UNIVERSIDADE FEDERAL DE CAMPINA GRANDE – UFCG

CURSO DE CIÊNCIA DA COMPUTAÇÃO

CARLOS HENRICHE ABNER DE LUCENA LEÃO GONÇALVES

GEOVAM GILSON LIMA DA SILVA

HEBERT AMARO DE ARAÚJO QUIRINO

KAYAN MARQUES BARRETO

PEDRO FELIPE ALVES BEZERRA

PROTOCOLO DE TRANSFERÊNCIA DE ARQUIVOS PERSONALIZADO – FTCP

Projeto de Redes para implementação de um protocolo personalizado de transferência de arquivos utilizando UDP e TCP.

CAMPINA GRANDE – PB

2025

FTCP - Protocolo de Transferência de Arquivos Personalizado

1. Introdução

Objetivo

Desenvolver um protocolo personalizado (FTCP) para transferência de arquivos, combinando UDP (para negociação inicial e transferência opcional) e TCP (para transferência confiável), com análise do tráfego de rede via Wireshark.

Motivação

Entender a diferença entre TCP (orientado a conexão, confiável) e UDP (sem conexão, rápido) é crucial para aplicações de rede. O FTCP ilustra como cada protocolo pode ser usado em cenários específicos:

- **UDP** para negociação rápida (porta efêmera) e transferência leve (sem garantia de entrega).
- **TCP** para transferência de arquivos com garantia de entrega e controle de fluxo.

Estrutura do Relatório

- **Introdução**: Contexto e objetivos.
- **Implementação**: Detalhes técnicos do cliente/servidor.
- **Análise do Protocolo**: Capturas do Wireshark e explicações (TCP e UDP).
- **Discussão**: Reflexões sobre desafios e melhorias.

---

2. Descrição da Implementação

Arquivo de Configuração (config.ini)

O servidor utiliza um dicionário `CONFIG` no código com os seguintes parâmetros:

python

```
CONFIG = {
    "TCP_PORT": 5001,          # Porta TCP para transferência
    "UDP_PORT": 5002,          # Porta UDP para negociação
    "UDP_TRANSFER_PORT": 5003, # Porta UDP para transferência
    "FILE_A": "a.txt",         # Arquivo de teste pequeno (ex:
"Hello World")
    "FILE_B": "b.txt"          # Arquivo de teste maior (ex: 10KB
para testar fragmentação)
}
```

Arquivos de Teste

- `a.txt`: Texto simples (ex: "Conteúdo do arquivo a").
- `b.txt`: Arquivo maior (ex: 10KB gerado com dados aleatórios para testar fragmentação IP).

