Comunicações por Computador

Licenciatura em Engenharia Informática

a106796: Tiago Miguel Barroso de Brito
a107292: Pedro Francisco Ferreira
a107363 Daniel Gonçalves Parente

Sumário

Introdução

Este projeto foi desenvolvido no âmbito da cadeira de Comunicações por Computador, 3º ano, 1º semestre da Licenciatura de Engenharia Informático, o projeto têm como objetivo principal o desenvolvimento de protocolos de aplicação, a comunicação entre um servidor e múltiplos clientes, aprofundar os conhecimentos sobre os protocolos de transporte **TCP** e **UDP**, e o desenvolvimento de api's de Observação sobre a forma de um servidor **HTTP**.

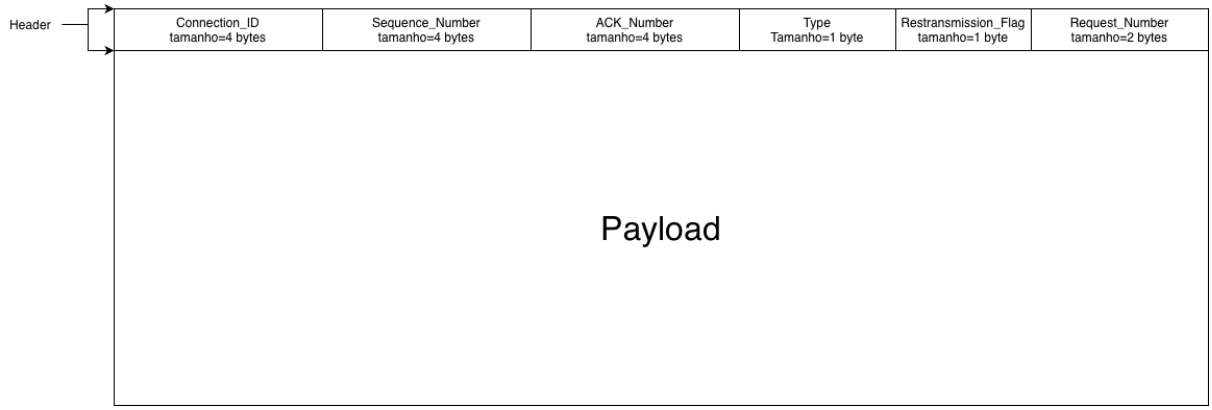
Protocolos

Os protocolos foram implementados de forma a cumprir os requisitos do enunciado, permitindo a comunicação entre rovers e a nave. No entanto, com o objetivo de generalizar a implementação e manter as camadas separadas, conceitos específicos como missão e pedidos de missão não são tratados diretamente nos protocolos. Em vez disso, apenas métodos fundamentais que asseguram a comunicação foram incluídos, tornando os protocolos flexíveis e reutilizáveis em outros contextos em que uma comunicação semelhante seja necessária.

MissionLink

O protocolo **MissionLink** visa transportar a informação necessária,permitindo o envio de missões da Nave-Mãe para os rovers e a receção de confirmações dos rovers.Este protocolo irá ser transportado através do protocolo de transporte **UDP** com mecanismos adicionais para garantir a entrega e a integridade dos pacotes a nível aplicacional, dada a natureza não confiável do **UDP**.

