FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Engenharia Electrotécnica

Sistemas de Telecomunicações

2012/2013

Trabalho 2:
Protocolos de nível Lógico com janela deslizante

Mestrado integrado em Engenharia Eletrotécnica e de Computadores

http://tele1.dee.fct.unl.pt

Índice

1. OBJETIVO

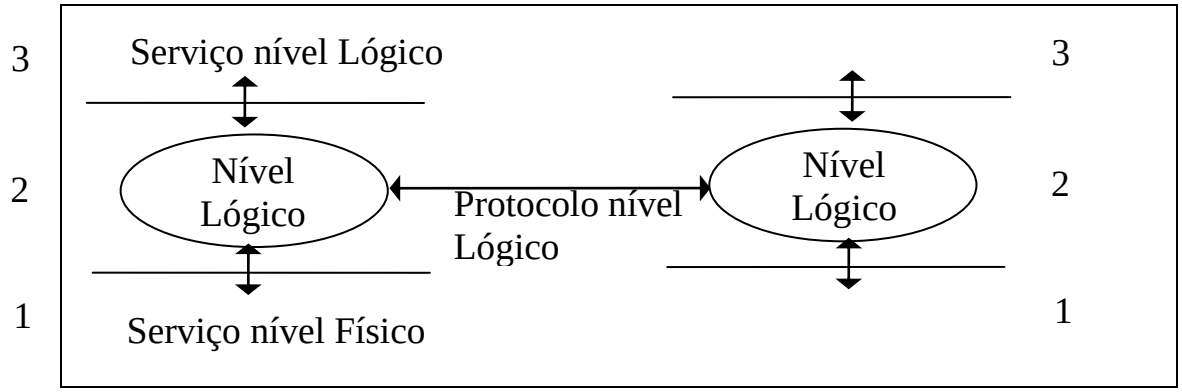
Familiarização com os protocolos de nível lógico baseados em janela deslizante do tipo Stop&Wait e Go-Back-N.

O trabalho consiste no desenvolvimento de um protocolo de nível lógico baseado em janela deslizante, do tipo Go-Back-N com NAck, de forma faseada. Para isso, é fornecido um simulador de sistema desenvolvido na disciplina sobre sockets TCP, que simula o funcionamento do protocolo de nível rede e de nível físico.

> **Sugestões**: Em certas partes do enunciado aparece algum texto formatado de um modo diferente que começa com a palavra "Sugestões". Não é obrigatório seguir o que lá está escrito, mas pode ser importante para os alunos ou grupos onde ainda não haja um à-vontade muito grande com programação, estruturas de dados e algoritmia.

2. ESPECIFICAÇÕES

Pretende-se desenvolver um protocolo de nível Lógico para uma ligação física ponto-a-ponto, que interage com os protocolos de nível Rede e de nível Físico através respetivamente das interfaces dos serviços de nível Lógico e de nível Físico. Desta forma vai ser simulado o sistema representado abaixo, através do conjunto de primitivas dos dois serviços.



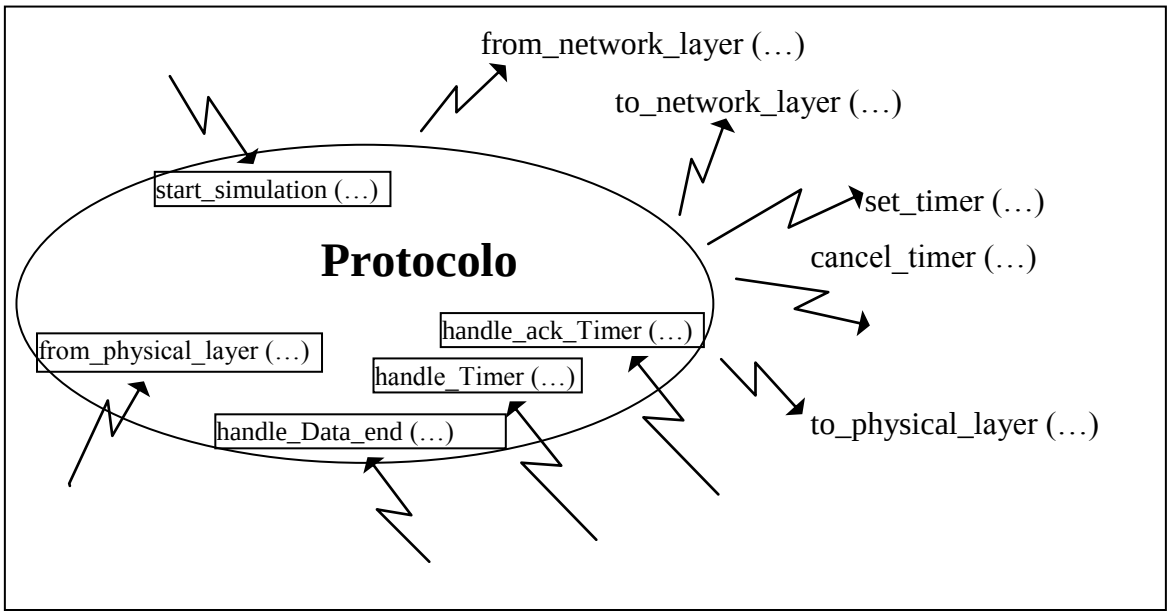
A abordagem proposta foi inspirada no simulador descrito nas secções 3.3 e 3.4 do livro recomendado (Computer Networks 5ª edição), embora faça uma simulação um pouco mais próxima da realidade (e.g. considera o tempo de transmissão das tramas de dados). O protocolo de nível lógico vai reagir a eventos e pode invocar comandos.

