Trabalho Prático Primeira Parte - Transporte

22 de junho de 2025

1 Objetivo

Você já pensou sobre o que de fato é uma conexão? Por que protocolos como o TCP requerem que uma conexão seja feita, enquanto outros, como UDP, não?

Neste trabalho você irá implementar um protocolo *ad hoc* na camada de transporte para controle de fluxo de dados: o SLOW. O SLOW tem algumas semelhanças ao QUIC: quando os responsáveis pela implementação do QUIC o estavam planejando, se depararam com o desafio de fazer com que sistemas operacionais importantes, como Linux, Windows e Mac, implementassem a nível de kernel um novo protocolo de transporte, pois sem suporte a nível de kernel seria impossível de usar. Sendo assim, decidiram utilizar o UDP, por ser um protocolo leve e que já estava implementado em quase todos os kernels, como infraestrutura para o QUIC.

O SLOW também utiliza o UDP como infraestrutura para trocar mensagens, e adiciona funcionalidades em cima dele. A seção a seguir relaciona os tipos de mensagens e atores (**central** e **peripheral**) que deverão ser suportados.

Importante: Apenas o **peripheral** deve ser implementado para este trabalho. A implementação deve ser feita em C ou C++, embora você possa implementar em outras linguagens caso queira testar (mas a submissão deve ser em C ou C++).

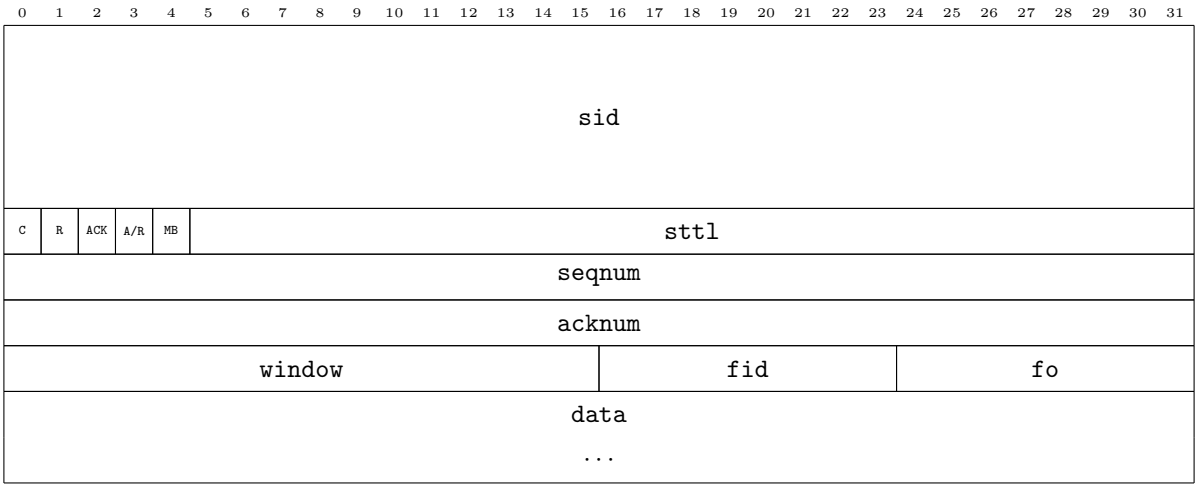
Obs: O próximo trabalho irá utilizar este protocolo como base para transmissão de dados, então é uma boa ideia tornar sua implementação o mais reutilizável e bem-feita possível.

2 Protocolo SLOW

O protocolo usa a porta `udp/7033` para troca de mensagens, tanto nos *centrals* quanto nos *peripherals*.

2.1 Pacote SLOW

O pacote SLOW não deve exceder 1472 bytes (incluindo o cabeçalho). Todos os bits são *little endian*.

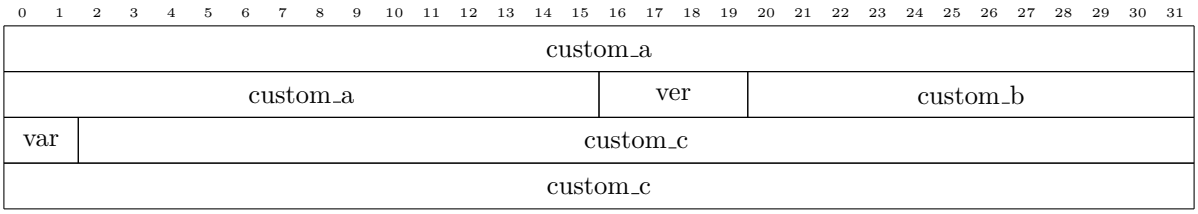


- `sid`: Session ID (UUIDv8), ID da sessão, único para uma sessão entre um *central* e um *peripheral* - 128 bits