Design



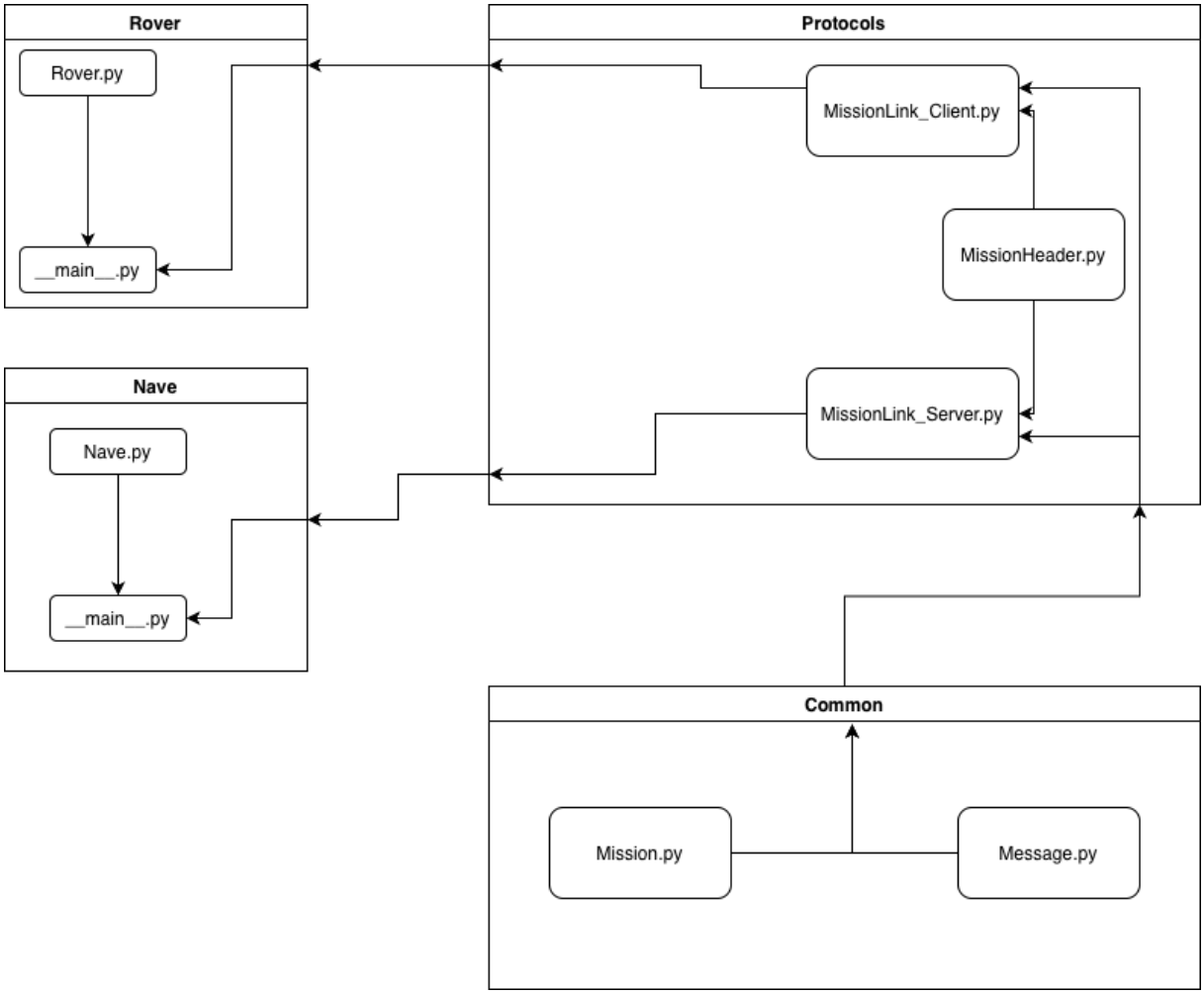
O protocolo aplicacional **MissionLink**, foi concebido para transportar múltiplos tipos de mensagens pelo que foi decidido que o protocolo iria abstrair a parte do *payload*, dando a responsabilidade de interpretar os dados contidos na *payload* para os utilizadores do protocolo, devido a isso a medidas de controlo, serão concebidas no cabeçalho, *header*, do protocolo, onde é possível ver os seguintes atributos:

- **Connection_ID**: identificador único da conexão entre o cliente-servidor
- **Sequence_Number**: identificador do número de sequência
- **ACK_Number**: identificador do número de ACK
- **Type**: identificador do tipo de mensagem(TYPE_ACK, TYPE_SYN, TYPE_SYNACK, TYPE_REQ, TYPE_DATA)
- **Restransmission_Flag**: identificador que diz se uma mensagem é uma retransmissão

- **Request_Number**: identificador do número do pedido feito pelo cliente

Arquitetura

A arquitetura do protocolo **MissionLink** teve como foco permitir estabelecer uma comunicação fiável entre uma nave-mãe e múltiplos rovers, para isso foi criados duas classes que iriam permitir o estabelecimento de uma conexão servidor-cliente, sendo chamadas respetivamente, **MissionLink_Server** e **MissionLink_Client**, que possuem os métodos e atributos necessários para gerir a conexão sobre o protocolo **MissionLink**. O cliente possui então duas formas distintas de comunicar com o servidor,através de um request(onde realiza um pedido e espera receber data do servidor), ou simplesmente enviando uma mensagem não esperando assim resposta do servidor.



