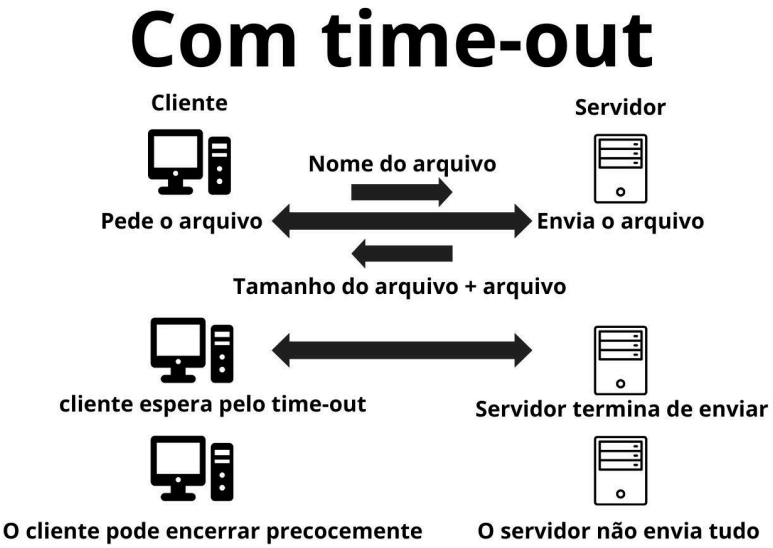
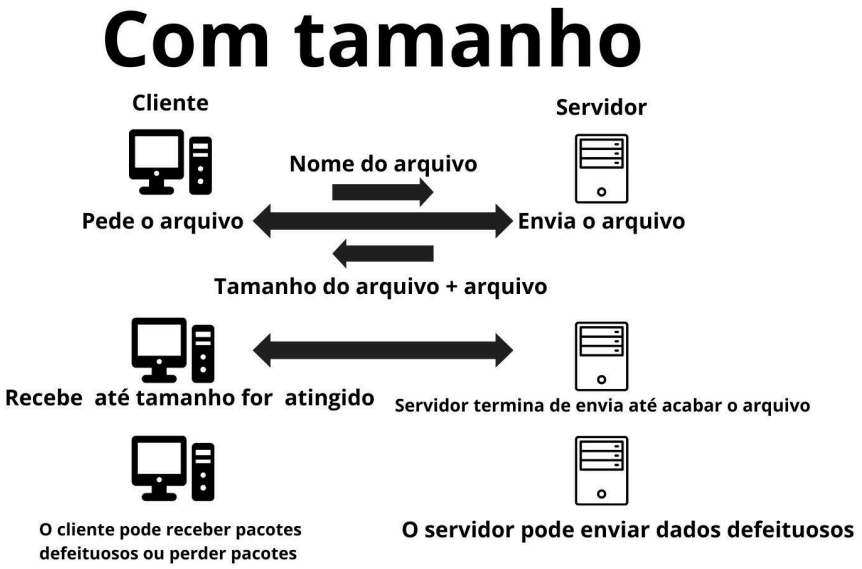
O problema começa desde o princípio, ao se utilizar o protocolo UDP para um servidor de arquivos. O protocolo UDP é orientado a pacotes e não a conexões(cada pacote é enviado de forma individual),sendo projetado para ser rápido e não confiável, algo que não faz sentido ser preconizado em um servidor de arquivos. A ausência de controle de fluxo neste protocolo pode resultar na perda de pacotes durante a transferência, sem qualquer garantia de reenvio ou da integridade dos dados. O processo simplesmente prossegue para a próxima iteração e envio, ignorando eventuais falhas na transmissão anterior. No caso de um servidor de arquivos, optar por usar UDP não é uma escolha tão adequada, já que ele exige confiabilidade e não agilidade.

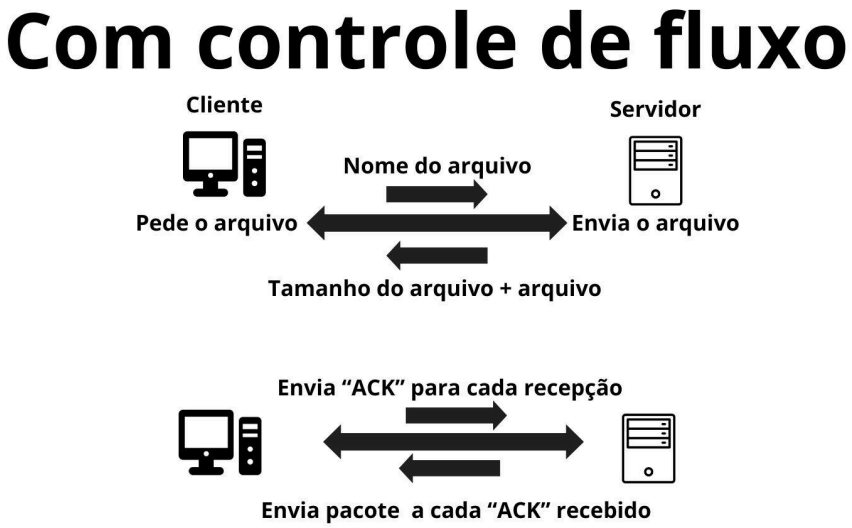
O uso do timeout não é o mais correto, pois é difícil garantir uma definição de um tempo ideal, que pode ser muito curto ou longo demais. Isso pode levar à desconexão antes de concluir a transmissão completa dos dados, tanto no cliente quanto no servidor.