- `flags`: SLOW Flags (C R A A/R MB) - 5 bits
  - `Connect`: este pacote é um pacote de início de *3-way connect*
  - `Revive`: este pacote é um pacote de *0-way connect*
  - `Ack`: este pacote confirma o recebimento do pacote no campo `acknum` (se o pacote é somente de `ack`, então `seqnum` e `acknum` são o mesmo número, do contrário ficaríamos mandando `ack`s infinitamente)
  - `Accept/Reject`: este pacote aceita ou rejeita uma conexão (*0-way* ou *3-way*)
  - `More Bits`: este pacote é parte de um pacote maior que foi fragmentado, e mais fragmentos chegarão
- `sttl`: Session TTL (ms), número de segundos até a sessão expirar (este campo é de uso exclusivo do *central*, valores deste campo passados do *peripheral* para o *central* serão ignorados) - 27 bits
- `seqnum`: Sequence Number, número único de um pacote em uma sessão, deve ser configurado pelo *central* - 32 bits
- `acknum`: Acknowledgement Number, este é o *sequence number* do último pacote recebido (ignorar se a *flag* `ack` estiver desligada) - 32 bits (**obs:** `seqnum` não deve ser incrementado em pacotes `ack`, ou seja, um pacote `ack` deve conter o `seqnum` igual ao `acknum`)
- `window`: Window Size, quantidade de espaços para receber pacotes no buffer de leitura após o último `ack` - 16 bits
- `fid`: Fragment ID, pacotes com o mesmo `fid` devem ser agrupados sequancialmente - 8 bits
- `fo`: Fragment Offset, ordem de um pacote dentro de um *Fragment ID* - 8 bits
- `data`: Data, máximo de 1440 bytes.

2.1.1 UUID

O UUID utilizado para session IDs deve ser do formato UUIDv8 conforme especificado no RFC9562:



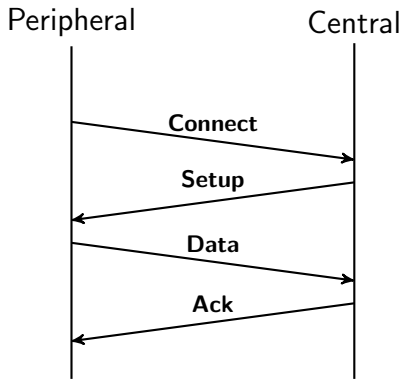
Aqui, `var` e `ver` devem ter valores conforme especificado no RFC9562, enquanto `custom_a`, `custom_b` e `custom_c` podem ser quaisquer.

2.2 Mensagens

Nesta seção estão listados os casos de uso do protocolo, contendo descrições das respecitvas mensagens utilizadas em cada um deles.

2.2.1 3-way connect

O *peripheral* inicia uma sessão com o *central*.

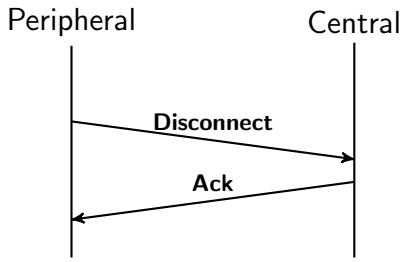


1. `Connect` (peripheral):
    - `sid`: Nil UUID, conforme RFC9562.
    - `sttl`: 0
    - `flags`: connect (1)
    - `seqnum`: 0
    - `acknum`: 0
    - `window`: tamanho restante no buffer de recebimento
    - `fid`: 0
    - `fo`: 0
    - `data`: campo inexistente nessa mensagem
2. `Setup` (central): configurações de início de sessão
    - `sid`: UUID da sessão, deve ser usado daqui em diante
    - `sttl`: `sttl` da sessão, deve ser usado daqui em diante
    - `flags`: Accept (1) / Reject (0)
    - `seqnum`: primeiro `seqnum` da sessão, deve ser usado e incrementado daqui em diante
    - `acknum`: 0
    - `window`: tamanho restante no buffer de recebimento
    - `fid`: 0
    - `fo`: 0
    - `data`: campo inexistente nessa mensagem
3. `Data` (peripheral): início de envio de dados
    - `sid`: UUID da sessão
    - `sttl`: `sttl` da sessão
    - `flags`: Ack (1) — More Bits (1) a depender do caso
    - `seqnum`: próximo número de sequência
    - `acknum`: número de sequência do último pacote recebido
    - `window`: tamanho restante no buffer de recebimento
    - `fid`: 0
    - `fo`: 0
    - `data`: quantidade arbitrária de dados
4. `Ack` (central): confirmação da mensagem de dados

