



Universidade do Minho

Braga, Portugal

TRABALHO PRÁTICO 2 - RELATÓRIO

COMUNICAÇÕES POR COMPUTADORES

Protocolos de Comunicação para Simulador Espacial

Grupo PL401

Engenharia Informática 2025/26

Equipa de Trabalho:

A106936 - Duarte Escairo Brandão Reis Silva

A106932 - Luís António Peixoto Soares

A106856 - Tiago Silva Figueiredo

13 Dezembro 2025

Índice

1. Objetivos	1
2. Protocolos	2
2.1. Mensagens	2
2.1.1. Reports	2
2.2. MissionLink	3
2.2.1. Definição da Missão	3
2.2.2. Implementação do Protocolo	4
2.3. TelemetryStream	7
2.3.1. Definição do Estado do Rover (Telemetria)	7
2.3.2. Implementação do Protocolo	8
2.4. API de Observação	8
2.4.1. Nave-Mãe	8
2.4.2. GroundControl	9
3. Interface do Ground Control	10
3.1. Interface	10
3.2. Execução	12
4. Testes na topologia	13
5. Conclusão	14

1. Objetivos

No âmbito da Unidade Curricular de Comunicações por Computador, foi-nos proposto este projeto que teve como objetivo desenvolver protocolos aplicacionais sobre UDP e TCP, para a comunicação entre Rovers e uma Nave-Mãe, num contexto de uma missão espacial. Para além disto, foi-nos também proposto o desenvolvimento de uma API de Observação disponibilizada pela Nave-Mãe, que seria consumida por um nó na topologia designado de Ground Control. Esta API deveria estar acessível através de HTTP ou WebSockets/SSE.

Ao longo das várias semanas que tivemos para o desenvolvimento do projeto, pensamos, desenhamos e implementamos os vários protocolos aplicacionais referidos, e desenvolvemos uma interface gráfica para o nó Ground Control, de modo a que as informações que a ele chegassem, pudessem ser vistas de uma forma mais clara e natural.

Para testar e demonstrar os protocolos e as demais funcionalidades do programa, criamos uma topologia de teste, com diversos níveis de dificuldade, onde se vêem os protocolos em ação.

Neste relatório iremos descrever em detalhe os protocolos MissionLink e TelemetryStream, especificar a API de Observação e mostrar o resultado dos testes obtidos.

2. Protocolos

2.1. Mensagens

Para este projeto, começamos por decidir que em ambos os protocolos iria circular uma mensagem comum, e definimos que essa mensagem seria composta por um cabeçalho e por um payload. Para o cabeçalho definimos os seguintes campos:

- Tipo de Mensagem
- Identificador da origem
- IP da origem
- Porta da origem
- Identificador do destino
- IP do destino
- Porta do destino

Tipo de Mensagem		
ID de Origem	IP de Origem	Porta de Origem
ID de Destino	IP de Destino	Porta de Destino
Payload		

Figura 1: Estrutura da mensagem usada nos protocolos

Desta forma, conseguimos gerir facilmente as mensagens que circulam, especialmente no lado da Nave-Mãe, pois aqui é importante conhecer as informações do Rover que envia as mensagens, já que para paralelizar o processamento da informação dos vários Rovers, é necessário direcionarmos a mensagem para a thread dedicada a cada Rover.

Para estas mensagens optamos por não dividir em segmentos, já que o tamanho total do byte array com os campos preenchidos, nunca ultrapassaria o tamanho máximo do payload do protocolo UDP.

2.1.1. Reports

Como iremos ver mais à frente, algumas tarefas das missões requeriam que os Rovers enviassem periodicamente reports para a Nave-Mãe, através do protocolo MissionLink, e neste caso, esses reports seriam imagens.

Tendo as imagens tamanhos superiores ao tamanho máximo do payload do protocolo UDP, optamos por dividi-las em pedaços de 4096 bytes, às quais chamamos frames. Com esta configuração, era necessário garantir que as frames que chegavam à Nave-Mãe eram depois juntas na ordem correta para reconstruir a imagem. Por estes motivos adaptamos a mensagem para este caso específico, acrescentando alguns campos adicionais ao cabeçalho:

- Identificador do Report
- Número de Frames do Report
- Número de Sequência

Tipo de Mensagem		
ID de Origem	IP de Origem	Porta de Origem
ID de Destino	IP de Destino	Porta de Destino
ID de Report	Nº Frames	Nº Sequência
Payload		

Figura 2: Estrutura do report usado no MissionLink

2.2. MissionLink

O protocolo MissionLink que corre sobre UDP, está presente tanto nos Rovers, como na Nave-Mãe, e em ambos, existe uma thread responsável por processar as mensagens relativas a este protocolo, o que permite que tanto este protocolo como o TelemetryStream, que corre sobre TCP, podem correr em simultâneo sem existirem conflitos.

