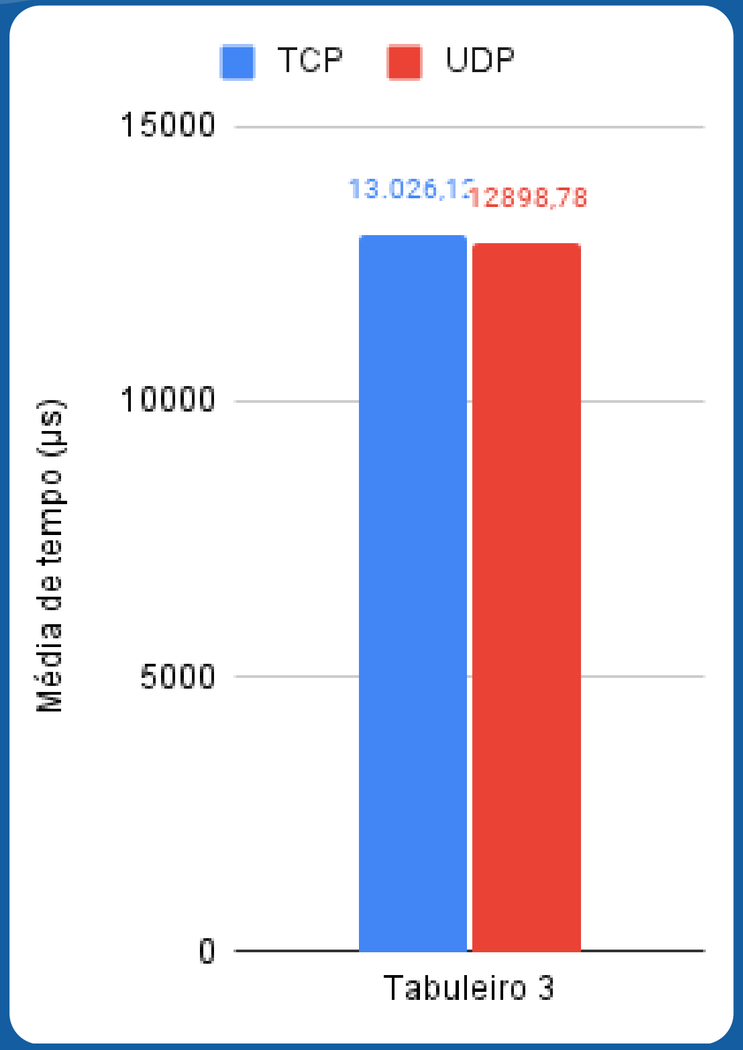
Uso de CPU - TCP



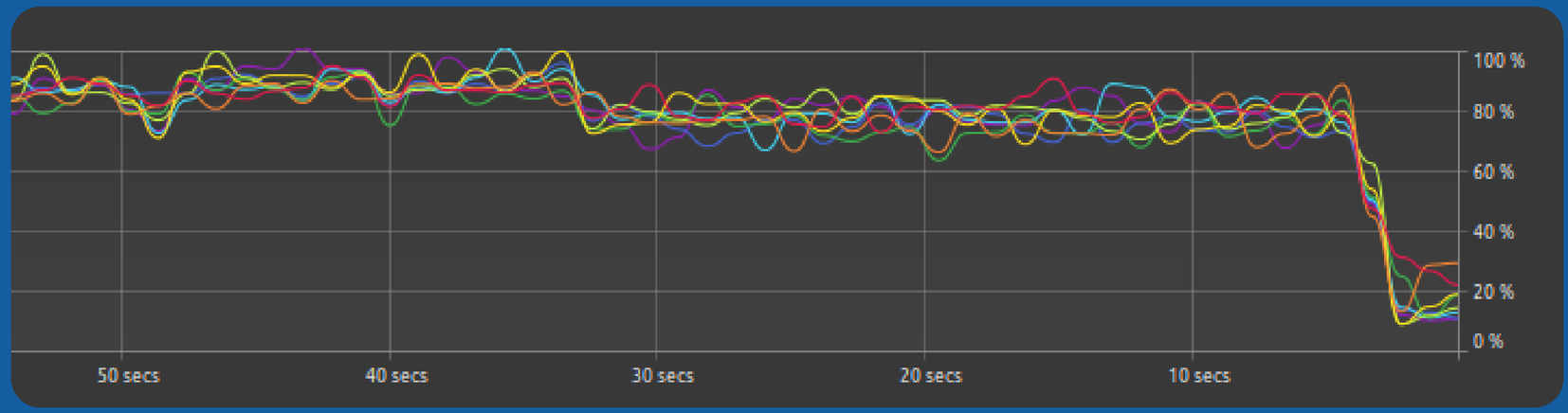
Uso de CPU - UDP



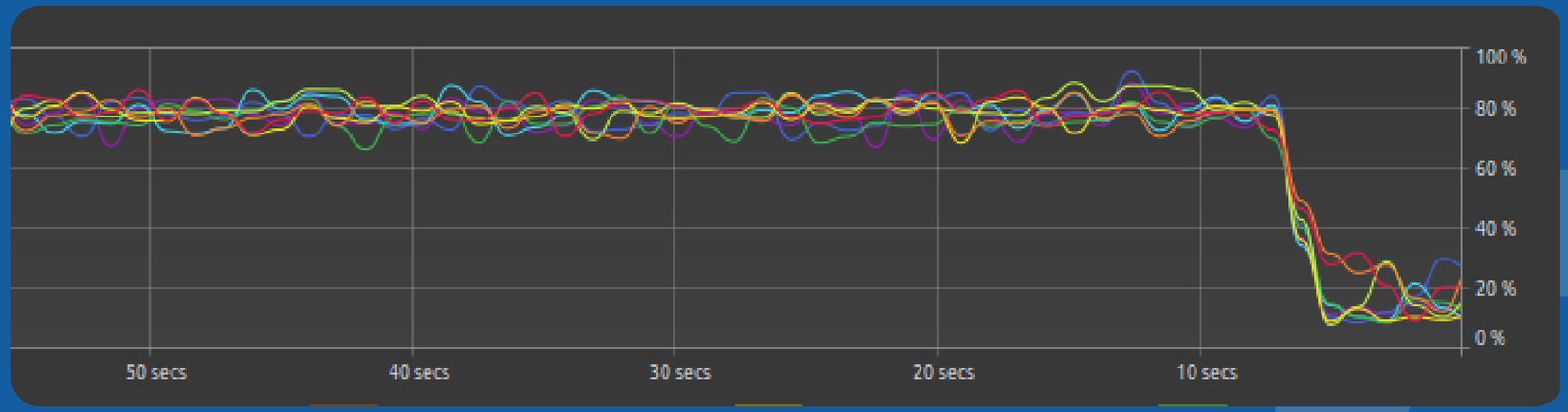
Análise de Desempenho (Tabuleiro 3)