O serviço **MissionLink_Server** possui 3 principais processos:

1. O **processo de receber as mensagens** enviadas pelos clientes, onde de forma assincronia o servidor recebe as mensagens enviadas pelos clientes, separando o *header* da *payload* e iniciando o processo de tratamento da mensagem.
2. O **processo de tratamento da mensagem** através do *header* da mensagem identifica o tipo da mensagem e executa a ação correspondente:
   - Se a mensagem for do tipo, **TYPE_ACK**, o servidor irá adicionar a mensagem à estrutura **pending**, estrutura essa que tem o propósito de guardar as mensagens do tipo **TYPE_ACK** referentes a uma determinada conexão.
   - Se a mensagem for do tipo, **TYPE_SYN**, o servidor irá começar o processo de **handshake**, onde será gerado um id de conexão único, e o servidor enviará uma mensagem do tipo **TYPE_SYNACK** com o respetivo id único para o endereço associado à mensagem recebida.
   - Se a mensagem for do tipo, **TYPE_DATA**, o servidor irá chamar a respetiva função callback que lhe fora atribuida e responderá ao cliente com uma mensagem do tipo **TYPE_ACK** para confirmar que a mensagem foi recebida.
   - Se a mensagem for do tipo, **TYPE_REQ**, o servidor responderá à mensagem recebida com uma mensagem do tipo **TYPE_ACK** e com uma mensagem de tipo **TYPE_DATA** contendo o resultado da funçao de callback para requests.
3. O **processo de enviar mensagens** aos clientes, onde de forma assíncrona envia as mensagens disponibilizadas numa fila de espera, *queue*, denominada por **send_queue**, para os respetivos clientes.

O serviço **MissionLink_Client** possui 2 principais processos:

1. O **processo de enviar um request** ao servidor, onde de forma assíncrona o cliente envia uma mensagem de tipo **TYPE_REQ** para o servidor e espera receber uma mensagem do tipo **TYPE_DATA**.
2. O **processo de enviar dados** ao servidor, onde de forma assíncrona o cliente envia para o servidor os dados pretendidos.

Estrutura de Mensagens

A estrutura das Mensagens no protocolo **MissionLink** envolve um 'header' que é consistente entre todas as mensagens que utilizam este protocolo, e a 'payload' onde será transportada a mensagem que será enviada e lida pelos destinatários. A payload pode ser distinguida em 3 tipos:

- Quando um cliente faz um pedido para obter uma missão, a payload será o identificador de missão executada previamente, onde caso o cliente não tenha missões prévias será enviado um identificador de missão igual a 0.

- Quando o servidor tenta enviar uma missão a um cliente que fez um pedido de missão, a payload será a composta pela missão a ser executada pelo cliente. A missão possui os seguintes atributos:
  - mission_id: identificador da missão enviada, representado por um valor inteiro
  - geographic_area: área onde missão irá decorrer, representado por um tuplo de 3 inteiros (posição x, posição y, raio)
  - task: tarefa a ser executada, representada por uma string
  - max_duration: duração que a tarefa demora a completar, representada por um inteiro
  - atualization_interval: intervalo em segundos que o rover deverá usar para enviar o dados sobre o estado da missão para a nave-mãe, representado por um inteiro.
  - status: estado da missão, representado por um inteiro (0 = Inactive, 1 = Started, 2 = Completed)

