

**RELATÓRIO SOBRE O TRABALHO DE CAMADA DE TRANSPORTE**

TRABALHO DA M2

**Estudantes:** André Aliardi, Eduardo Amaro Maciel, Jhonatan Mariani, Jhonatan Fernandes e João Vitor Dubiela

**Disciplina:** Redes de Computadores 1

**Professor:** Felipe Viel

**Data:** 20/10/2021

**ÍNDICE**

[**1.** **Descrição do trabalho proposto** 3](#_Toc85404612)

[**2.** **Descrição do protocolo implementado** 4](#_Toc85404613)

[**2.1.** **O que é o RDT?** 4](#_Toc85404614)

[**2.2.** **Versões existentes do RDT** 4](#_Toc85404615)

[**2.2.1.** **Versão 1.0** 4](#_Toc85404616)

[**2.2.2.** **Versão 2.0** 4](#_Toc85404617)

[**2.2.3.** **Versão 2.1** 5](#_Toc85404618)

[**2.2.4.** **Versão 2.2** 5](#_Toc85404619)

[**2.2.5.** **Versão 3.0** 5](#_Toc85404620)

[**3.** **Códigos importantes da implementação** 7](#_Toc85404621)

[**4.** **Resultados obtidos com a implementação** 7](#_Toc85404622)

[**5.** **Resultados obtidos com as simulações** 7](#_Toc85404623)

[**6.** **Análise e discussão sobre os resultados** 7](#_Toc85404624)

1. Descrição do trabalho proposto
2. Descrição do protocolo implementado

2.1. O que é o RDT?

2.2. Versões existentes do RDT

1. Códigos importantes da implementação.
2. Resultados obtidos com a implementação (tabelas, gráficos, etc.).
3. Resultados obtidos com as simulações.
4. Análise e discussão sobre os resultados.
5. **Descrição do trabalho proposto**

Nesse trabalho deve-se implementar em 2 partes as versões propostas do protocolo Reliable Data Transfer (RDP). Sendo, a primeira parte sobre a versão 2.1 e 2.2. Já na segunda parte deverá ser implementada a versão 3.0 do protocolo.

Para a primeira parte da implementação do algoritmo, ele deve englobar os seguintes requisitos:

* Seqnum – sequência do pacote enviado
* Acknum – número do Ack gerado
* Checksum – valor da soma de verificação calculado
* Carga útil

Para a segunda parte deve-se utilizar, ou seja, reaproveitar a parte 1 já implementada anteriormente, adicionando o temporizador para o controle do tempo para envio de resposta do pacote.

Para a simulação da implementação realizada, deve-se realizar simulações que contenham uma transmissão normal do pacote, uma transmissão do pacote com atraso, uma transmissão do pacote com perda e uma transmissão do pacote com corrompimento dos dados.

1. **Descrição do protocolo implementado**

## **O que é o RDT?**

O RDT (Reliable Data Transfer) que em português significa transferência confiável de dados. Ele foi criado advindo do protocolo UDT que é um protocolo da camada de transporte, muito utilizado em aplicações que exigem um transporte rápido e contínuo de dados entre os equipamentos, exemplos de tipo de aplicações que utilizam são streaming de áudio e vídeo.

Com isso, não há verificação do recebimento e a integridade dos dados enviados, sendo os dados transmitidos apenas uma vez. Quando os pacotes chegam corrompidos, eles são descartados, sem que o emissor fique sabendo do problema.

Com a utilização do RTD, nenhum dos dados transferidos são corrompidos ou perdidos, e todos são entregues na ordem em que foram enviados. Portanto, um protocolo de transferência confiável de dados possuir essa implementação de abstração de serviço.

* 1. **Versões existentes do RDT**

### **Versão 1.0**

Na versão mais simples do RDT, tem-se a implementação onde o canal de transferência é completamente confiável. Com isso, temos uma transmissão unidirecional, sem a confirmação e entrega dos dados ao remetente.

### **Versão 2.0**

Na versão 2.0 do RDT, temos a verificação de bits do pacote verificando se esses estão ou não corrompidos. Esses erros podem ocorrer nos componentes físicos da rede enquanto está sendo transmitido o pacote. Embora, ainda nessa versão do protocolo é feita uma suposição de que todos os pacotes transmitidos sejam recebidos pelo destinatário, na ordem que foram enviados pelo remetente.

Portanto, para verificar se há presença de erros são exigidas 2 ferramentas:

* **Detecção de erros:**

O destinatário deve detectar se há erros de bits nos pacotes recebidos. Para isso, é utilizado técnicas onde o remetente envie bits extras, além dos bits do dado original para que o destinatário possa detectar se há erros.