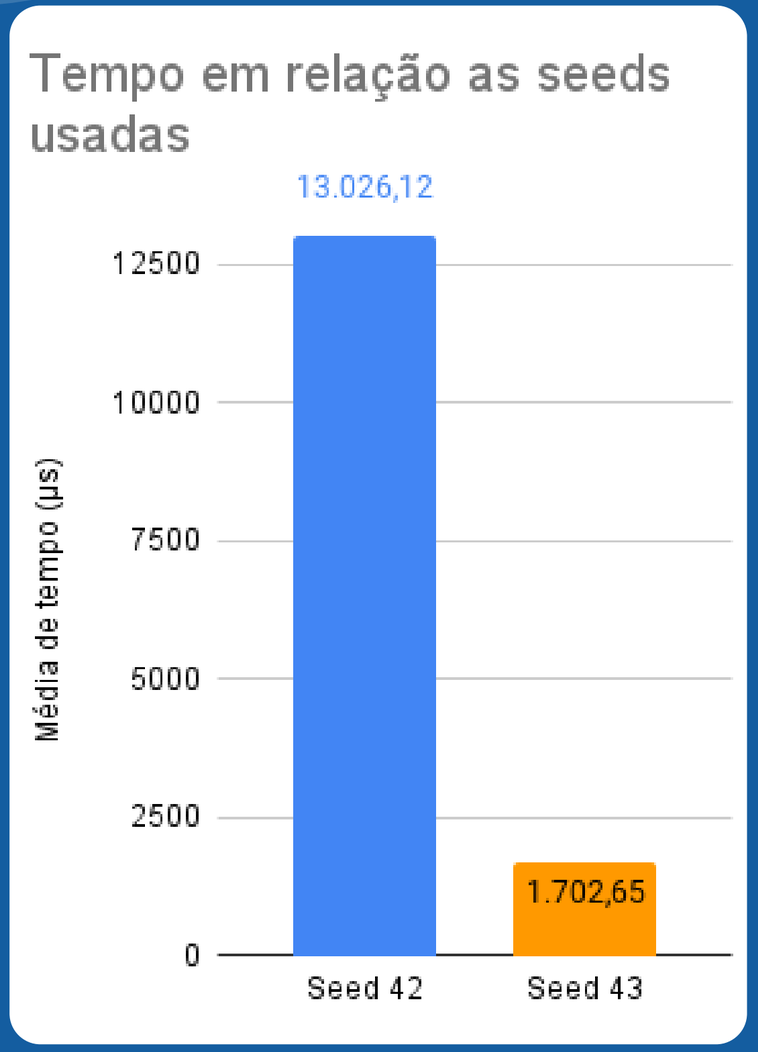
Uso de CPU - TCP



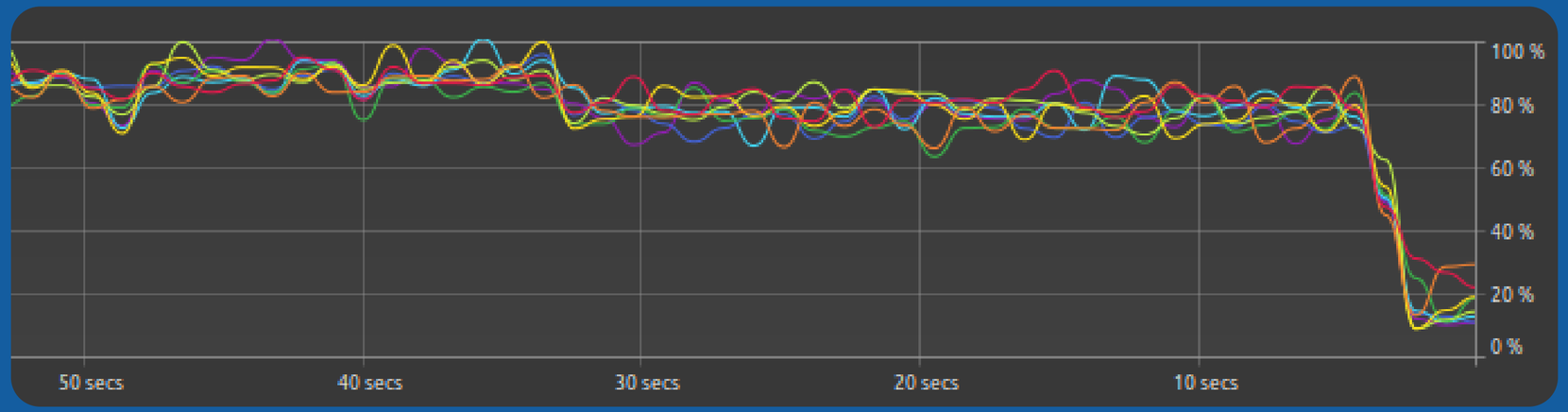