- Quando o cliente envia o estado da missão, a payload será composta pelo estado da missão, representado pela classe **Message_Status**, possuindo os seguintes atributos:
  - rover_id: identificador do rover que está a executar a missão, representado por um inteiro
  - mission_id: identificador da missão enviada, representado por um valor inteiro
  - max_duration: duração que a tarefa demora a completar, representada por um inteiro
  - current_duration: duração que o rover esteve a executar a tarefa, representado por um inteiro
  - status: estado da missão, representado por um inteiro (0 = Inactive, 1 = Started, 2 = Completed)
  - completion: percentagem da missão completada, representada por um inteiro entre 0 e 100

Mecanismos de Controlo

Os mecanismos de controlo utilizados pelo protocolo **MissionLink**, foram:

1. A utilização de um processo inicial de conexão onde é feito um "three-way handshake", ou seja, um cliente tenta iniciar a conexão com o servidor enviando ao servidor um pacote com o tipo **TYPE_SYN**, caso o pacote tenha chegado ao servidor, o servidor responderá com um pacote do tipo **TYPE_SYNACK**, com o id de conexão para o cliente, e caso o cliente receba o pacote a comunicação a conexão estará estabelecida.Finalmente o cliente envia um pacote do tipo **TYPE_ACK** confirmando a receção. Somente após este processo, o cliente poderá interagir com o servidor.
2. A utilização de mensagens de "Acknowledgement", ACK's, para garantir que as mensagens transmitidas chegam ao destinatário de forma ordenada.
3. Retransmissão de pacotes, de forma a garantir a entrega do pacote. O protocolo guardará sempre o **RTT**(Round-Trip Time) do último pacote enviado e seu respetivo "Acknowledgement", sendo assim calculado um **Retransmission Timeout** .Se nenhum "Acknowledgement" chegar dentro desse tempo, será retransmitido o pacote.