Para além disso, no lado da Nave-Mãe, quando são recebidas mensagens no protocolo MissionLink, é verificado que Rover foi responsável por enviar cada uma delas, sendo depois reencaminhadas para uma thread responsável por tratar das mensagens de cada um dos Rovers.

Desta forma é garantido que mensagens de vários Rovers podem ser processadas em paralelo.

2.2.1. Definição da Missão

Outro dos aspetos que definimos previamente foi a estrutura das missões que seriam enviadas da Nave-Mãe para os Rovers através do protocolo MissionLink. Ficou decidido que essas missões teriam os seguintes campos:

- Identificador da Missão
- Dois pares de coordenadas $(x_1, y_1)(x_2, y_2)$ que delimitam a área da missão
- Tarefa a realizar
- Duração da missão (em minutos)
- Frequência de Updates (em segundos)

As tarefas das missões podem variar entre as seguintes: Tirar Fotos, Gravar Vídeo, Explorar, Recolher Amostra, Medir Temperatura e Análise de Rochas.

Relativamente ao envio de reports da missão do Rover para a Nave-Mãe através do protocolo MissionLink, decidimos que apenas as tarefas Tirar Fotos e Gravar Vídeo enviariam reports para a Nave-Mãe, numa frequência determinada pelo valor do campo Frequência de Updates. Estes reports são imagens, que como já referido anteriormente, seriam divididas em pedaços (frames), pois o tamanho delas é superior ao tamanho máximo do payload do protocolo UDP.

2.2.2. Implementação do Protocolo

O protocolo MissionLink que desenvolvemos, corre sobre UDP, e sendo este o responsável pela comunicação principal entre o Rover e a Nave-Mãe, foi necessário torna-lo robusto e fiável a nível aplicacional, já que o UDP não dá garantias de fiabilidade.

Para a solicitação de uma missão da parte do Rover à Nave-Mãe, começamos por fazer uma adaptação do 3-Way Handshake existente no TCP.

O Rover começa por enviar uma mensagem de tipo ML_SYN para a Nave-Mãe, esperando receber um ML_SYNACK. Caso não receba a resposta dentro de um intervalo de tempo de 2 segundos, a mensagem com o ML_SYN é retransmitida.

No lado da Nave-Mãe, quando é recebido um ML_SYN é então enviado um ML_SYNACK, e é esperado receber como resposta um ML_REQUEST, caso contrário o ML_SYNACK é retransmitido.

Após a receção no Rover do ML_SYNACK, é enviado um ML_REQUEST, que tem como objetivo pedir uma missão concreta à Nave-Mãe. Neste caso, o ML_REQUEST não espera por nenhuma resposta concreta e não é retransmitido automaticamente, desta forma evitamos que a Nave-Mãe passe missões à frente e envie sempre a próxima missão que tem em fila. Se o Rover, após enviar um ML_REQUEST, receber um ML_SYNACK, significa que o ML_REQUEST se perdeu, e é reenviado, mas neste caso não é uma retransmissão automática, pois não existe um intervalo de tempo de espera. Caso após o ML_REQUEST seja recebido um ML_DATA, significa que o ML_REQUEST não se perdeu e no payload deste, está uma missão pronta a ser executada.

Na Nave-Mãe, quando é recebido um ML_REQUEST, é enviado um ML_DATA, que espera como resposta um ML_CONFIRM, caso contrário o ML_DATA é retransmitido. Durante os testes deparamo-nos com uma situação onde devido a atrasos e perdas, o Rover recebia 2 ML_SYNACK, o que gerava 2 ML_REQUEST e por sua vez eram enviadas 2 missões, o que estava errado. Percebendo isto, ajustamos o protocolo para apenas contabilizar o primeiro ML_REQUEST recebido de um Rover.