* **Realimentação do destinatário:**

Envio de feedback ao remetente, assim, o remetente saberá se um pacote chegou ao seu destino. Para isso, são utilizados NAKs e ACKs são exemplos dessas respostas.

O remetente nessa versão (2.0) possui dois estados:

1. **Espera a chamada de cima:**

Nesse estado o protocolo está esperando que os dados sejam passados pela camada superior. Quando haver um evento de envio de dados, o remetente criará um pacote contendo os dados que serão enviados juntamente com a soma de verificação do pacote.

1. **Espera o ACK ou NAK:**

Nesse estado o protocolo fica esperando um pacote ACK ou NAK do destinatário. Caso chegue um pacote ACK, o remetente saberá que o pacote mais recente chegou ao destinatário. Se um NAK for recebido, o protocolo retransmitirá o último pacote e esperará por uma nova resposta do destinatário. Enquanto o remetente está esperando por uma resposta, ele não pode receber mais dados da camada superior.

Já o destinatário nessa versão 2.0 do RDT tem apenas um estado, sendo muito parecido com o RDT na versão 1.0. Quando ele recebe um pacote, ele responde ao remetente um pacote com um ACK ou NAK dependendo do estado do pacote recebido. Mesmo funcionando nessa versão 2.0 essa resposta, ainda possui defeitos, sendo um deles a possibilidade de haver uma corrupção dos pacotes NAK e ACK. Para isso, uma solução viável a numeração pelo remetente dos pacotes enviados, sendo essa numeração sequencial.

### **Versão 2.1**

Na versão 2.1 do RDT há a implementação contra falha de recebimento dos pacotes, ainda mesmo que com a falha citada anteriormente. Por isso, quando um pacote fora de ordem é recebido pelo destinatário é enviado ao remetente um ACK, já quando um pacote recebido esteja corrompido um NAK é enviado. Um remetente que recebe dois ACKs para o mesmo pacote, interpreta que o destinatário não recebeu corretamente o pacote seguido.

### **Versão 2.2**

Na versão 2.2 do RDT há uma melhoria da versão 2.1, que é a implementação da numeração em sequência do pacote reconhecido pelo destinatário por uma mensagem ACK e o remetente verifica esse número que está sendo reconhecido também por uma mensagem ACK.

### **Versão 3.0**

Na versão 3.0 do RDT, possui um temporizador de contagem regressiva para que o remetente fique esperando um ACK sem saber se ele foi perdido no meio do caminho, se está atrasado ou se o pacote nem chegou ao destino.

Então o remetente terá que adotar um tempo intermediário de recebimento do pacote ACK dentro desse período, caso contrário fará o reenvio do pacote para o destinatário. Mas como o destinatário controla os pacotes recebidos quando há um atraso no envio? Através da numeração deles, consegue-se ter esse controle.

O remetente após o tempo definido e com o não recebimento do ACK, não se sabe se o pacote teve atraso na entrega, se o mesmo foi perdido, se o ACK teve atraso na entrega ao remetente ou ainda se o ACK no meio do caminho (destinatário-remetente) foi perdido. Nesse caso é feito o reenvio do pacote para o destinatário.

1. **Códigos importantes da implementação**

Aisjiajsiajsjaisjaiojsoiajsiajsaosjaisoajsoijaisjaiosjoiajsiajsiwqjçuodqhdançudhqliduwhaluwdhawliduwalidhwadliawudbaiydwaldbwayodiwludabdlybawd9lwaubdyoawbgdawudbwyaodgwabydoyawdgwagd8oawdga7wgd78owgadgao8d

1. **Resultados obtidos com a implementação**

Aisjiajsiajsjaisjaiojsoiajsiajsaosjaisoajsoijaisjaiosjoiajsiajsiwqjçuodqhdançudhqliduwhaluwdhawliduwalidhwadliawudbaiydwaldbwayodiwludabdlybawd9lwaubdyoawbgdawudbwyaodgwabydoyawdgwagd8oawdga7wgd78owgadgao8d

1. **Resultados obtidos com as simulações**

Aisjiajsiajsjaisjaiojsoiajsiajsaosjaisoajsoijaisjaiosjoiajsiajsiwqjçuodqhdançudhqliduwhaluwdhawliduwalidhwadliawudbaiydwaldbwayodiwludabdlybawd9lwaubdyoawbgdawudbwyaodgwabydoyawdgwagd8oawdga7wgd78owgadgao8d

1. **Análise e discussão sobre os resultados**

Aisjiajsiajsjaisjaiojsoiajsiajsaosjaisoajsoijaisjaiosjoiajsiajsiwqjçuodqhdançudhqliduwhaluwdhawliduwalidhwadliawudbaiydwaldbwayodiwludabdlybawd9lwaubdyoawbgdawudbwyaodgwabydoyawdgwagd8oawdga7wgd78owgadgao8d