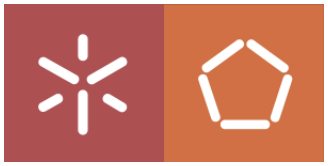

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Comunicações por Computador

Trabalho Prático 2

Ano Letivo de 2024/2025

Network Monitoring System

Flávia Alexandra da Silva Araújo (A96587)
Joshua David Amaral Moreira (A105684)
Miguel Torres Carvalho (A95485)

29 de novembro de 2024

CC

Resumo

No presente relatório, é apresentada a solução desenvolvida para a Unidade Curricular de **Comunicações por Computador**, como parte do projeto final do semestre - ***Network Monitoring System***. Este trabalho teve como objetivo projetar um sistema capaz de monitorizar o tráfego de rede, bem como o desempenho dos dispositivos, entre um servidor centralizado e vários agentes distribuídos, permitindo a execução de tarefas de monitorização, com a respetiva recolha de métricas de desempenho e envio de alertas previamente configurados.

No sistema desenvolvido implementaram-se dois protocolos aplicacionais:
AlertFlow: destinado ao envio de alertas quando certas condições de monitorização definidas são excedidas, utilizando, como protocolo de transporte, o *Transmission Control Protocol* (TCP).
NetTask: implementado para a comunicação de tarefas de monitorização e recolha de métricas, sobre o *User Datagram Protocol* (UDP), este protocolo aplicacional foi desenvolvido de forma a assegurar a comunicação robusta e confiável entre o servidor e os agentes.

O trabalho foi concluído com a implementação e validação das principais funcionalidades do sistema, demonstrando a sua eficácia em ambientes de teste representativos em ambientes virtualizados. Este relatório documenta detalhadamente o *design*, implementação, utilização e resultados obtidos.

Palavras-Chave: *Network Monitoring System*, TCP, UDP, *AlertFlow*, *NetTask*, *python3*, *iperf3*, *ping*, *Comunicações por Computador*.