O nível Lógico começa com o evento que é servido pelo método `start_simulation()`. Para receber pacotes (*strings*) do nível Rede, o nível Lógico usa o método `from_network_layer()` e para enviar pacotes (*strings*) para o nível Rede invoca o método `to_network_layer()`.Os dados devem ser entregues pela mesma ordem em que foram recebidos, recuperando de erros do canal de nível Físico.

O protocolo pode enviar tramas para o nível Físico usando `to_physical_layer()` e pode receber tramas do nível Físico no seu método `from_physical_layer()`, invocado pelo protocolo de nível Físico.

Por fim, pode usar um conjunto de funcionalidades de suporte para gerir temporizadores. Usando os métodos `set_timer` e `cancel_timer`, pode armar ou cancelar um temporizador identificado por um número maior ou igual a zero (designado de *key*). Quando o tempo expira, o

método `handle_timer()`, é chamado. A figura em baixo ilustra o que se acabou de descrever. Os alunos devem programar o balão designado por "Protocolo".



2.1. TIPOS DE TRAMAS

O protocolo pode enviar ou receber três tipos de tramas: DATA, ACK ou NACK.

Tramas de dados (DATA)

As tramas de dados têm os seguintes campos:

- Número de sequência (`seq`)
- Confirmação (`ack`)
- Informação (`info`)

As tramas são numeradas, com um número `seq` compreendido entre `0` e um número máximo especificado numa janela (`sim.get_max_sequence()`).O campo `ack` contém o número de sequência da última trama de dados recebida.

As tramas de dados têm um tempo de transmissão não nulo, portanto o seu envio é realizado em duas fases:

1) É iniciado o envio da trama com o método `to_physical_layer`;
2) É recebido um evento `handle_Data_end`, a informar que terminou o envio da trama de dados.

Entre o início do envio da trama de dados e a receção do evento de fim de trama, não podem ser enviadas outras tramas.

Tramas de confirmação de receção (ACK)

As tramas de confirmação de receção (ACK) são geradas após a receção de tramas de dados, indicando o número de sequência da última trama de dados recebida com sucesso. Considera-se a trama instantânea, sendo enviada numa única invocação do método `to_physical_layer`. A trama ACK contém um único campo:

- Confirmação (`ack`)

Como é mais eficiente enviar esta informação através de tramas de dados (por *piggybacking*), definiu-se um temporizador auxiliar (*ack_timer*) que pode ser usado para esperar

por uma trama de dados, só enviando o ACK após este tempo. Desta forma, após receber uma trama de dados deve-se:

1) Armar o temporizador de ACK, usando o método `start_ack_timer()`;

2a) Se aparecer uma trama de dados para transmitir, o temporizador pode ser cancelado com o método `cancel_ack_timer()`;

2b) Se o relógio expirar, é gerado o evento `handle_ack_Timer`, devendo-se enviar a trama ACK.

Tramas de confirmação negativa (NACK)

As tramas de confirmação negativa (NACK) podem ser geradas em protocolos de janela deslizante quando são recebidas tramas de dados fora de ordem (e.g. está-se à espera da trama 0 e recebe-se a 1) em resultado de se ter saltado algum número de sequência. Também é considerada uma trama instantânea, e tem apenas um campo:

- Confirmação (`ack`), com o número de sequência da trama de dados que deve ser retransmitida.

O recetor desta trama deverá retransmitir a trama de dados pedida o mais depressa possível. Para evitar que o emissor esteja continuamente a retransmitir a mesma trama, o NACK só pode ser gerado uma vez para cada número de sequência.

2.2. PROTOCOLOS DE NÍVEL LÓGICO

Nas secções 3.3 e 3.4 do livro recomendado são descritos os três tipos básicos de protocolos de nível lógico que se vão realizar neste trabalho. Esta seção reproduz resumidamente os aspetos fundamentais de cada um, mas recomenda-se uma leitura atenta ao livro para uma correta realização do trabalho.