Uso de CPU - UDP



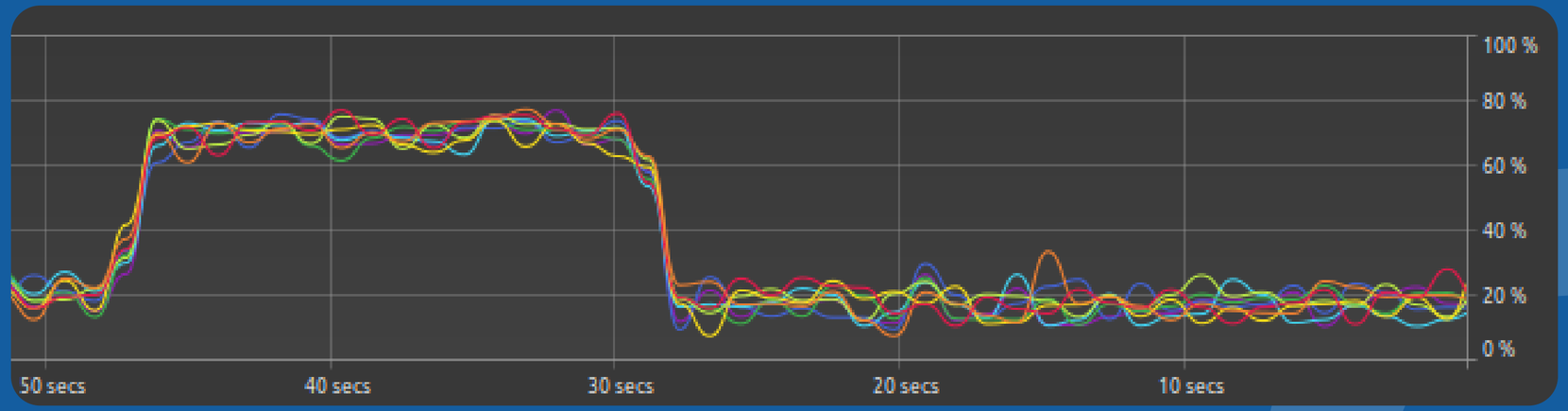
Análise de Desempenho (Tabuleiro 3)