Índice

Índice de Figuras

Índice de *Snippets*

1 Arquitetura da Solução

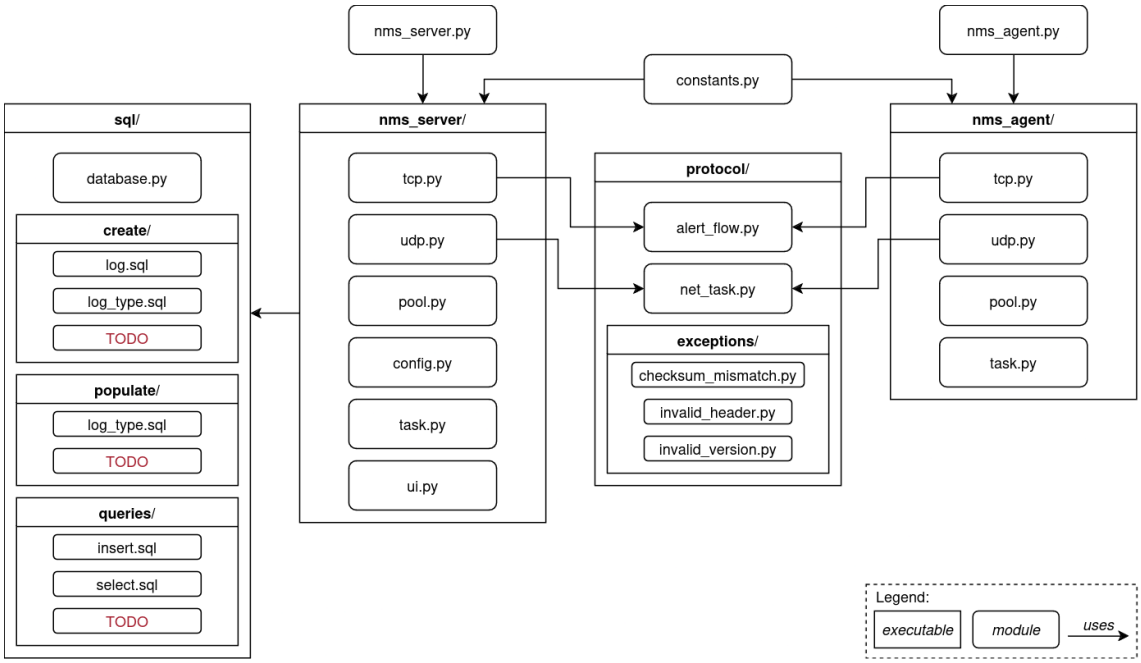

Figura 1.1: Arquitetura da solução *Network Monitoring System*

2 Especificações dos Protocolos Aplicacionais

2.1 *AlertFlow*

2.1.1 Formato de Mensagem

2.1.2 Diagrama de Sequência

- Normal

2.2 *NetTask*

O protocolo *NetTask* é essencial para a funcionalidade harmonizada do ***Network Monitoring System***, sendo este usado para a maioria das comunicações entre o servidor e os agentes, tais como, a primeira conexão de um agente ao servidor, o envio de tarefas pelo servidor, o envio de resultados de tarefas pelos agentes, e a terminação de conexões nos dois sentidos.

Deste modo, o protocolo *NetTask* foi desenvolvido para ser robusto e adaptável a condições adversas de rede, garantindo a entrega fiável e integral de mensagens, sobretudo em rotas deterioradas, com perdas ou duplicação de pacotes, latências elevadas e taxas de débito variáveis.

Para combater tais adversidades, o protocolo aplicacional *NetTask* responsabiliza-se pelas funcionalidades que serão exploradas no seguinte subcapítulo Descrição de Funcionalidades.

Nos próximos subcapítulos, serão detalhadas as especificações do protocolo, nomeadamente o formato do cabeçalho e descrição dos respetivos campos, descrição de funcionalidades e diagramas de sequência que ilustram o comportamento do protocolo em situações normais e adversas.

2.2.1 Formato de Cabeçalho e Descrição de Campos

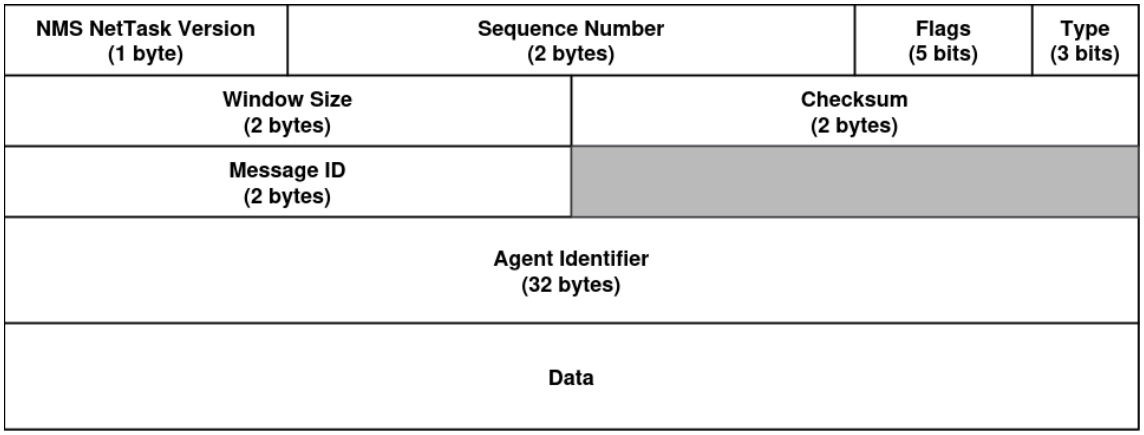

Figura 2.1: Formato do cabeçalho do protocolo *NetTask*

- ***NMS NetTask Version*** (1 *byte*): versão do protocolo, para assegurar a compatibilidade de versões entre o servidor e os agentes;
- ***Sequence Number*** (2 *bytes*): número de sequência da mensagem, para a ordenação de pacotes, deteção de pacotes duplicados e identificação de *acknowled-*

gements;

- ***Flags*** (5 *bits*): *flags* de controlo:

  ACK (1º *bit*): *Acknowledgement*, utilizado para confirmar a receção de pacotes;

  RET (2º *bit*): *Retransmission*, indica que o pacote é uma retransmissão;

  URG (3º *bit*): *Urgent*, indica que a mensagem é urgente;

  WP (4º *bit*): *Window Probe*, utilizado para o controlo de fluxo;

  MF (5º *bit*): *More Fragments*, para (des)fragmentação de pacotes.

