Engenharia de Telecomunicações
Projetos de Protocolo

Prof. Marcelo Maia Sobral
Ameliza Souza Correa
Camilla Barreto de Sousa
Elisa Rodrigues

Manual do Protocolo de Comunicação

18/12/2020

1 Introdução

Neste manual será apresentado a API de utilização do protocolo de enlace que foi estruturado para funcionar em camadas. O protocolo define a subcamada *Framing* responsável pela transmissão e recepção de quadros, além de implementar mecanismos de enquadramento e detecção de erro.

Para uma aplicação utilizar o protocolo de enlace é necessário implementa-la como uma subcamada, que como no exemplo, chamaremos de *Application*. Após realizar a conexão entre elas a comunicação poderá fluir no nível de abstração da aplicação, possibilitando transmissão e recepção de dados.

1.1 Implementando subcamadas

As subcamadas *Framing* e *Application* são responsáveis por tarefas distintas, mas as duas herdam da classe *Sublayer* que padroniza a estrutura de comunicação entre elas e de futuras subcamadas que podem existir entre elas. A classe *Sublayer* por sua vez herda da classe *Callback* que é responsável por tratar eventos em interfaces.

Por padrão a subcamada *Framing* é responsável por monitorar uma interface serial, onde ocorrerá a comunicação entre programas. A subcamada *Application* também deve monitorar uma interface, como por exemplo a entrada padrão ou um arquivo de texto, isso irá depender de como o usuário deseja que a aplicação funcione.

1.1.1 Escopo de uma camada de aplicação

Como a classe *Application* herda de outras classes, alguns métodos devem ser implementados nela. Segue abaixo um escopo de uma camada de aplicação, com os métodos que precisam ser implementados para que ocorra a comunicação.

```
1  #!/usr/bin/env python3
2  import poller
3  from poller import Callback
4  from sublayer import Sublayer
5  from serial import Serial
6
7  class Application(Sublayer):
8
9      def __init__(self, file, tout: float):
10         Sublayer.__init__(self, file, tout)
11
12     def handle(self):
13         ## método executado quando ocorrer um evento na interface monitorada
14
15     def handle_timeout(self):
16         ## método executado quando ocorrer um timeout
17
```

```
18      def receive(self, msg):
19          ## método executado quando for chamado pela camada inferior
```

1.1.2 Métodos

Como indicado no escopo, cada método tem sua funcionalidade. Segue abaixo definições mais detalhadas de cada um deles:

- **handle:** Por ser um protocolo orientado a eventos, quando um novo evento ocorre, uma ação deve ser executada e o método chamado para trata-la é o handle. A *Application* deve monitorar os eventos de uma interface de entrada de dados, onde estes serão lidos, reservados em um buffer para então serem enviados para o enquadramento. No enquadramento é analisado a quantidade limite de bytes, definida nesta aplicação em 128 bytes. Caso esse limite seja ultrapassado na transmissão, os dados são divididos, sendo transmitidos em partes menores, mas tendo o tamanho permitido os dados são encaminhados para o tratamento do cálculos do CRC e inclusão de flags, para assim, montar o quadro e enviá-lo pela porta serial definida. Em contrapartida, no recebimento os dados que ultrapassam esse limite são descartados.
- **handle_timeout:** Cada tarefa que trata o evento tem um tempo máximo para sua execução. Este tempo é definido como parâmetro para a classe, no qual pode também ser indefinido (set 0). Ao ocorrer um estouro desse tempo pré-definido um timeout é gerado e tratado por este método, no qual irá finalizar a tarefa e os dados em buffer serão descartados.
- **receive:** Esse método é responsável por receber dados da camada inferior. Esses dados provém da interface serial no qual são recebidos pela classe *Framing*. Os dados chegam nessa camada já desenquadrados e com a verificação de erros concluída. Após receber esses dados, este método é responsável por imprimi-los no terminal para o usuário.

1.1.3 Como instanciar uma subcamada de aplicação

Para instanciar uma *Application* devem ser passados alguns parâmetros na inicialização da classe. Os argumentos necessários são *interface de dados* e *timeout*.

- **O parâmetro interface de dados:** Esse parâmetro indica qual será o descritor de arquivos provedor de dados para a camada *Application* e que deve ser monitorido pelo método *handle()*. Esses dados são recebidos através da leitura escolhida, podendo ser de arquivo, terminal ou outros, e então enviados para a classe inferior, no caso o *Framing*.
- **O parâmetro timeout:** Este é o tempo máximo em segundos em que o handle poderá executar sua tarefa. Se o estouro desse tempo ocorrer, o método *handle_timeout()* será executado. Esse valor pode ser definido como 0 caso não deseje aplicar um tempo limite.

1.2 Instanciando subcamadas e conectando

Está classe Sublayer é a que representa o esqueleto das camadas implementadas neste protocolo. Os métodos presentes nela são herdados pelas classes adjacentes e devem ser definidos posteriormente.

```
1
2  #!/usr/bin/env python3
3  import poller
4  from poller import Callback
5
6
7  class Sublayer(poller.Callback):
```

```
8
9     def __init__(self, fd, timeout):
10        Callback.__init__(self, fd, timeout)
11        self.lowerLayer = None
12        self.upperLayer = None
13
14    def handle(self):
15        ## método chamado na ocorrência de novos eventos
16        pass
17
18    def handle_timeout(self):
19        ## método chamado na ocorrência de um timeout
20        pass
21
22    def send(self):
23        ## Recebe os octetos da camada superior, trata os dados
24        ## e envia para a camada inferior
25        pass
26
27    def receive(self):
28        ## Recebe os octetos da camada inferior, trata os dados
29        ## e envia para a camada superior
30        pass
31
32    def connect(self, lower, upper):
33        ## Realiza a conexão entre as camadas
34        self.lowerLayer = lower
35        self.upperLayer = upper
```

1.2.1 Métodos

Esta classe representa todos os métodos mínimos necessários para a implementação das camadas. Como visto anteriormente os métodos presentes na classe *Application* são implementados na mesma. Podemos observar que na camada de exemplo não temos o método *send* definido, pois a camada de aplicação seria a ultima camada, logo não possui uma camada superior.

- **handle:** Método que trata os eventos ocorridos. Neste protocolo esse método está vinculado ao recebimento e leitura de dados à qualquer momento, seja da interface de dados (camada de aplicação) ou seja da interface serial (camada de enquadramento).
- **handle_timeout:** Este método trata as ocorrência de timeout. Ao haver o estouro do tempo máximo de execução de tarefas, esse método é convocado.
- **send:** Este método é implementado para receber os dados da camada superior, fazer o tratamento necessário e então enviá-los para a camada inferior.
- **receive:** Este método é implementado para receber os dados da camada inferior, fazer o tratamento necessário e então enviá-los para a camada superior.
- **connect:** Este método define quais são as camadas superiores e inferiores das classes criadas.