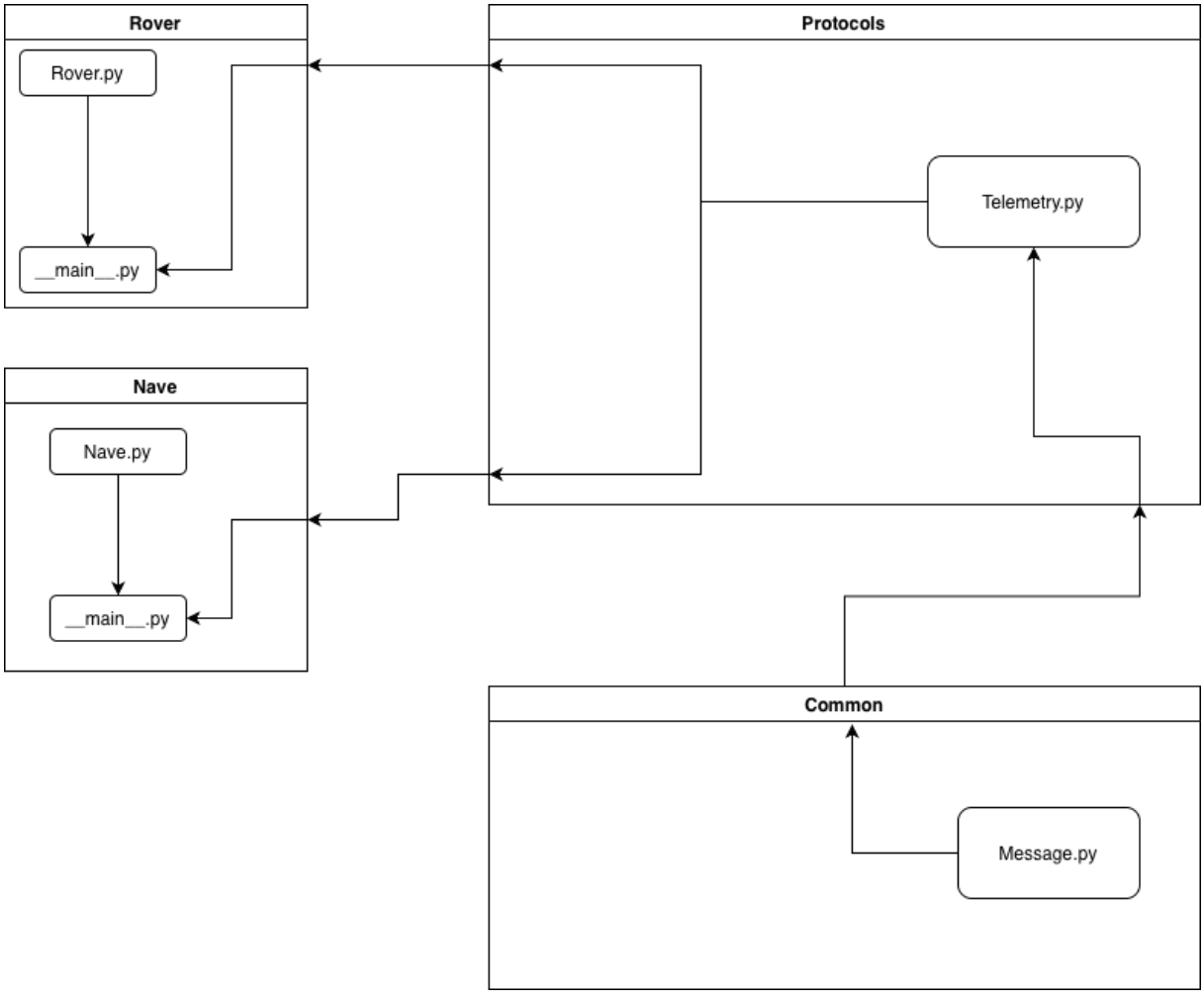
TelemetryStream

O protocolo **TelemetryStream** visa transportar a informação relativa aos estados dos rovers para a nave-mãe. O protocolo será transportado através do protocolo **TCP**, utilizando um mecanismo que desconecta os clientes do servidor caso estejam inativos.

Design

Devido ao protocolo **TelemetryStream** atuar sobre a camada de transporte **TCP**, foi decidido que o design do protocolo deveria ser o mais simples possível, pelo que o protocolo **TelemetryStream** é apenas composto por 2 campos: **tamanho da *payload*** + *payload*.

Arquitetura



O serviço de gestão do protocolo **TelemetryStream** é disponibilizado no modulo **Telemetry**, podendo ser inicializado com o modo para clientes e servidores. Caso o modo inicializado seja o cliente, será possível conectar e enviar mensagens para o servidor através do endereço de ip e do *port* utilizado na inicialização da classe **Telemetry**, podendo ainda terminar a ligação cliente--servidor. Caso o modo inicializado seja o servidor, será possível inicializar o servidor, onde o servidor mantém-se à espera de receber pedidos de clientes a requisitar o estabelecimento de uma conexão, enquanto lê as mensagens enviadas pelos clientes, mantendo um dicionário onde para cada endereço de ip conectado ao servidor está associado o tempo em que decorreu a última mensagem recebida pelo servidor, tornando assim possível remover a clientes que estejam inativos.

Estrutura de Mensagens

A estrutura das Mensagens no protocolo **TelemetryStream** envolve um 'header' que é consistente entre todas as mensagens que utilizam este protocolo, e a 'payload' onde será transportada a mensagem que será enviada e lida pelos destinatários.

- A payload terá a forma de a estrutura de dados de **Message_Telemetry** possuindo os seguintes atributos:
  - **rover_id**: representa o id do rover através de um inteiro
  - **rover_status**: representa o estado do rover, através de um inteiro entre 0,1,2 representado respetivamente os seguintes estados (0 = Waiting), (1 = In Mission), (2 = Walking).
  - **rover_position**: posição do rover, representado por um tuplo de inteiros x,y

Mecanismos de Controlo

Os mecanismos de controlo usados foram:

1. A utilização de um mecanismo de término prévio da conexão caso um cliente não disponibilize relatórios passado um determinado tempo.

Api de Observação

A api de observação disponível por parte da nave mãe, estabelece um servidor http que procura por pedidos feitos por clientes atendendo-os de forma sequencial, enviando as informações pedidas através da estrutura de dados json.