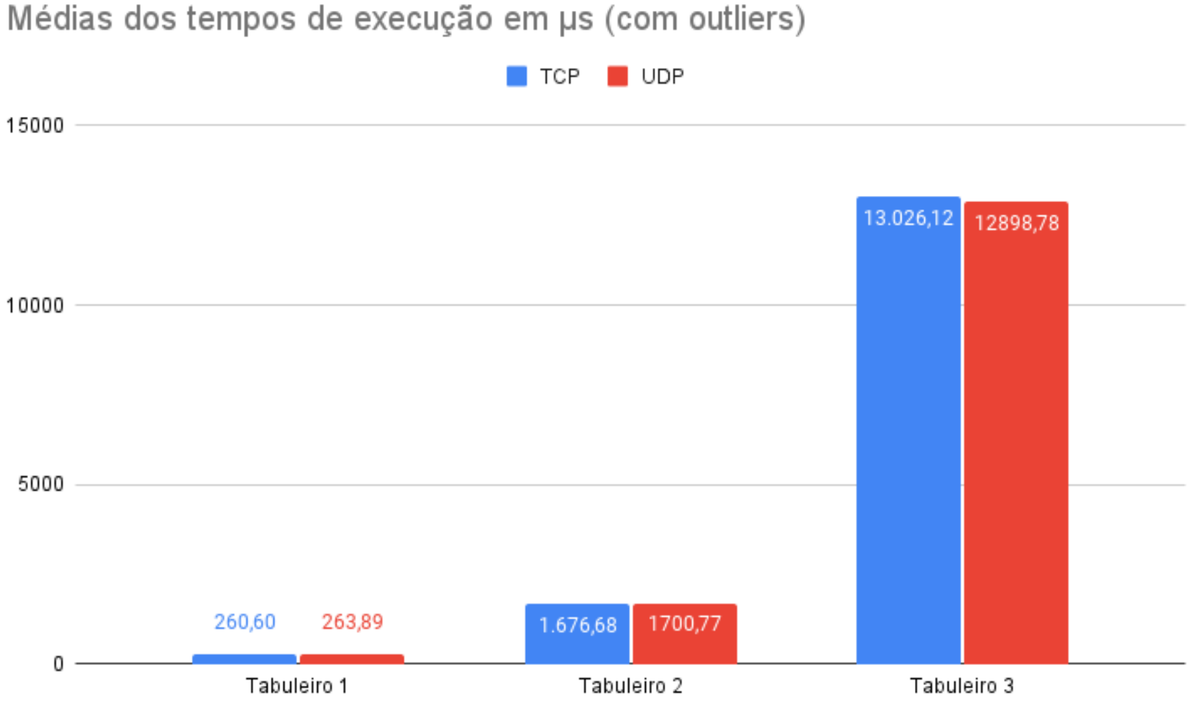
Uso de CPU - seed 42

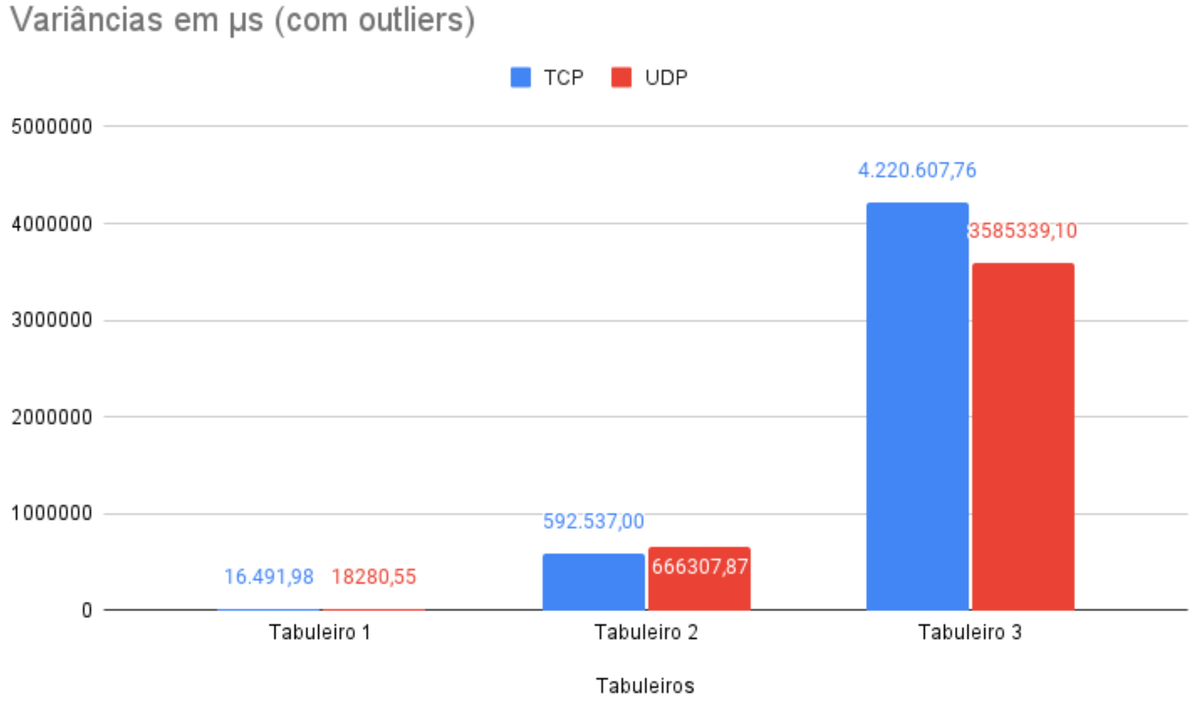


Uso de CPU - seed 43

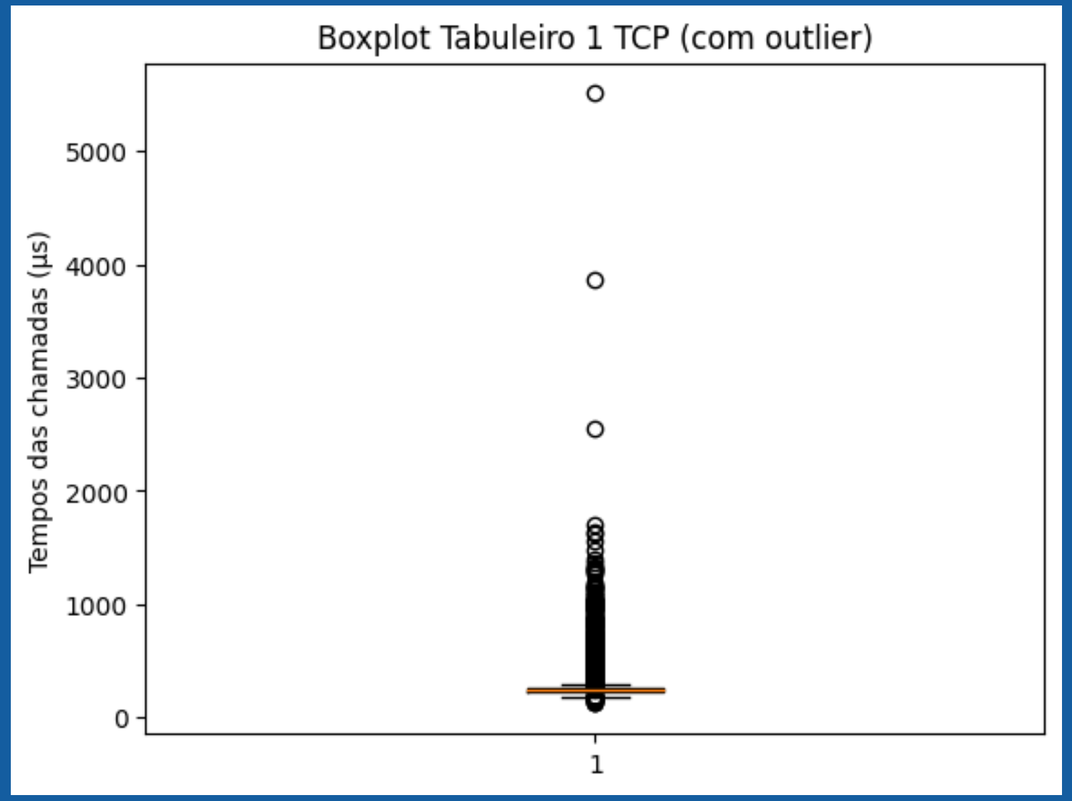