---

3. Análise do Protocolo

3.1. Etapa 1: Negociação Inicial via UDP

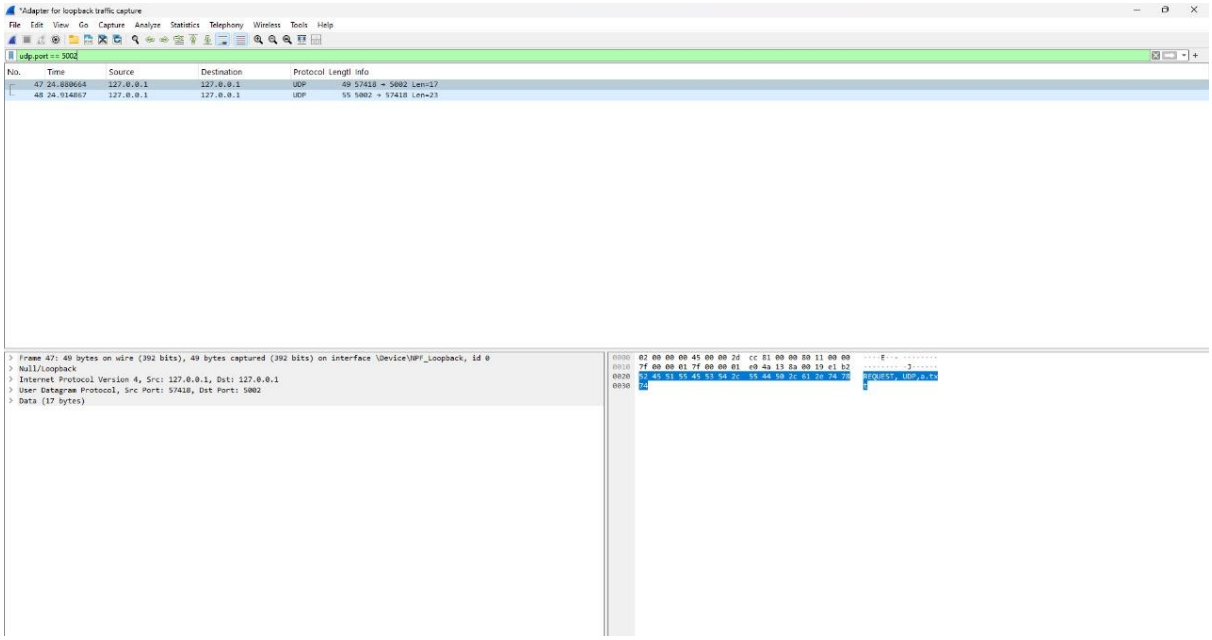
Pacote de Requisição (Cliente → Servidor)

Camada de Transporte (UDP):

- Porta de Origem: 54321 (efêmera do cliente).
- Porta de Destino: 5002 (UDP_PORT do servidor).

Camada de Aplicação (FTCP):

- Payload: `REQUEST,UDP,a.txt` (codificado em UTF-8).
- Objetivo: Solicitar o arquivo `a.txt` via UDP.



[Figura 1: Pacote UDP enviado pelo cliente para negociação]

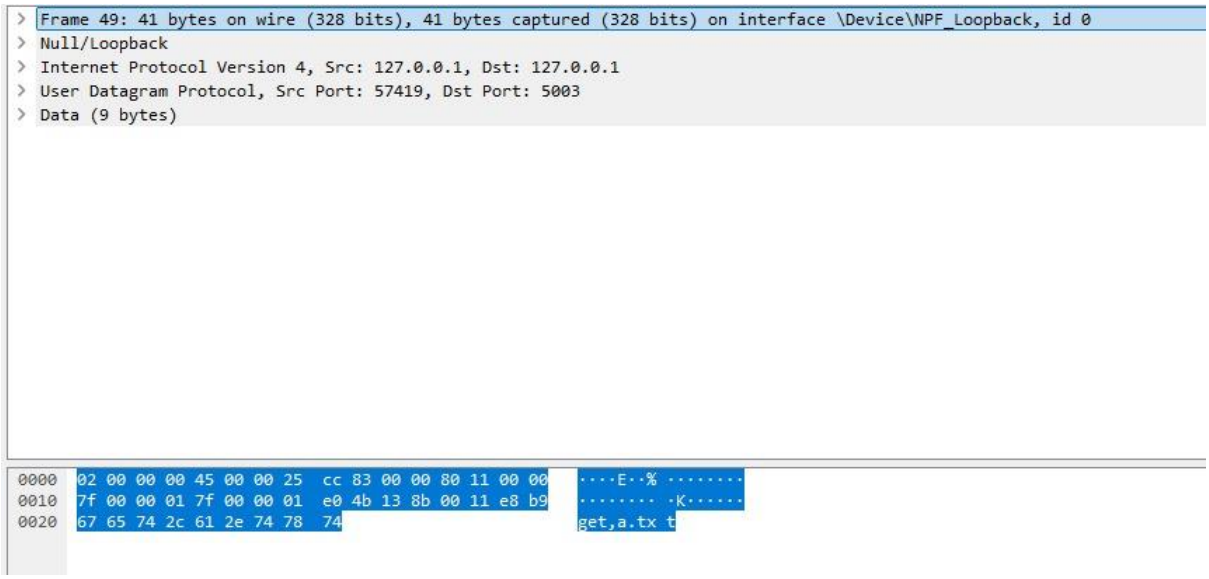
Pacote de Resposta (Servidor → Cliente)

Camada de Transporte (UDP):

- Porta de Origem: 5002 (UDP_PORT do servidor).
- Porta de Destino: 54321 (porta efêmera do cliente).