Endpoints

A api de observação disponibiliza 3 endpoints, para serem usados pelo o cliente nos pedidos efetuados pelo mesmo:

1. **/active_rovers**: Disponibiliza informações sobre os rovers
2. **/missions**: Disponibiliza informações sobre as missões
3. **/reports**: Disponibiliza informações sobre as mensagens de telemetria

As informações disponibilizadas pela api de observação, seguem a estrutura de dados json obtendo a seguinte forma em cada endpoint:

active rovers
[{“rover_id”: 252, “status”: “In Mission”, “position”: “[5, 6]”, “last_update”: “2025-12-07 17:36:41”}]

missions
[{“mission_id”: 1, “geographic_area”: “(5, 6, 1)”, “task”: “Clean_Area”, “max_duration”: 30, “atualization_interval”: 15, “status”: “Inactive”}]

reports
[{“id”: 14, “rover_id”: 252, “mission_id”: 1, “mission_status”: “In Mission”, “current_duration”: 30, “completion”: 100, “timestamp”: “2025-12-07 17:36:50”}]

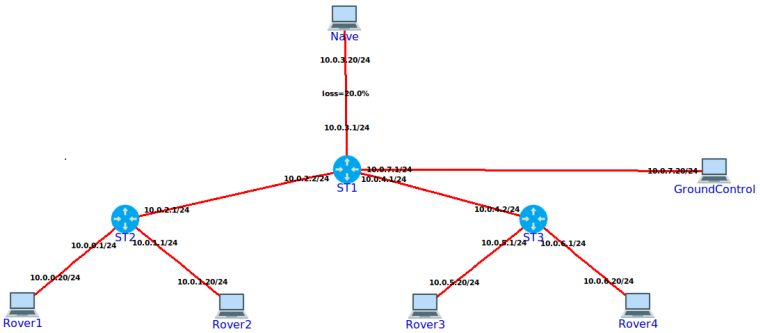
Testes Realizados

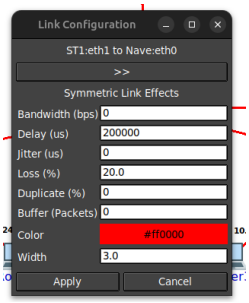
Para podermos comprovar a fiabilidade dos protocolos foram realizados 2 testes. Todos os testes tiveram duração suficiente para que os rovers a serem utilizados concluissem 1 missão, sendo o teste terminado aquando do pedido da 2 missão.

1 - Perda de pacotes e latência

Este teste possui como objetivo principal comprovar a fiabilidade dos protocolos em sitações com perda de pacotes e latência a níveis notórios. Foi então definida uma perda de 20% e uma latência de 200 ms. Neste teste, teremos apenas um rover(nodo Rover1), a comunicar com a Nave

Topologia CORE do teste





Resultados

O teste procedeu sem erros, funcionando perfeitamente a lógica pretendida de pedido de missão por parte do Rover, envio de estados da missão bem como em paralelo o envio de dados telemetria. A imagem apresentada embaixo trata-se de uma captura através da ferramenta **wireshark** no nodo **Rover1**.

No.	Time	Source	Destination	Protocol	Length	Info
16	16.469006213	10.0.0.20	10.0.3.20	UDP	58	35476 → 50000 Len=16
22	21.472792092	10.0.0.20	10.0.3.20	UDP	58	35476 → 50000 Len=16
33	31.478869413	10.0.0.20	10.0.3.20	UDP	58	35476 → 50000 Len=16
51	51.485189637	10.0.0.20	10.0.3.20	UDP	58	35476 → 50000 Len=16
52	51.886031568	10.0.3.20	10.0.0.20	UDP	58	50000 → 35476 Len=16
53	51.886503225	10.0.0.20	10.0.3.20	UDP	58	35476 → 50000 Len=16
54	51.886534777	10.0.0.20	10.0.3.20	UDP	58	35476 → 50000 Len=16
56	52.287096355	10.0.3.20	10.0.0.20	UDP	58	50000 → 35476 Len=16
57	52.318481746	10.0.3.20	10.0.0.20	UDP	107	50000 → 35476 Len=65
58	52.318832343	10.0.0.20	10.0.3.20	UDP	58	35476 → 50000 Len=16