A solução do envio do tamanho é correta, porém os problemas estão associados à própria natureza do UDP. Para arquivos pequenos, não se faz tão necessário um mecanismo de confiabilidade, diferentemente de arquivos maiores, onde a confiabilidade é essencial, podendo ocorrer erros no envio e recepção de dados que podem atrapalhar o programa no geral. Ademais, o processamento dos dados podem ser mais lento em um dos lados, possibilitando que um lado envie dados enquanto outro está processando envio anteriores, já que cada envio é individualizado.

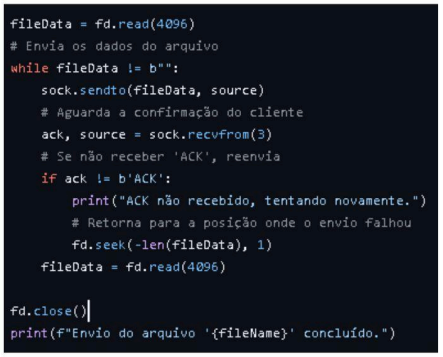




A solução para esse problema é simples, basta só criar o controle de fluxo, sinalizando o recebimento dos pacotes e, caso não sejam recebidos, realizar o envio desses novamente. Outra alternativa é fazer uso do TCP, já que ele provém todos esses mecanismos de forma nativa. O TCP oferece controle de fluxo e retransmissão de pacotes perdidos, tudo de maneira automática, sem precisar implementar esses mecanismos manualmente. Implementar o TCP seria a solução mais viável, pois não é preciso programar manualmente cada funcionalidade de confiabilidade, que já são automáticas no TCP.



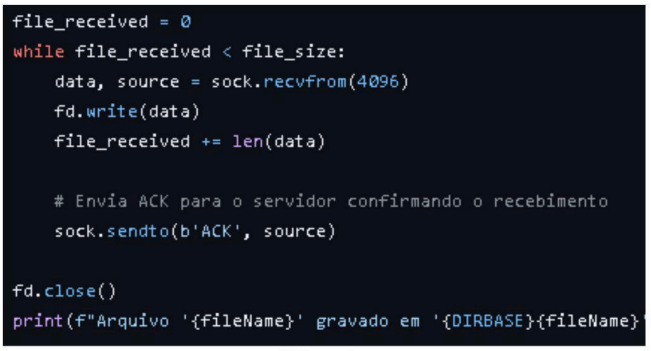
:



```
fileData = fd.read(4096)
# Envia os dados do arquivo
while fileData != b"":
    sock.sendto(fileData, source)
    # Aguarda a confirmação do cliente
    ack, source = sock.recvfrom(3)
    # Se não receber 'ACK', reenvia
    if ack != b'ACK':
        print("ACK não recebido, tentando novamente.")
        # Retorna para a posição onde o envio falhou
        fd.seek(-len(fileData), 1)
    fileData = fd.read(4096)

fd.close()
print(f"Envio do arquivo '{fileName}' concluído.")
```

Servidor



```
file_received = 0
while file_received < file_size:
    data, source = sock.recvfrom(4096)
    fd.write(data)
    file_received += len(data)

    # Envia ACK para o servidor confirmando o recebimento
    sock.sendto(b'ACK', source)

fd.close()
print(f"Arquivo '{fileName}' gravado em '{DIRBASE}{fileName}'
```

Cliente

Em conclusão, o uso do protocolo UDP em servidores de arquivos não é adequado, pois ele não garante a confiabilidade necessária para transferências de dados, especialmente quando lidamos com arquivos grandes. O UDP pode ser rápido, mas a falta de controle de fluxo e a ausência de garantia de reenvio de pacotes tornam o processo arriscado, com risco de perda de dados. A melhor solução seria usar o protocolo TCP, que já oferece controle de fluxo e retransmissão automática de pacotes perdidos, garantindo maior confiabilidade e simplificando a implementação, tornando-o a escolha mais apropriada para esse tipo de aplicação.