- `sid`: UUID da sessão
- `sttl`: `sttl` da sessão (atualizado)
- `flags`: Ack (1)
- `seqnum`: próximo número de sequência
- `acknum`: número de sequência do último pacote recebido (o de dados)
- `window`: tamanho restante no buffer de recebimento
- `fid`: 0
- `fo`: 0
- `data`: campo inexistente nessa mensagem

2.2.2 Disconnect

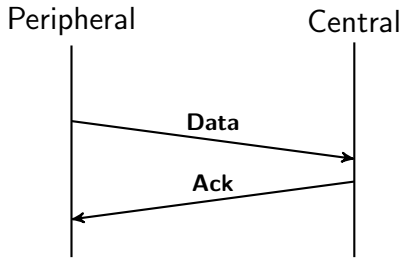
Faz com que a sessão seja posta em modo inativo, podendo ser reativada utilizando o *0-way connect*. Pacotes recebidos em sessões inativas que não de `revive` deverão ser ignorados por ambas as partes.



1. `Disconnect` (peripheral)
   - `sid`: UUID da sessão
   - `sttl`: `sttl` da sessão
   - `flags`: Ack (1), Connect (1), Revive (1) – Quando ambos `connect` e `revive` estiverem ligados, isso simboliza um `disconnect`.
   - `seqnum`: próximo número de sequência
   - `acknum`: número de sequência do último pacote recebido
   - `window`: 0
   - `fid`: 0
   - `fo`: 0
   - `data`: campo inexistente nessa mensagem
2. `Ack` (central) tal qual 4

2.2.3 Dados

O *peripheral* envia dados para o *central*.

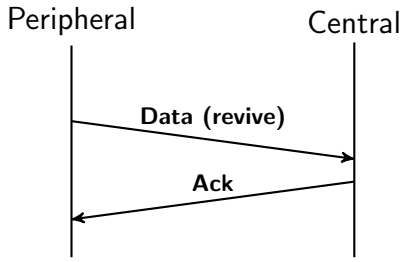


1. `Data` (peripheral) tal qual 3
2. `Ack` (central) tal qual 4

Obs: Ver Anexos A e B sobre fragmentação e janela deslizante.

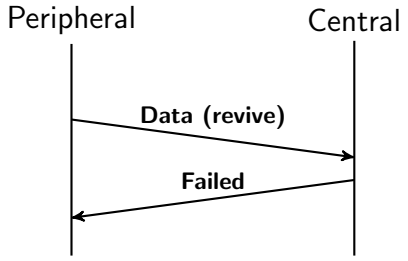
2.2.4 0-way connect

Se o `session time (stime)` de uma conexão anterior ainda estiver válido, é possível reiniciar a mesma conexão com apenas uma *flag* na mensagem. Nesse caso a mensagem é a mesma da de dados, mas com a *flag* `revive` valendo 1.



1. `Data (revive)` (peripheral) tal qual 3 mas com a flag `revive` ligada
2. `Ack` (central) tal qual 4 mas com a flag `accepted/failed` ligada.

Ou, em caso de falha:



1. `Data (revive)` (peripheral) tal qual 3 mas com a flag `revive` ligada
2. `Failed` (central) tal qual 4, mas com a flag `accepted/failed` desligada
   - `sid`: Nil UUID
   - `sttl`: 0
   - `flags`: Reject (0)
   - `seqnum`: 0
   - `acknum`: 0
   - `window`: p
   - `fid`: 0
   - `fo`: 0
   - `data`: campo inexistente nessa mensagem

2.3 Resumo: mensagens a serem implementadas

Para facilitar o desenvolvimento, a seguir está uma lista resumida contendo apenas as mensagens que devem ser implementadas pelo seu *peripheral*:

1. Enviar `3-way connect` e receber `accept/fail`;
2. Enviar dados (começando uma *0-way connection* ou não) e receber `ack` (se não receber, reenviar dados);
3. Enviar `disconnect`.