Nesta imagem, com ajuda do campo length,source e destination conseguimos analisar o fluxo de um request feito pelo rover :
1ªLinha - Inicialização do processo de Handshake,tendo sido enviado apenas o MissionHeader(16bytes) com TYPE_SYN

2ªLinha a 4ªLinha - Retransmissão da mensagem inical

5ªLinha - Receção de mensagem proveniente da **Nave** apenas com o MissionHeader(16bytes) com TYPE_SYNACK

6ªLinha - Envio de "Acknowledgement", através do MissionHeader(16bytes) com TYPE_ACK.Dá-se assim por terminado o Handshake

7ªLinha - Envio de request por parte do **Rover1** atraves do MissionHeader(16bytes) com TYPE_REQ.

8ªLinha - Receção de "Acknowledgement" proveniente da **Nave** apenas com o MissionHeader(16bytes) com TYPE_ACK.

9ªLinha - Receção da mensagem proveniente da **Nave** contendo o MissionHeader mais a informação da missão.

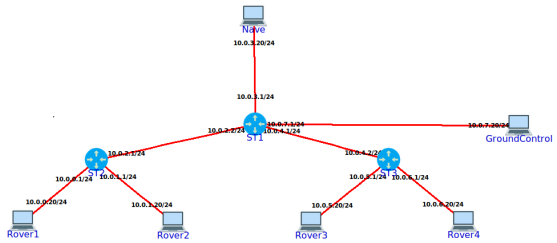
10ªLinha - Envio de "Acknowledgement" para **Nave** dando assim por encerrado o processo do **Rover1** de pedir uma missão.

Através de "prints" realizados pela Nave podemos também verificar o número de pacotes retransmitidos que recebeu neste período de tempo bem como o número de pacotes recebidos com os dados de telemetria.

2- Múltiplos Rover em paralelo

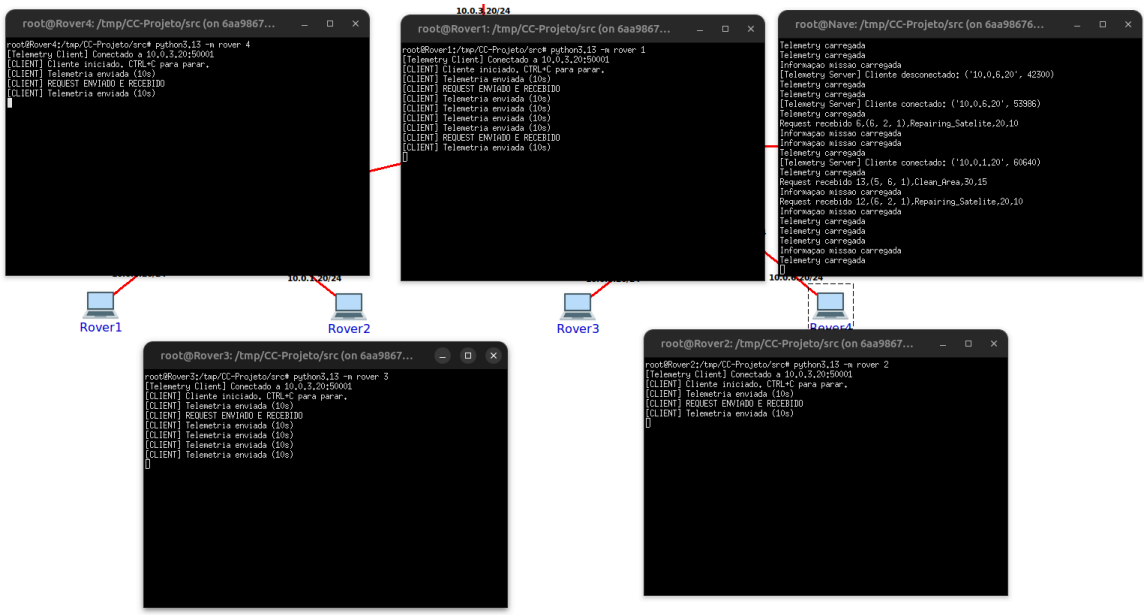
Este segundo teste possui como objetivo principal assegurar a capacidade da **Nave-Mãe** de processar missões e telemetria em simultâneo. Neste teste, os quatro Rover comunicam com a Nave-Mãe de forma paralela.