Depois do Rover receber a missão, ele envia um ML_CONFIRM para confirmar a receção da missão, e espera receber um ML_CONFIRM_ACK, caso contrário o ML_CONFIRM é retransmitido.

Quando a Nave-Mãe recebe o ML_CONFIRM, envia o ML_CONFIRM_ACK.

Apenas quando o Rover recebe o ML_CONFIRM_ACK, é que a missão recebida é executada, pois só assim é garantido que todos os passos foram feitos com sucesso, e é finalmente seguro executar a missão.

O intervalo de tempo de espera para começar as retransmissões é de 2 segundos, para todos os tipos de mensagens.

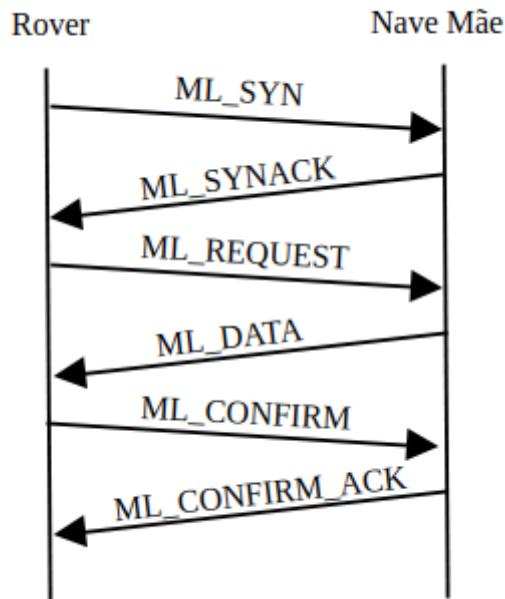


Figura 3: Protocolo MissionLink durante a requisição de uma missão

Durante a execução de uma missão por parte do Rover, no caso da tarefa da missão ser Tirar Fotos ou Gravar Vídeo, é necessário que o Rover envie periodicamente reports relativos à missão através do protocolo MissionLink.

Nesta parte de protocolo toda a comunicação é feita não através da estrutura mensagem referida na Secção 2.1, mas sim através da estrutura report referida na Secção 2.1.1, que contém alguns campos adicionais.

No envio de um report, o Rover começa por enviar uma mensagem de tipo ML_FRAMES, onde os campos do cabeçalho ID de Report e N° de Frames (número de pedaços em que a imagem vai ser dividida) vão preenchidos com os respetivos valores. É esperado que após o envio do ML_FRAMES, seja recebido um ML_OK para confirmar a receção. Caso contrário o ML_FRAMES é retransmitido.

Quando a Nave-Mãe recebe um ML_FRAMES, é criado um coletor de reports, uma espécie de buffer para armazenar as frames da imagem que serão recebidas, e organiza-las por ordem de número de sequência. É depois enviado como resposta um ML_OK.

Após a receção do ML_OK, o Rover divide a respetiva imagem do report nas várias frames, e envia-as todas numa mensagem com tipo ML_REPORT, tendo o payload dessa mensagem os bytes relativos a uma frame. Estas mensagens do tipo ML_REPORT têm o campo N° Sequência do cabeçalho preenchido, que indica que frame da imagem é transportada. No fim do envio de todas as frames da imagem, o Rover envia uma mensagem ML_END para notificar a NaveMãe que já terminou o envio de todas as frames, e que agora ela deve verificar se tem todos os frames que espera, ou não. Este ML_END fica à espera de um ML_MISS ou ML_FIN, caso contrário é retransmitido.

Na Nave-Mãe quando é recebido um ML_REPORT, os bytes da frame recebida, contidos no payload da mensagem, são guardados no coletor de report correspondente

àquele ID de Report. Quando chega um ML_END, a Nave-Mãe verifica o coletor de report do ID de report correspondente. Caso o número de frames recebido seja igual ao esperado, a resposta enviada é do tipo ML_FIN, e a imagem é reconstruída, senão, se faltarem frames, é enviado como resposta um ML_MISS, onde no payload vai um byte array com valores 0 ou 1, onde as posições preenchidas a 1 significam que a frame com número de sequência igual à posição foi recebida, e as posições preenchidas a 0 significam que a frame com número de sequência igual à posição está em falta.