2.4 Exemplos de pacotes SLOW

3 Testes

Uma vez implementado, você deve testar seu *peripheral* utilizando um *central* que está em *slow.gmelodie.com:7033*

4 Entrega

Entregue o repositório do seu código como um tarball zipado `.tar.gz`. Explique sucintamente o funcionamento em um arquivo `README.md` e forneça exemplos de utilização. Não esqueça de documentar bem o código com *docstrings* e comentários. No entanto, repense sua abordagem se sentir necessidade de comentar demasiadamente o código, pois um código bem organizado e legível dispensa a maioria dos comentários.

O trabalho pode ser feito em grupo de até 3 alunos. O nome de todos deve ser informado.

Entrega: 29/06/2025

A Fragmentação

Quando os dados a serem enviados excedem o máximo de 1456 bytes, será necessário dividi-los em vários pacotes. Similarmente, quando um pacote fragmentado for recebido, será necessário remontar o pacote. Para tanto, o protocolo SLOW utiliza a flag `MB` (*More Bits*) juntamente com os campos `fo` (*Fragment Offset*) e `fid` (*Fragment ID*).

Quando um fragmento de um pacote chega, ele é identificado por um `fid`, ou seja, todos os fragmentos pertencentes a uma mesma mensagem terão o mesmo `fid`. Já a ordem dos fragmentos é identificada pelo `fo`, ou seja, o pacote de `fo = 0` é o primeiro pacote do fragmento, seguindo do `fo = 1`, seguido do `fo = 2`, e assim por diante.

Para saber ainda há fragmentos a serem recebidos ou não, o último fragmento vem com a flag `mb` (*More Bits*) desligada, enquanto que em todos os outros fragmentos ela deve estar ligada. Assim que um fragmento com a flag MB desligada é recebido, o receptor sabe que não deve esperar mais fragmentos daquela mensagem.

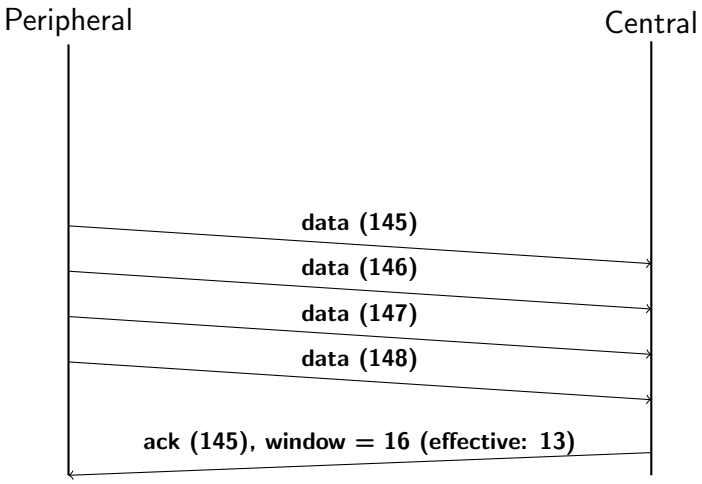
B Janela deslizante

Como o protocolo SLOW implementa confirmação de recebimento (através de mensagens `ack`), é preciso saber quantas mensagens podem ser enviadas sem que um `ack` seja recebido em retorno. Por exemplo, se um *peripheral* envia 20 pacotes slow e não recebe nenhum `ack` em retorno, é possível que nossas mensagens estejam sendo ignoradas porque o *buffer* de recebimento do *central* esteja cheio.

Para resolver esse problema, o SLOW utiliza janelas deslizantes, que são essencialmente contadores que mostram quantos bytes livres há no *buffer* do remetente. No protocolo SLOW, esse contador está presente no campo `window` dos pacotes.

Por exemplo, imagine que o último pacote recebido do *central* pelo seu *peripheral* tem o valor 16 no campo `window` e `acknum = 145`. Isso significa que, quando o *central* processou o pacote de `seqnum = 145`, ele ainda podia receber 16 bytes. No entanto, se já tivermos mandado os pacotes `seqnum = 146, 147` e `148`, temos que supor que a janela efetiva do *central* é de 13 bytes (16 reportados menos os 3 pacotes em trânsito que ainda não chegaram, mas vão chegar). Este exemplo está ilustrado no esquemático abaixo.

Seu programa deve respeitar as janelas reportadas pelo *central*, bem como informar corretamente suas próprias janelas de acordo com o *buffer* por você definido.


Peripheral
Central
data (145)
data (146)
data (147)
data (148)
ack (145), window = 16 (effective: 13)