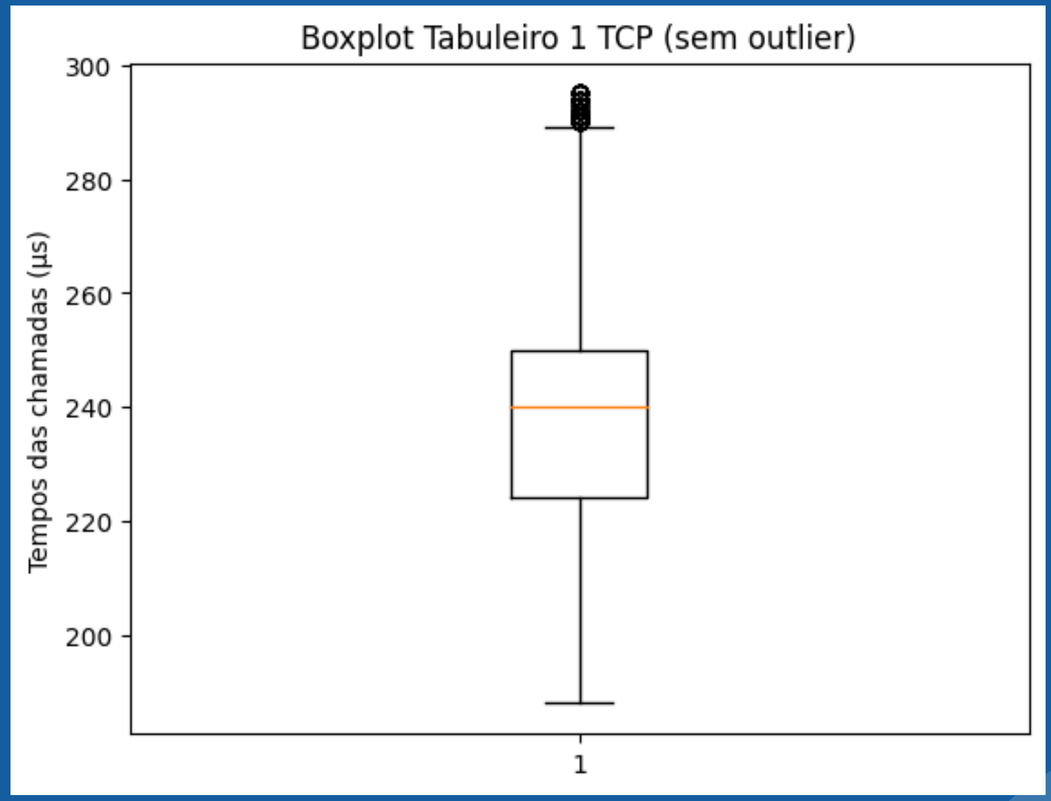
Estatísticas de Desempenho



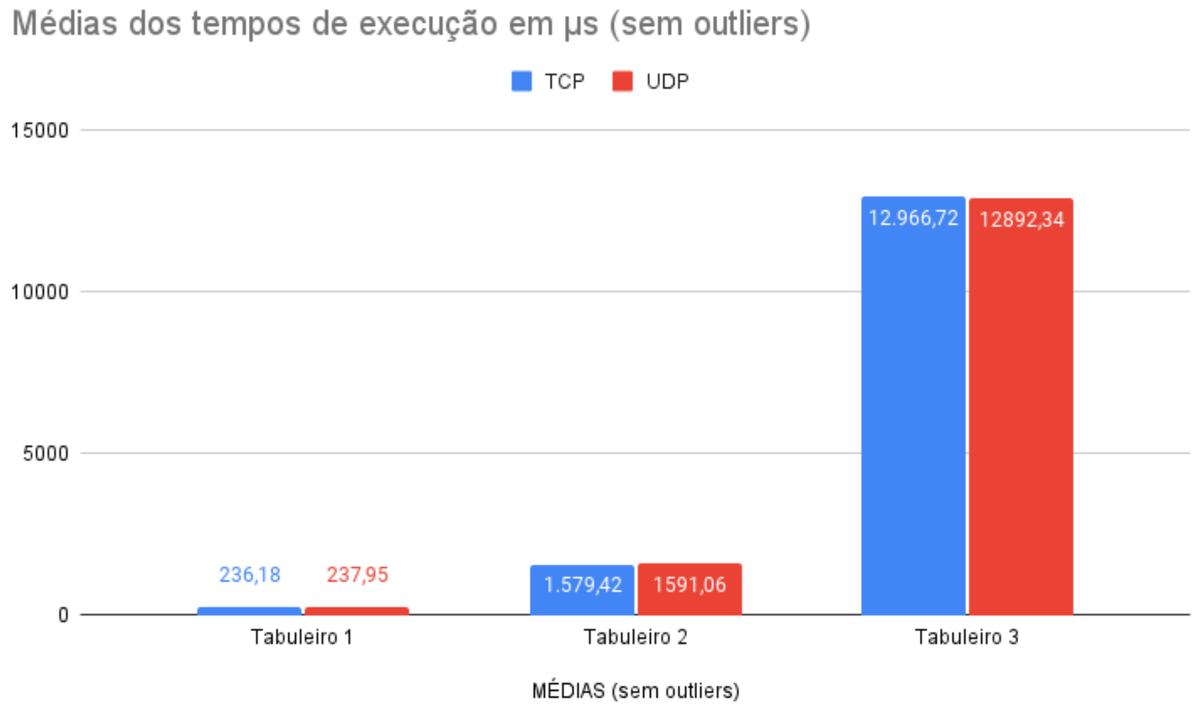


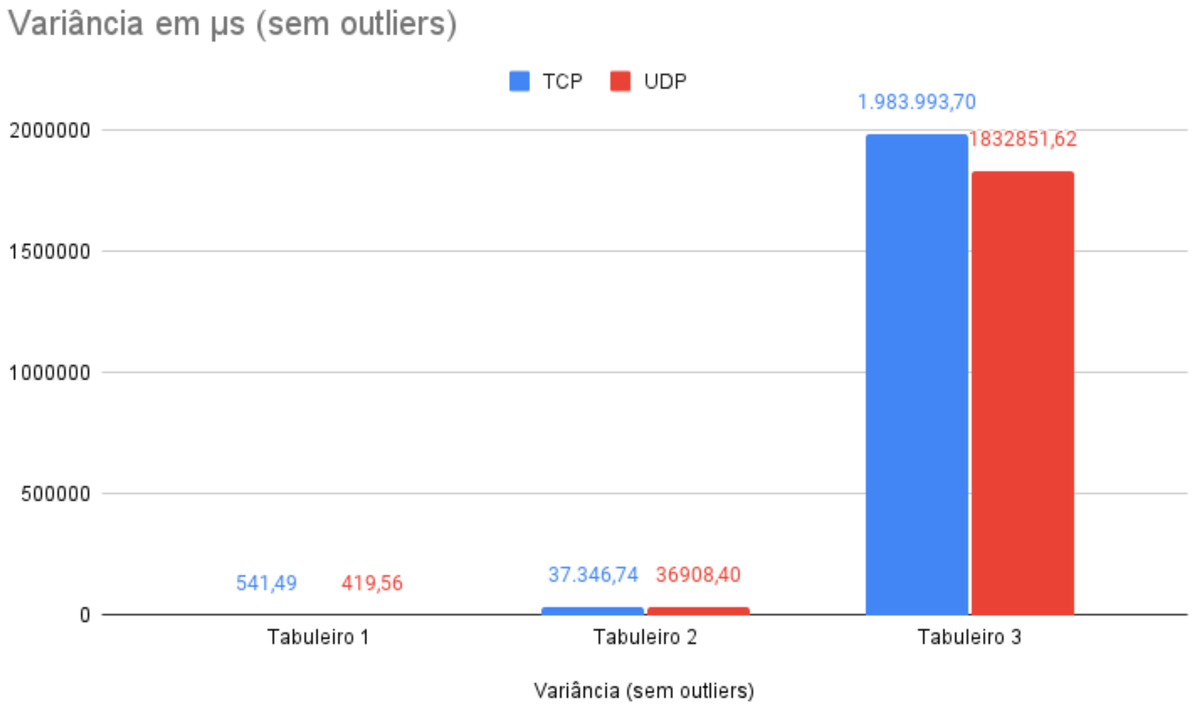
Remoção dos outliers





