Projeto final CD

2020/2021

Paulo Pereira - 98430
Ricardo Rodriguez - 98388

Arquitetura Peer-to-Peer

Tentamos usar uma organização descentralizada em que os “slaves” comunicam entre si e com o servidor com o objetivo de decifrar a palavra passe correta. Sendo esta uma procura conjunta assim que um encontra a palavra passe os outros deixaram de a procurar.

Protocolo

A função alive tem como objetivo procurar quantos slaves se encontram ativos e assim detetar eventuais falhas e quedas de slaves.

A função slave info recebe informação das mensagens multicast vindas de outros slaves.

```
f slave_info(self):

  print("slave_info")

  header = self.multicast_sock.recv(4)
  print(header)
  if header:

      header = header.decode('utf-8').replace('f','')
      print(header)
      data = self.multicast_sock.recv(int(header))
      print(data)

      data = data.decode('utf-8')
      data = json.loads(data)

      print("Data:")
      print(data)
      slave_id = data["id"]

      #nova socket que coneta o slave atual a outro slave existente e adiciona ao dicionário
      if (slave_id not in self.slaves_info):
          self.slaves_info.append(slave_id) #adiciona um novo camarada
          print("novo slave recebido: " + str(slave_id))
      else:
          self.comparator_info_slaves.append(slave_id)

      print("return true")
      return True
  else:
      print("return false")
      return False
```

```
def alive(self):

    alive_message = {"id" : self.id}
    print(alive_message)
    data = json.dumps(alive_message).encode('utf-8')
    message_size = str(len(data))
    size_header = len(message_size)
    newheader =  'f'*(4-size_header) + message_size
    print(newheader)
    self.multicast_sock.sendto(newheader.encode('utf-8'), (MCAST_GRP, MCAST_PORT))
    self.multicast_sock.sendto(data, (MCAST_GRP, MCAST_PORT))

    # se receber mensagem do grupo multicast é porque já existem slaves no grupo, caso contrário este é o único (e passa a
    # ser o elemento central no P2P)
    self.comparator_info_slaves = {}
    self.reestructure = False
    otherSlaves = True
    while otherSlaves:
        otherSlaves = self.slave_info()

    #vai ver se houve algum slave que se desconetou
    for key in self.slaves_info:
        if key not in self.comparator_info_slaves:
            self.reestructure = True
            self.slaves_info.remove(key)

    if self.reestructure == True:
        self.reestruct_algorithm()
```

Resultados

Conseguimos estabelecer uma ligação multicast entre slaves e servidor que permite que estes se conheçam mutuamente. Conseguimos também