Topologia CORE do teste

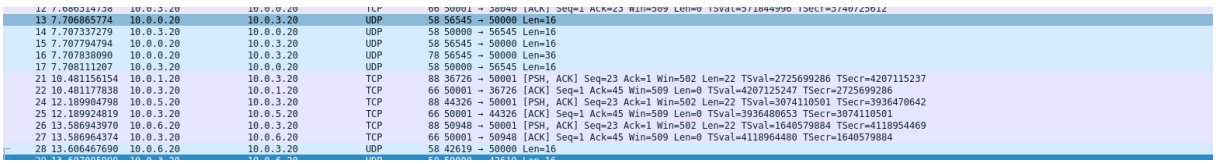


Resultados

O teste procedeu sem erros, funcionando perfeitamente a lógica pretendida de pedido de missões por parte dos Rover, envio de estados da missão bem como em paralelo o envio de dados telemetria. Podemos ver de seguida os terminais dos 4 Rover bem como da Nave-Mãe enquanto o teste era realizado.



Semelhante ao primeiro teste, foi capturado tráfego através da ferramente Wireshark na Nave-Mãe onde podemos ver através dos campos source,destination, esta a comunicar com os diferentes Rover.



13	7.706865774	10.0.0.20	10.0.3.20	UDP	58 56545 → 50000 Len=16
14	7.707337279	10.0.3.20	10.0.0.20	UDP	58 50000 → 56545 Len=16
15	7.707794794	10.0.0.20	10.0.3.20	UDP	58 56545 → 50000 Len=16
16	7.707838090	10.0.0.20	10.0.3.20	UDP	78 56545 → 50000 Len=36
17	7.708111207	10.0.3.20	10.0.0.20	UDP	58 50000 → 56545 Len=16
21	10.481156154	10.0.1.20	10.0.3.20	TCP	88 36726 → 50001 [PSH, ACK] Seq=23 Ack=1 Win=502 Len=22 TSval=2725699286 TSecr=4207115237
22	10.481177838	10.0.3.20	10.0.1.20	TCP	66 50001 → 36726 [ACK] Seq=1 Ack=45 Win=509 Len=0 TSval=4207125247 TSecr=2725699286
24	12.189904798	10.0.5.20	10.0.3.20	TCP	88 44326 → 50001 [PSH, ACK] Seq=23 Ack=1 Win=502 Len=22 TSval=3074110501 TSecr=3936470642
25	12.189924819	10.0.3.20	10.0.5.20	TCP	66 50001 → 44326 [ACK] Seq=1 Ack=45 Win=509 Len=0 TSval=3936480653 TSecr=3074110501
26	13.586943970	10.0.6.20	10.0.3.20	TCP	88 50948 → 50001 [PSH, ACK] Seq=23 Ack=1 Win=502 Len=22 TSval=1640579884 TSecr=4118954469
27	13.586964374	10.0.3.20	10.0.6.20	TCP	66 50001 → 50948 [ACK] Seq=1 Ack=45 Win=509 Len=0 TSval=4118964480 TSecr=1640579884
28	13.606467690	10.0.6.20	10.0.3.20	UDP	58 42619 → 50000 Len=16

Reflexão Crítica

O projeto permitiu desenvolver os conhecimentos sobre os diversos protocolos que atuam sobre a camada de transporte ,**TCP** e **UDP**, tornando possível entender os mecanismos de controlo que os diferenciam. O projeto também permitiu desenvolver o conhecimento sobre o protocolo **HTTP** no desenvolvimento de uma ***API*** de observação. No geral, consideramos que projetamos protocolos sólidos, fiáveis e rápidos cumprindo assim com o pedido no enunciado.