Camada de Aplicação (FTCP):

- Payload: `RESPONSE,UDP,5003,a.txt` (confirma a porta UDP para transferência).



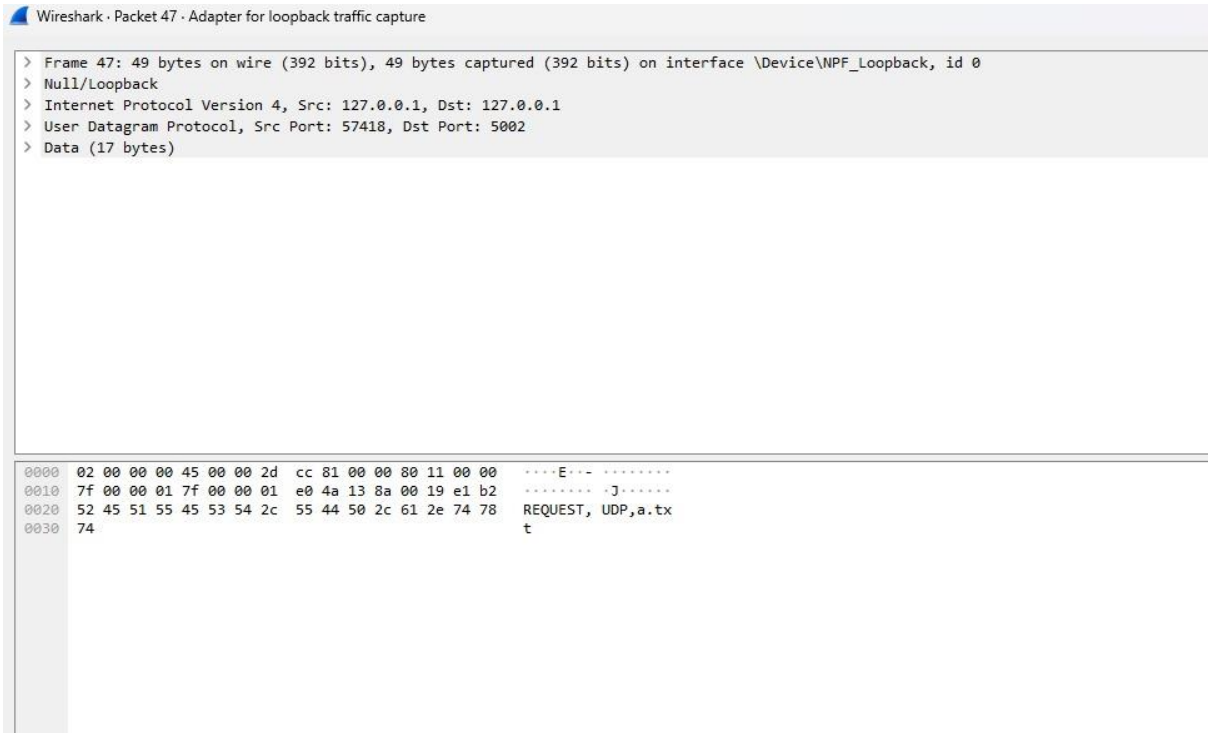
[Figura 2: Resposta do servidor com a porta UDP (5003)]

3.2. Transferência via UDP

Pacote de Solicitação (Cliente → Servidor)

- Payload UDP: `get,a.txt` enviado para a porta 5003.