Caso o Rover receba um ML_MISS, ele reenvia as frames que estavam em falta, conforme o byte array que recebe no payload. No fim desse reenvio das frames, reenvia o ML_END para a Nave-Mãe voltar a verificar se está tudo certo. Este processo repete-se até a Nave-Mãe receber a imagem completa.

Caso o Rover receba um ML_FIN, é enviado para a Nave-Mãe um ML_FINACK, e é esperado receber um ML_STOP_CON.

Quando a Nave-Mãe recebe um ML_FINACK, envia de resposta um ML_STOP_CON.

Com receção de um ML_STOP_CON, o Rover percebe que este report foi feito com sucesso, e pode agora fechar a thread responsável pelo envio do report.

Todas as mensagens trocadas entre o Rover e a Nave-Mãe, têm o campo ID de Report do cabeçalho preenchido, para ambas as partes saberem exatamente que report está a ser tratado, e não haver enganos no processamento de vários reports em simultâneo.

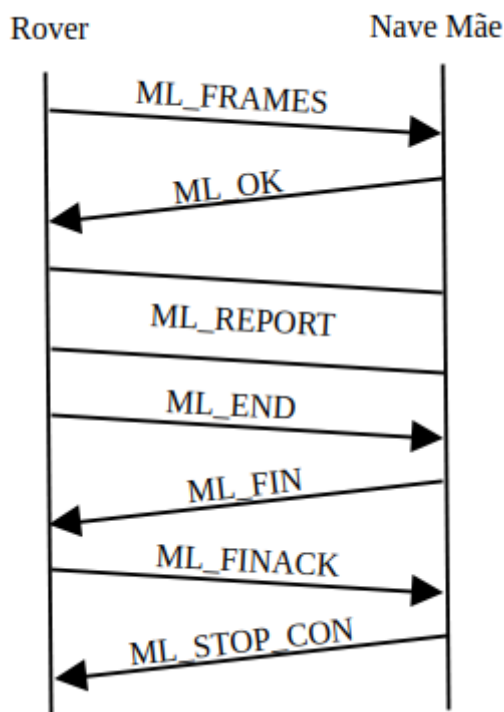


Figura 4: Fluxo do envio de um report sem ML_MISS

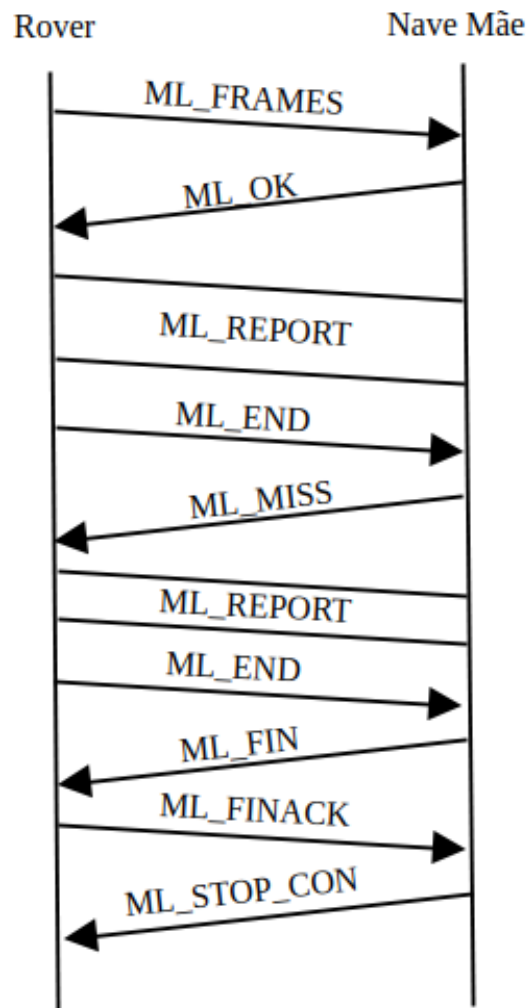


Figura 5: Fluxo do envio de um report quando ocorre ML_MISS

2.3. TelemetryStream

2.3.1. Definição do Estado do Rover (Telemetria)

O estado do Rover (informações de telemetria), foi também um aspeto que definimos previamente. Estas informações do Rover seriam enviadas de forma periódica para a Nave-Mãe, e seriam armazenadas por ela, e teriam a seguinte estrutura:

- Um par de coordenadas (x, y) com a localização atual do Rover
- Estado operacional do Rover
- Valor do nível de bateria atual
- Valor da velocidade atual

Para o estado operacional do Rover são possíveis os seguintes valores: Parado, Espera de Missão, A Caminho, Em Missão e Inoperacional (quando o Rover estiver sem bateria).