- ***Type*** (3 *bits*): tipo da mensagem:

  0 ***Undefined***: mensagem indefinida, utilizada para testes ou quando nenhum tipo de mensagem é aplicável, por exemplo no envio de *window probes*;

  1 ***First Connection***: primeira conexão de um agente ao servidor;

  2 ***Send Tasks***: envio de tarefas pelo servidor;

  3 ***Send Metrics***: envio de resultados de tarefas pelos agentes;

  4 ***EOC (End of Connection)***: terminação de conexões nos dois sentidos;

  ***** ***Reserved***: reservado para futuras extensões. (de 5 a 7);

- ***Window Size*** (2 *bytes*): indica o tamanho da janela de receção, para o controlo de fluxo;

- ***Checksum*** (2 *bytes*): soma de verificação da mensagem, para a deteção de erros;

- ***Message Identifier*** (2 *bytes*): identificador da mensagem, utilizado para a desfragmentação e ordenação de pacotes;

- ***Agent Identifier*** (32 *bytes*): identificador do agente, podendo este ser recetor ou emissor da mensagem;

- ***Data/Payload*** (n *bytes*): carga útil da mensagem com tamanho variável, contendo a informação a ser transmitida nas mensagens do tipo *Send Tasks* e *Send Metrics*.

2.2.2 Descrição de Funcionalidades

Retransmissão de Pacotes Perdidos

A retransmissão de pacotes perdidos é efetuada quando o emissor não recebe um *acknowledgement* de um pacote enviado, após um determinado intervalo de tempo. Para a concretização desta funcionalidade, o emissor guarda todos os pacotes enviados em memória, reenviando os pacotes não confirmados após o intervalo de tempo definido no fichero `constants.py` sobre a nomenclatura `RETRANSMIT_SLEEP_TIME`, os pacotes mantém o cabeçalho original, sendo apenas ativada a *flag RET*.

(Des)Fragmentação e Ordenação de Pacotes

A fragmentação de pacotes ocorre quando a carga útil da mensagem excede o tamanho máximo permitido, sendo este tamanho predefinido para 1500 *bytes*, no ficheiro `constants.py` sobre a nomenclatura `BUFFER_SIZE`, uma vez que este é o valor máximo para o *Maximum Transmission Unit* (MTU).

Na fragmentação de pacotes os dados da mensagem (campo *Data*) é dividido em fragmentos, de forma a que, com adição do cabeçalho, o tamanho total do pacote não exceda o tamanho máximo permitido. Os primeiros fragmentos são marcados com a *flag MF*, com exceção do último fragmento, indicando que não existem mais fragmentos a serem enviados. Para a identificação dos fragmentos, é atribuído um *Message Identifier* único, sendo este igual ao número de sequência do primeiro fragmento. Os números de sequência dos fragmentos são incrementados de forma sequencial.

Na desfragmentação, o recetor mantém em memória os fragmentos recebidos, cada pacote recebido é guardado em um *array* caso seja fragmentado. Para identificar se um pacote é ou não um fragmento, é verificado se a *flag MF* está desativada e se o *Message Identifier* é igual ao número de sequência do pacote. Se tal não se verificar, o pacote é guardado em memória até que todos os fragmentos sejam recebidos, ou seja, devem existir todos os fragmentos tal que o seu número de sequência esteja entre os seus *Message Identifiers* e o número de sequência do último fragmento, este procedimento garante que a mensagem é desfragmentada apenas quando todos os fragmentos são recebidos, uma vez que a ordem de chegada destes não é garantida.

Uma vez que todos os fragmentos são recebidos, estes são ordenados pelo número de sequência e os dados da mensagem são concatenados para formar a mensagem original.

O campo *Message Identifier* permite que fragmentos de diferentes mensagens sejam distinguidos, garantindo a integridade dos pacotes recebidos.

Deteção e Manuseamento de Pacotes Duplicados

A deteção e manuseamento de pacotes duplicados, tal como na (des)fragmentação e ordenação de pacotes, recorre ao *Sequence Number* da mensagem. O recetor mantém um registo dos números de sequência pacotes recebidos para a deteção de pacotes duplicados, escartando pacotes que já tenham sido recebidos anteriormente, e reenviando um *acknowledgement* ao emissor para confirmar a receção do pacote de forma a evitar retransmissões desnecessárias.

NOTA TODO ainda só fazemos esta deteção para pacotes fragmentados!

Deteção de Erros

A deteção de erros é efetuada através da soma de verificação da mensagem (*Checksum*), calculada com base no cabeçalho e na carga útil da mensagem. O recetor, ao receber um pacote, calcula a soma de verificação e compara-a com a recebida. Se a soma de verificação calculada for diferente da recebida, o pacote é descartado, e o emissor uma vez que não recebe um *acknowledgement* reenviará o pacote.