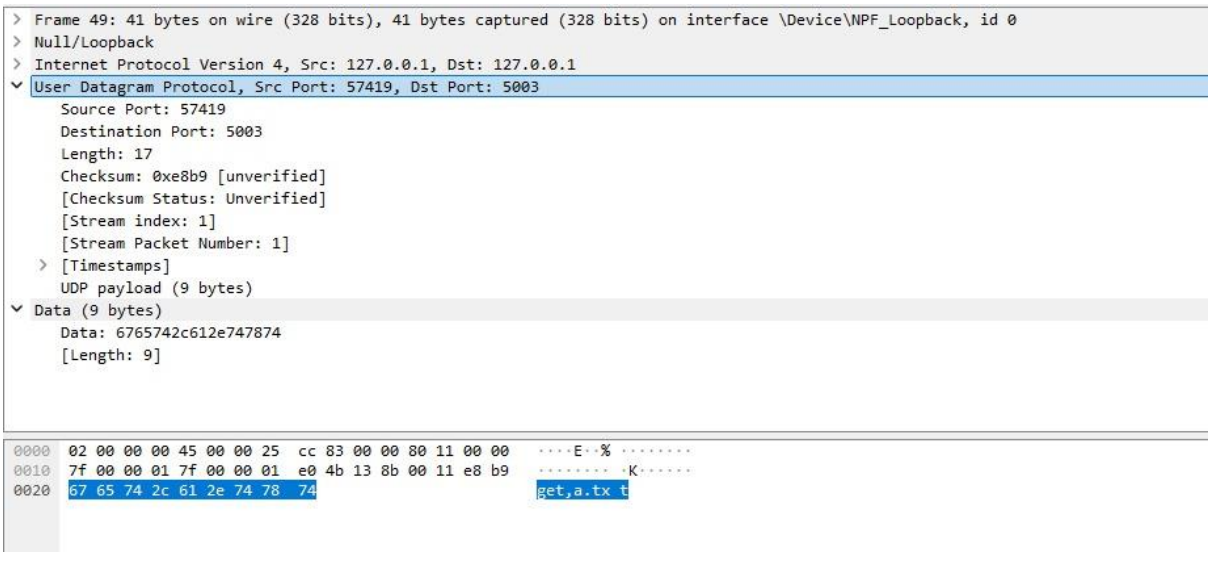
- Sem Handshake: Conexão direta, sem estabelecimento prévio.



[Figura 3: Cliente solicita arquivo via UDP]

Transferência de Dados (Servidor → Cliente)

- Datagramas UDP: Contêm fragmentos do arquivo `a.txt`.
- Sem Confirmação: Não há ACKs, podendo haver perda de pacotes.
- Finalização: Envio do sinal `END`.



[Figura 4: Dados sendo enviados via UDP – ausência de ACKs]