2.3.2. Implementação do Protocolo

O protocolo TelemetryStream desenvolvido baseia-se em comunicação via TCP e tem como função o envio constante de dados de monitorização/telemetria de um Rover para a Nave-Mãe, onde a mesma é responsável por receber e processar estas dados em tempo real.

Devido à natureza da telemetria e à fiabilidade do TCP sobre o qual foi implementado o TelemetryStream, decidimos que o envio dos dados de monitorização (estado do Rover) seria feito através de um fluxo unidirecional e constante, onde o Rover, numa frequência fixa de 1 segundo, envia as informações relativas ao seu estado para a Nave-Mãe através de uma mensagem do tipo TS_TCP.

No lado da Nave-Mãe, sempre que é recebido o primeiro TS_TCP de cada Rover, a mesma delega a conexão TCP por onde a mensagem foi enviada para uma nova thread que ficará responsável por receber os dados de telemetria do Rover que iniciou a conexão, garantindo assim a capacidade da Nave-Mãe de suportar o envio de telemetria de múltiplos Rovers em paralelo.

Essa mesma thread, a partir do momento em que é criada, irá continuar a funcionar e a receber mensagens TS_TCP do Rover, enquanto este mantiver o seu funcionamento, onde sempre que a thread recebe uma mensagem, irá atualizar as informações relativas ao estado do Rover armazenadas na Nave-Mãe, substituindo o estado anterior pelo estado mais recente que vai estar encapsulado nessa mesma mensagem.

2.4. API de Observação

A API de Observação implementada no projeto pode ser dividida em duas secções: os endpoints HTTP implementados no lado da Nave-Mãe e a forma como o GroundControl usa esses mesmos endpoints.

2.4.1. Nave-Mãe

A Nave-Mãe expõe uma API de Observação acessível via HTTP, permitindo que outros nós (como o GroundControl) consultem o estado atualizado do sistema: informação sobre rovers, missões e reports. A API segue um modelo simples baseado em endpoints HTTP GET que retornam dados em formato binário, de forma compacta, usando DataInputStream/DataOutputStream. Os endpoints disponibilizados são:

- */rovers* - retorna a lista de IDs dos rovers ativos.
- */rovers/{id}/estado* - retorna o estado de um rover específico (consoante o valor *id* passado).
- */rovers/{id}/missao* - retorna a missão que um rover específico encontra-se a executar (consoante o valor *id* passado).
- */missoes/concluidas* - retorna a lista de todas as missões concluídas.
- */reports* - retorna o Map de reports com os últimos dados enviados pelos rovers.

O protocolo escolhido foi o HTTP por apresentar uma maior facilidade de implementação e permitir consultas simples. Optou-se por um formato binário nas mensagens, garantindo maior eficiência e compactação, o que resulta numa transmissão rápida e adequada ao volume reduzido de dados envolvido na simulação.

A API disponibiliza exclusivamente endpoints HTTP GET, uma vez que o GroundControl obtém informação através de polling periódico. Este modelo foi considerado suficiente para o contexto do projeto, mantendo a implementação simples e coerente com a estrutura inicial do código, que já havia sido construída com esta abordagem em mente.

2.4.2. GroundControl

O GroundControl constitui o nó responsável por acompanhar a execução das várias missões comunicando diretamente com a Nave-Mãe através da API de Observação. Este funciona como cliente da API, consumindo endpoints expostos via HTTP GET e recebendo as respostas em formato binário. A partir destes dados, o GroundControl apresenta ao utilizador informação consolidada e atualizada sobre o estado global da missão usando os seguintes métodos:

- *getListRovers()* → consome */rovers*
- *getEstadoRover(id)* → consome */rovers/{id}/estado*
- *getMissaoRover(id)* → consome */rovers/{id}/missao*
- *getMissoesConcluidas()* → consome */missoes/concluidas*
- *getMapReports()* → consome */reports*

Estes encapsulam o processo de comunicação com a API e processamento dos dados em binário. Depois disto, devolvem os dados que a interface gráfica usará para desenhar as informações obtidas na tela

Desta forma, o GroundControl consegue monitorizar em tempo próximo do real: o estado dos rovers, as missões ativas (o seu progresso) e as missões concluídas, e os últimos reports enviados pelos rovers.