Conpatibilidade de Versões

A compatibilidade de versões é garantida através do campo *NMS NetTask Version* do cabeçalho da mensagem, permitindo a identificação da versão do protocolo entre o servidor e os agentes. Se a versão do protocolo do emissor for diferente da versão do recetor, a mensagem é descartada, evitando possíveis incompatibilidades e erros.

Controlo de Fluxo

O controlo de fluxo é efetuado através do tamanho da janela de receção (*Window Size*), indicando ao emissor o número de pacotes que o recetor pode receber sem congestionar a ligação. O valor inicial da janela de receção é definido no ficheiro `constants.py` sobre a nomenclatura `INITIAL_WINDOW_SIZE`, sendo este valor de 32 pacotes. O emissor, ao enviar um pacote, ...

TODO: neste momento usamos o window size no array de pacotes a ordenar/desfragmentar, não deverá ser nos pacotes por receber acks? ou ambos?

Neste descrição de funcionalidades, não falamos que o servidor guarda por exemplo, pacotes por ordenar/desfragmentar, num dicionário para cada agente, devemos referir?

2.2.3 Diagramas de Sequência

- First Connection
- End of Connection

- Retransmission
- Desfragmentação e Ordenação de pacotes
- Deteção e Manuseamento de Pacotes Duplicados
- Deteção de Erros
- Controlo de Fluxo

3 Implementação

3.1 Parâmetros dos Executáveis

3.1.1 *nms_server.py*

```
$ ./nms_server.py --help
usage: nms_server.py [-h] [-c CONFIG] [-v]

Network Management System Server

options:
  -h, --help         show this help message and exit
  -c CONFIG, --config CONFIG
                     Configuration file path
  -v, --verbose      Enable verbose output
```

Snippet 3.1: Parâmetros do executável *nms_server.py*

```
$ ./nms_agent.py --help
usage: nms_agent.py [-h] [-s SERVER] [-v]

Network Management System Agent

options:
  -h, --help         show this help message and exit
  -s SERVER, --server SERVER
                     Server IP
  -v, --verbose      Enable verbose output
```

Snippet 3.2: Parâmetros do executável *nms_agent.py*

3.2 Ficheiro de Configuração

3.3 Bibliotecas Utilizadas

Dúvida: deve-se referir todas as bibliotecas utilizadas no desenvolvimento?
por exemplo: *socket*, *os*, *sys*, *threading*, etc Ou apenas as bibliotecas mais relevantes para a solução? *ping3*, *iperf*, etc

3.4 Detalhes Técnicos? Maybe

4 Testes e Resultados

As demonstrações para situcoes normais e adversas, contam para este capítulo? Ou teremos de fazer testes mais específicos?

Para além da testagem manual contínua ao longo do desenvolvimento, foram desenvolvidos testes unitários para as classes e funções mais críticas do sistema. Estes testes foram desenvolvidos com recurso à biblioteca *unittest* do Python.

5 Conclusões e Trabalho Futuro

5.1 Conclusões

A realização deste projeto permitiu a aplicação prática dos conhecimentos adquiridos durante o semestre na Unidade Curricular de **Comunicações por Computador**, nomeadamente no que diz respeito à conceção e implementação de protocolos aplicacionais para a comunicação entre sistemas.

A implementação do ***Network Monitoring System*** permitiu a integração de vários conceitos e técnicas de comunicação, como a fragmentação de pacotes, retransmissão de pacotes perdidos, controlo de fluxo, deteção de erros e ordenação de pacotes, garantindo a fiabilidade e robustez do sistema.

A utilização de protocolos aplicacionais personalizados, como o *AlertFlow* e o *NetTask*, permitiu a implementação de funcionalidades específicas para a monitorização e análise de tráfego de rede, adaptadas às necessidades do sistema.

Por conseguinte, o sistema desenvolvido demonstrou ser eficaz na monitorização e análise de tráfego de rede, permitindo a recolha de métricas de desempenho e a execução de tarefas de monitorização de forma eficiente e fiável.

5.2 Trabalho Futuro

TODO Encriptação de mensagens Gerar análises gráficas dos resultados

Referências

[1] Connolly, T., & Begg, C. (2015). Database Systems: Example (6th ed.). Pearson Education. London, UK.

Anexos

[I] Exemplo de Anexo