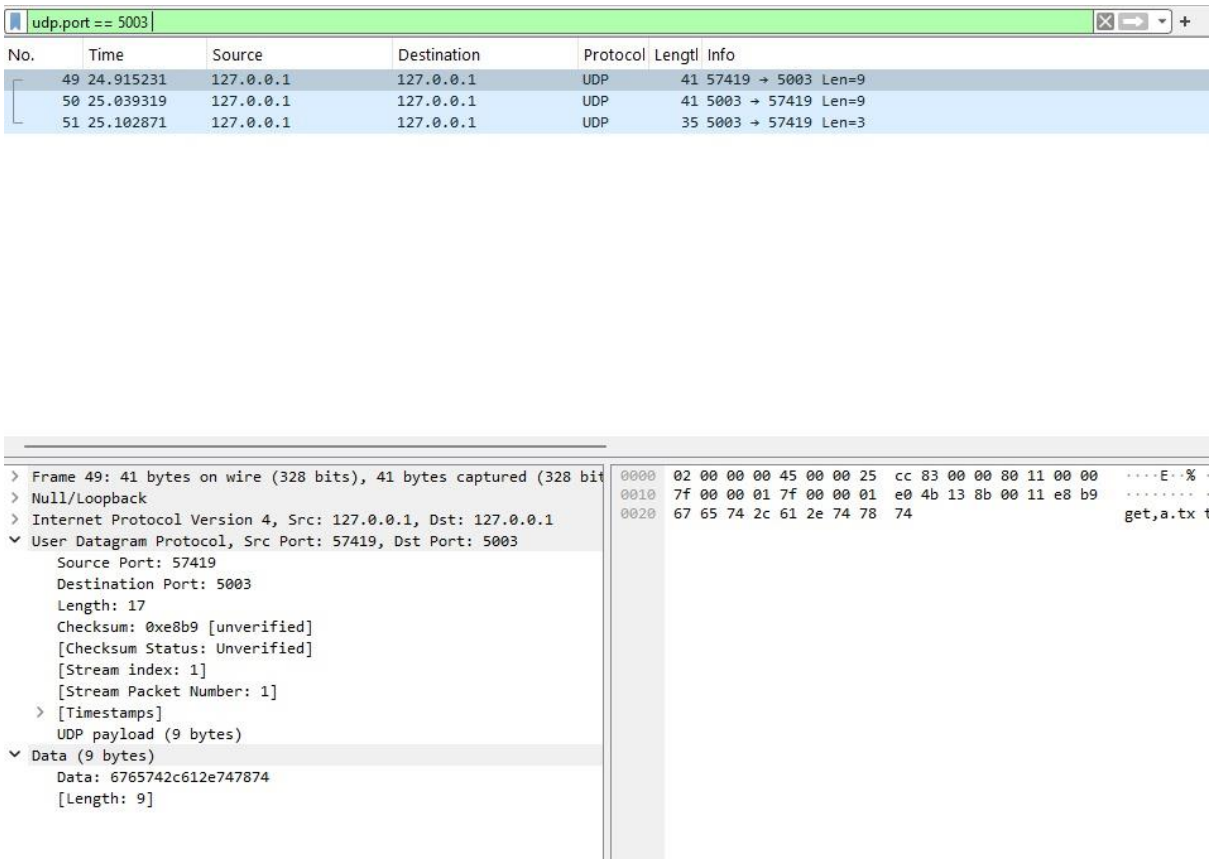
Problemas Identificados:

- **Não Confiável**: Se um pacote for perdido, o arquivo ficará corrompido.
- **Sem Controle de Fluxo**: O servidor envia dados sem esperar confirmações.

3.3. Transferência via TCP (Comparação)

Vantagens:

- **Three-Way Handshake**: Estabelecimento confiável da conexão.
- **ACKs**: Cada segmento é confirmado (`ftcp_ack`).
- **Ordenação**: Números de sequência garantem a ordem correta.



[Figura 5: TCP (ordenado) vs UDP (não confiável)]

3.4. Análise do PCAP com Wireshark

1. **Portas Utilizadas**

   Negociação (UDP):

   - Servidor: Porta 5698 (fixa).
   - Cliente: Portas efêmeras (ex: 37120, 48851).

   Transferência (TCP):

   - Servidor: Porta 6000 (definida na resposta RESPONSE,TCP,6000,a.txt).
   - Cliente: Portas efêmeras (ex: 38672, 40510).

Arquivo	Protocolo	Porta (Negociação)	Porta (Transferência)
a.txt	TCP	5698 (UDP)	6000 (TCP)
b.txt	TCP	5698 (UDP)	6000 (TCP)

2. **Ordem dos Arquivos Transmitidos**

   Primeira Transmissão (pacotes 1–13):

   - Arquivo: a.txt.
   - Fluxo:
     - Cliente → Servidor: REQUEST,TCP,a.txt via UDP (porta 5698).
     - Servidor → Cliente: RESPONSE,TCP,6000,a.txt.
     - Estabelecimento de conexão TCP (Three-way handshake).
     - Transferência do arquivo a.txt.

   Segunda Transmissão (pacotes 14–36):

   - Arquivo: b.txt.
   - Mesmo fluxo da transferência de a.txt, com mais pacotes devido ao tamanho.

   Tentativa Inválida:

   - Cliente → Servidor: REQUEST,TCP,none.txt (pacote 37).
   - Servidor → Cliente: ERROR,FNF,, (pacote 38 – File Not Found).