3. Interface do Ground Control

3.1. Interface

A interface do Ground Control que o grupo desenvolveu foi implementada com recurso a uma biblioteca do Java chamada *Swing*. Assim, foi criada uma interface gráfica simples que desenha as informações consoante aquilo que a API de Observação lhe fornece.

Esta pode ser dividida em 4 painéis principais:

- MissõesAtuaisPanel — mostra o progresso das missões que se encontram a ser executadas;

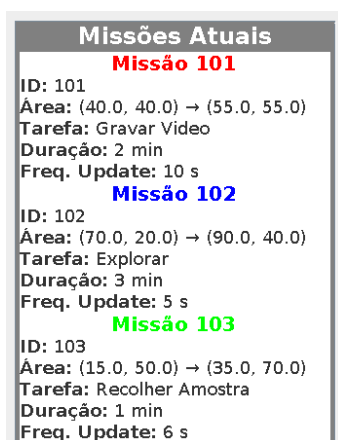


Figura 6: Lista das Missões atuais

- MissõesConcluidasPanel - lista missões já finalizadas;

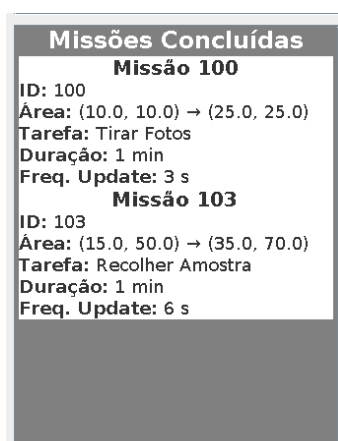


Figura 7: Lista das Missões Concluídas

- RoversPanel - lista os rovers, os seus estados e as suas posições;



Figura 8: Lista dos Rovers ativos

- MapaPanel - representa graficamente a posição dos rovers e respetivamente a área da sua missão (em que a cor da área é igual à cor do rover e da missão que este se encontra a executar);

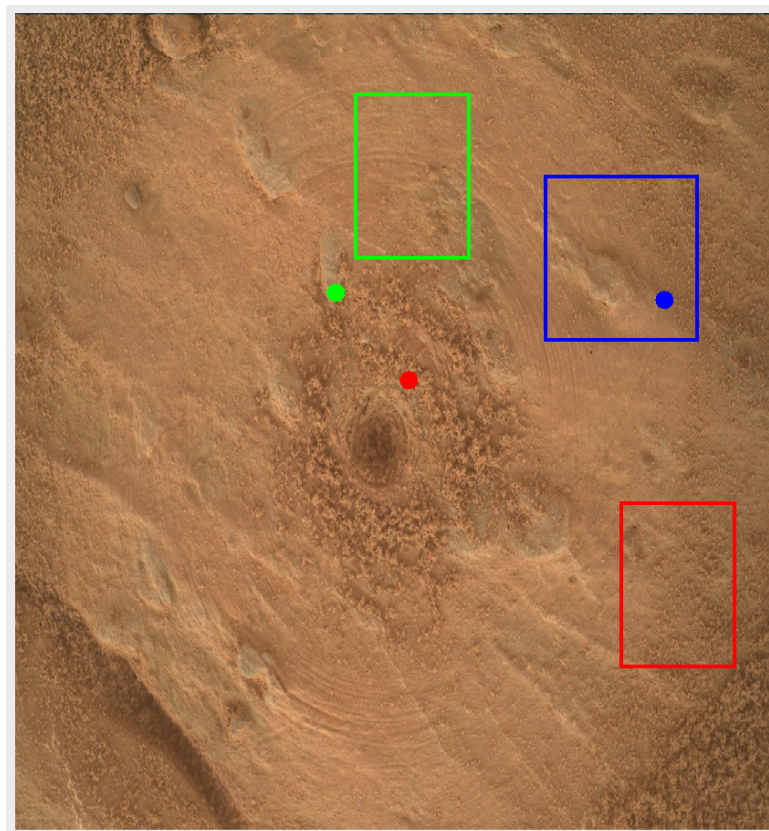


Figura 9: Mapa com os Rovers a executarem as suas Missões

Com estes 4 painéis implementados ficamos com a interface totalmente implementada:

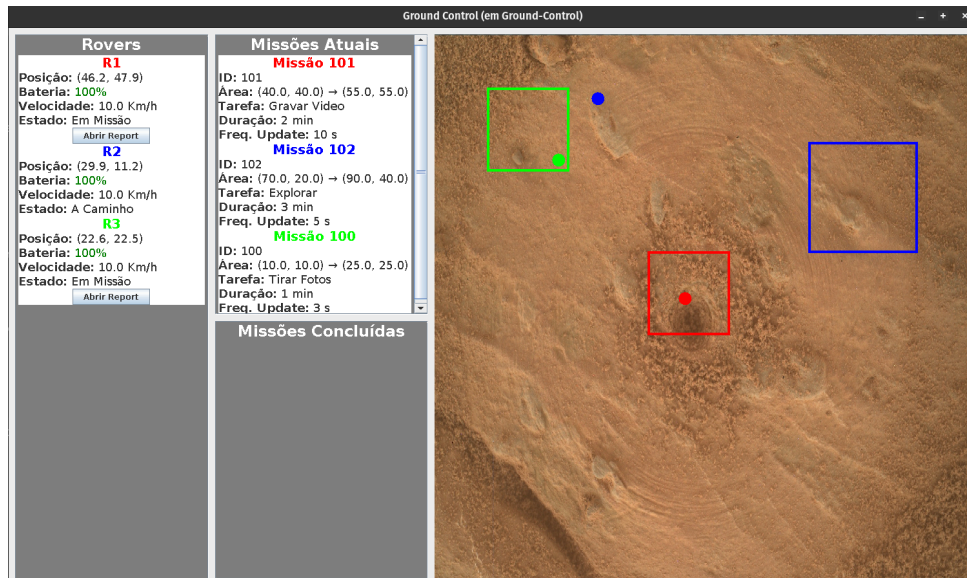


Figura 10: Visão geral da Interface Gráfica

3.2. Execução

Para executar o nó do GroundControl, é apenas necessário que a Nave-Mãe esteja a correr (para que esta disponibilize os endpoints que o GroundControl irá consumir) e executar no terminal deste o comando:

```
make groundcontrol
```

4. Testes na topologia

Para verificarmos e confirmarmos a robustez e fiabilidade dos protocolos desenvolvidos, construímos uma topologia de teste com três níveis de dificuldade. Num primeiro nível não há quaisquer perdas entre os nós da topologia. No nível intermédio existe alguma perda entre os nós da topologia, e no nível mais alto, existem valores de perda elevados, para além de delay entre os nós.