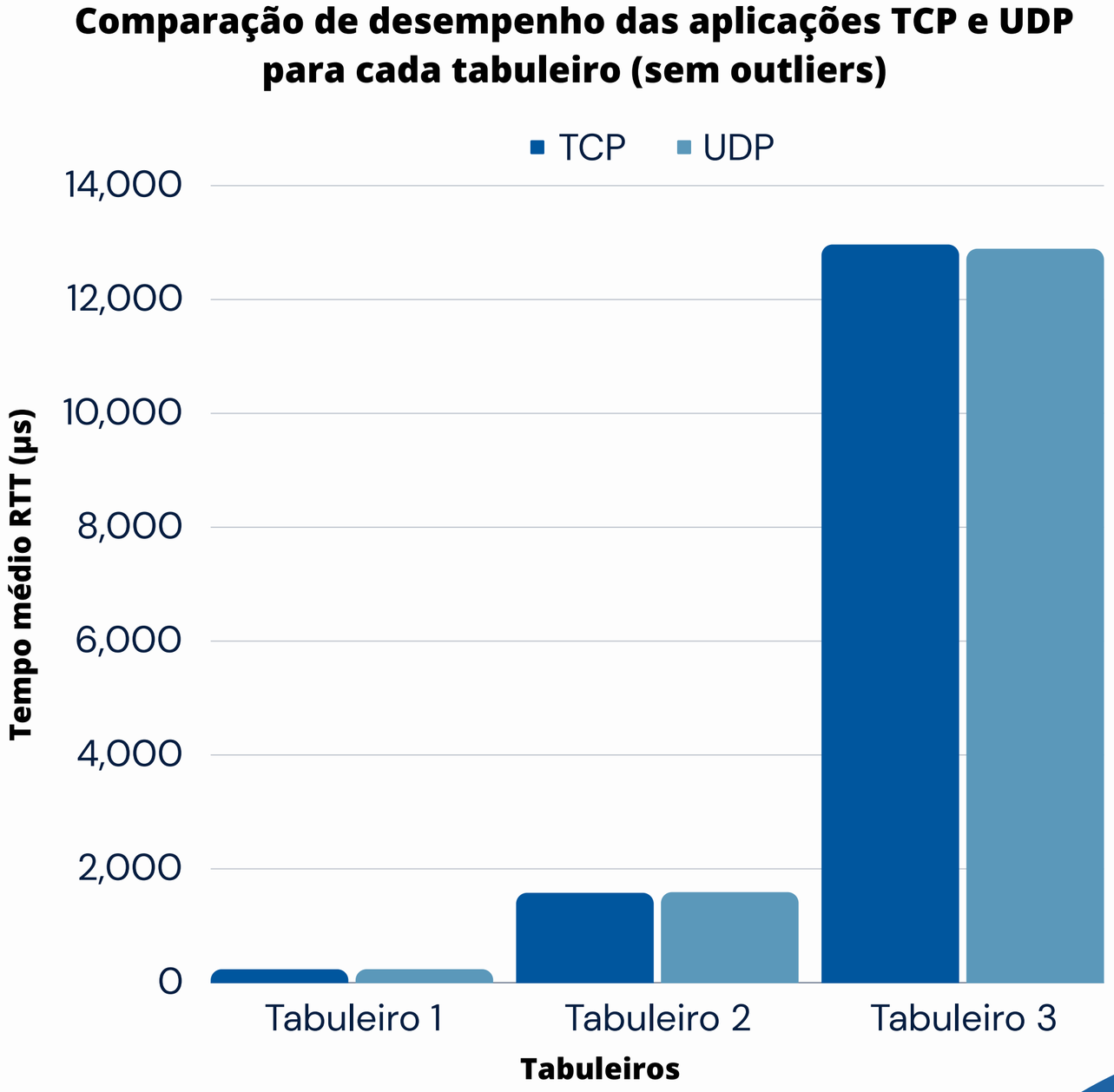
Remoção dos outliers





Resultados

É possível observar que apesar da consideravel diferença de implementaçaõ entre os protocolos TCP e UDP, o tempo medio RTT ficou bem proximo, isso se deve em grande parte a necessidade de um maior preprocessamento de cada nova requisição por parte do servidor UDP, pois como o protocolo UDP não é orientado a conexão, cada nova request recebida pelo servidor deve ser tratada isoladamente, exigindo uma identificação e processamento individual.



Obrigado