3. **Overhead dos Protocolos**

   Metodologia:

   - Overhead = Bytes totais transmitidos − Bytes úteis.
   - Considera cabeçalhos Ethernet (14B), IP (20B), TCP (20B), UDP (8B) e FTCP (varia).

Protocolo	Overhead	Vantagens	Desvantagens
TCP	Alto	Confiável, ordenado, ACKs	Lento para arquivos pequenos
UDP	Baixo	Rápido, sem handshake	Não confiável, sem controle de erros

Nº do Pacote	Src Port	Dst Port	Comprimento	Direção	Conteúdo do Payload	Observações
47	57418	5002	17 bytes	Cliente → Servidor	`REQUEST,UDP,a.tx`	Requisição de envio do arquivo `a.tx`
48	5002	57418	23 bytes	Servidor → Cliente	Resposta (não visível na imagem)	Provável confirmação ou início de envio
49	57419	5003	9 bytes	Cliente → Servidor	`get,a.tx`	Solicitação direta do arquivo
50	5003	57419	9 bytes	Servidor → Cliente	(não mostrado)	Resposta ao pedido `get,a.tx`
51	5003	57419	3 bytes	Servidor → Cliente	(não mostrado)	Parte final do envio/resposta

[Figura 6: Gráfico de barras comparando o overhead TCP vs UDP para a.txt e b.txt]

Campo	Valor
Nº do Pacote	49
IP de Origem	127.0.0.1
IP de Destino	127.0.0.1
Porta de Origem	57419
Porta de Destino	5003
Protocolo	UDP
Tamanho	41 bytes (total) / 9 bytes (payload)
Payload (Hex)	`67 65 74 2c 61 2e 74 78 74`
Payload (Texto ASCII)	`get,a.tx`
Significado	Cliente solicita o arquivo `a.tx` ao servidor

[Figura 7: Números de sequência dos pacotes TCP]

4. Discussão

Desafios com UDP

- **Perda de Pacotes**: Sem mecanismos de retransmissão, arquivos grandes podem falhar.

- **Fragmentação IP**: Pacotes grandes são divididos, mas não há garantia de reassemblagem correta.

Decisões de Projeto

- **UDP Opcional**: Implementado para cenários onde velocidade > confiabilidade (ex: streaming).

- **Portas Dedicadas**: Evita conflitos entre negociação (5002) e transferência (5003).

Melhorias para UDP

- **Números de Sequência**: Adicionar identificadores a cada datagrama.

- **ACKs Básicos**: Cliente poderia confirmar blocos recebidos.

- **Timeout e Retransmissão**: Reenviar pacotes não confirmados após um tempo.

Impacto do Overhead na Escolha do Protocolo

Para arquivos pequenos (ex: a.txt), o overhead do TCP pode ser maior que o próprio arquivo, tornando o UDP uma opção mais eficiente em situações onde a perda de dados é tolerável.

Para arquivos grandes (ex: b.txt), o overhead do TCP é diluído no volume total de dados e sua confiabilidade compensa o custo adicional, garantindo a integridade e a ordem dos dados transmitidos.

Exemplo de Código Aprimorado (UDP Confiável)

```
# No cliente UDP:
seq_number = 0
while True:
    chunk, _ = udp_sock.recvfrom(1024)
    if chunk.startswith(b"SEQ:"):
        seq = int(chunk.split(b":")[1])
        if seq == seq_number:
            udp_sock.sendto(f"ACK:{seq}".encode(), server_addr)
            seq_number += 1
```

Conclusão

O FTCP evidenciou os trade-offs entre TCP e UDP, destacando que:

- TCP é indispensável para transferências críticas, apesar do alto overhead.
- UDP é viável apenas em cenários onde velocidade é prioritária e perdas são toleráveis.

A análise do PCAP reforçou a importância de escolher o protocolo conforme o tamanho do arquivo e requisitos de confiabilidade.