Hard

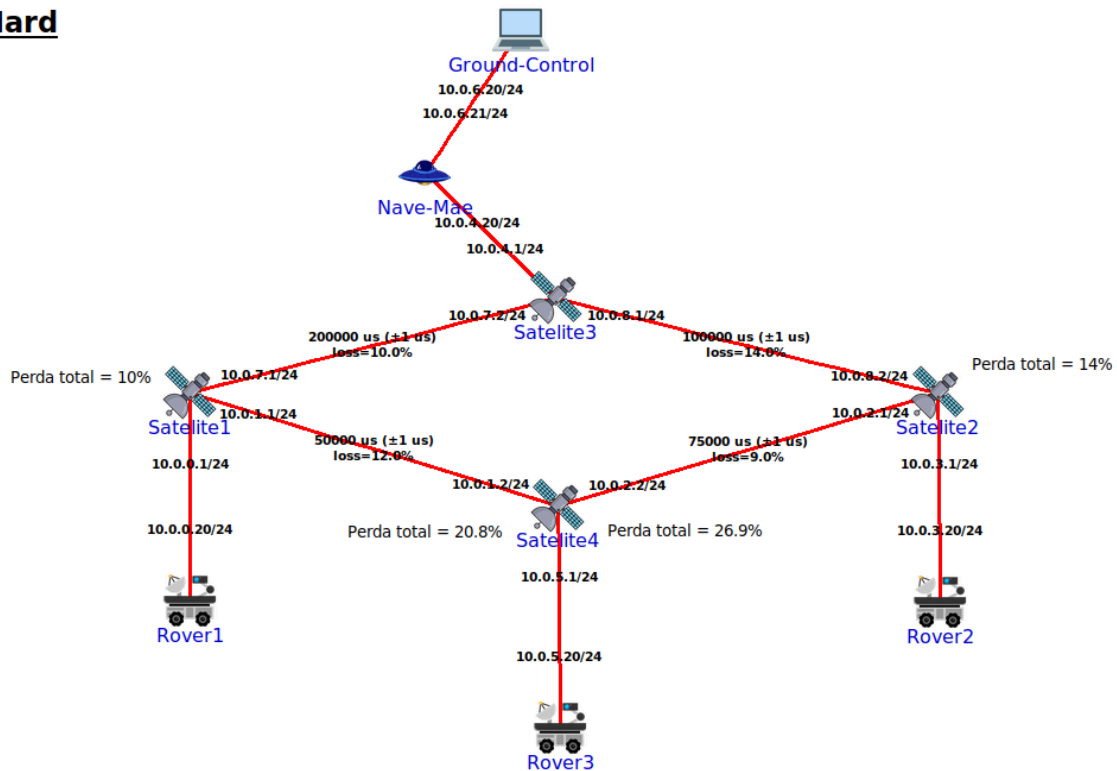


Figura 11: Topologia com o maior nível de perda

Como é expectável, ao testarmos na topologia onde não existem perdas nem delay, não se observam retransmissões de mensagens, nem reenvio de frames de reports relativos ao protocolo MissionLink.

Nas topologias onde já existem perdas entre os nós, verificam-se retransmissões de mensagens e ML_MISS quando as missões passadas aos Rovers requerem envio de reports periódicos. No nível mais difícil da topologia, onde a perda tem valores mais elevados, e existe ainda delay entre os nós, verificam-se retransmissões muito mais frequentemente.

Relativamente à paralelização dos Rovers, a topologia criada mostra perfeitamente que os protocolos implementados permitem que mais do que um Rover possa comunicar com a Nave-Mãe ao mesmo tempo sem causar quaisquer problemas, e sem misturar mensagens e conteúdos de diferentes Rovers. Na topologia apenas colocamos três Rovers, mas da forma que os protocolos foram desenvolvidos é possível termos quantos Rover quisermos a correrem ao mesmo tempo sem a existência de conflitos.

5. Conclusão

Ao longo deste trabalho prático fomos capazes de desenvolver todos os pontos que foram propostos no enunciado, aplicando os conhecimentos que fomos adquirindo ao longo do semestre. Conseguimos implementar o protocolo MissionLink que corre sobre UDP, responsável pela comunicação principal entre os Rovers e a Nave-Mãe, de forma robusta e fiável, conseguindo ultrapassar as fragilidades presentes no UDP.

Conseguimos também aproveitar as vantagens do protocolo TCP para implementar o protocolo TelemetryStream, que é responsável por atualizar a Nave-Mãe com os estados atuais dos Rovers.

Relativamente à API de Observação baseada em HTTP, consideramos que a solução implementada cumpriu plenamente os objetivos definidos. A utilização de endpoints simples com o método HTTP GET revelou-se adequada ao contexto do projeto, permitindo ao Ground Control obter informação atualizada de forma periódica e eficiente.

A decisão do uso de um formato binário para a troca de dados contribuiu para reduzir a sobrecarga das mensagens e melhorar o desempenho da comunicação, sem comprometer a clareza da informação transmitida. Apesar de não ter sido implementado um modelo reativo (como WebSockets ou SSE), o mecanismo de polling adotado mostrou-se suficiente para a monitorização em tempo quase real do sistema.

No que diz respeito à Interface do Ground Control, a utilização da biblioteca Swing permitiu desenvolver uma interface gráfica funcional, clara e intuitiva, capaz de representar visualmente o estado global da missão. A divisão da interface em vários painéis facilitou a organização da informação, permitindo ao utilizador acompanhar simultaneamente o estado dos rovers, as missões ativas, as missões concluídas e a sua representação espacial no mapa. A integração direta com a API de Observação garantiu que os dados apresentados estivessem sempre sincronizados com o estado real do sistema, tornando a interface uma ferramenta eficaz de apoio à monitorização e análise da missão.