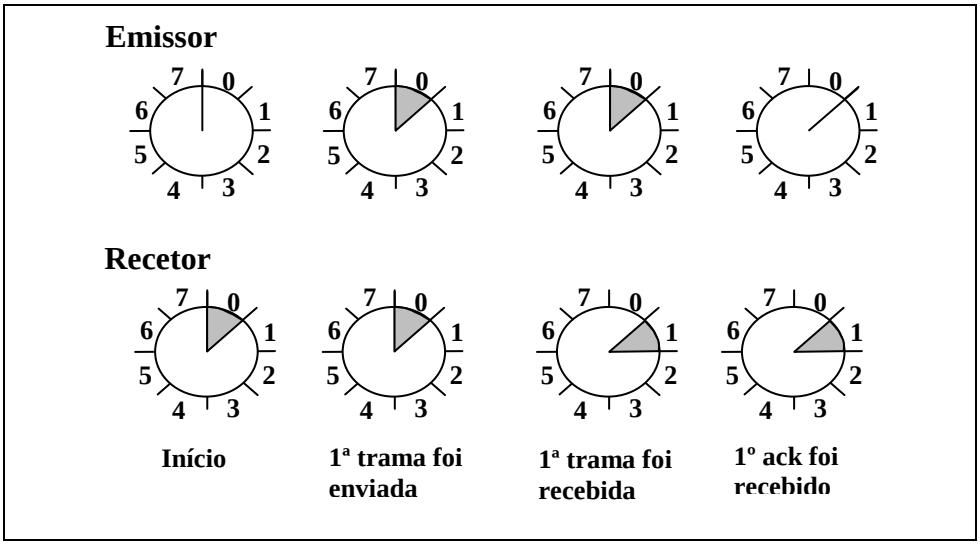
Protocolo Utópico

O protocolo utópico está descrito na secção 3.3.1 do livro e corresponde a realizar o envio das tramas sequencialmente sem nenhum mecanismo de recuperação de erros. O recetor limita-se a receber as tramas e enviar para o nível rede.

É fornecido o código completo de uma versão bidirecional deste protocolo juntamente com o enunciado.

Protocolo Stop & Wait

O protocolo Stop & Wait está descrito na secção 3.4.1 do livro e corresponde a um protocolo de janela deslizante com janelas de transmissão e receção unitárias. Os emissores mantêm o próximo número de sequência a transmitir e o recetor o próximo receber, de acordo com o esquema à direita.

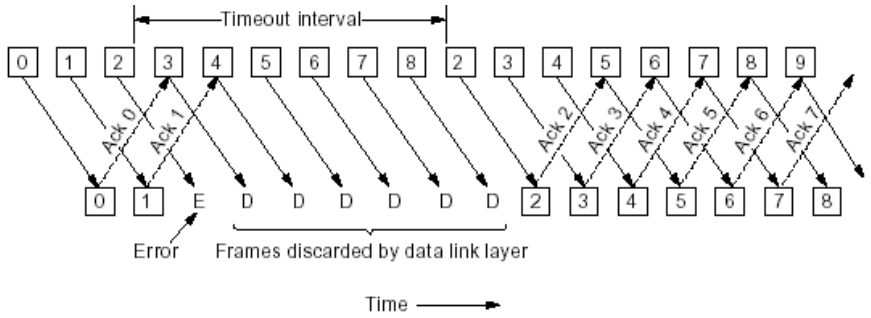


O emissor deve armar um temporizador cada vez que envia uma trama. Caso expire, deve reenviar a trama. Quando um ACK (ou trama de dados) confirmam a trama, deve-se cancelar o temporizador e enviar a próxima trama.

Protocolo Go-Back-N

O protocolo Go-Back-N está descrito na secção 3.4.2 do livro. Corresponde a um protocolo de janela deslizante com janela de receção unitária, onde é usado *pipelining* para melhorar o desempenho quando o produto atraso*banda é elevado.

Neste caso, o emissor vai necessitar de manter um array de buffers de transmissão, podendo transmitir até um máximo de `sim.get_send_window()` tramas sem receber confirmações (i.e. janela de transmissão). Quando ocorre um erro, tem de retransmitir todas as tramas de dados a partir da que não foi confirmada, como está representado na figura.



A gestão de temporizadores é um pouco mais complexa neste protocolo. O ideal será manter um temporizador para cada trama pendente individual, mas pode-se também usar apenas um único temporizador relativo à trama mais antiga em cada instante. Neste segundo caso, o temporizador seria reiniciado cada vez que é confirmada uma nova trama de dados.

> **Sugestões**: Como a gestão de vários temporizadores pode tornar o programa um pouco mais complexo, sugere-se que a realização seja feita em duas fases: i) na primeira pode-se usar apenas um temporizador; ii) na segunda passar a usar um temporizador para cada trama pendente.

Protocolo Go-Back-N com NACK

O protocolo Go-Back-N com NACK não está descrito diretamente no livro. Corresponde ao protocolo Go-Back-N a que se junta a trama NACK, tal como está descrita na secção 3.4.3 do livro, introduzida no contexto do protocolo de retransmissão seletiva. No Go-Back-N é um pouco mais simples, pois vai ser necessário transmitir tudo a partir de `seq`. Portanto, permite acelerar a resposta à perda de tramas face a esperar pelo expirar do temporizador.

2.3. CENÁRIO DE SIMULAÇÃO

Neste trabalho simula-se uma rede com tempo de propagação variável e com uma taxa de perda de tramas constante. São usadas dois programas:

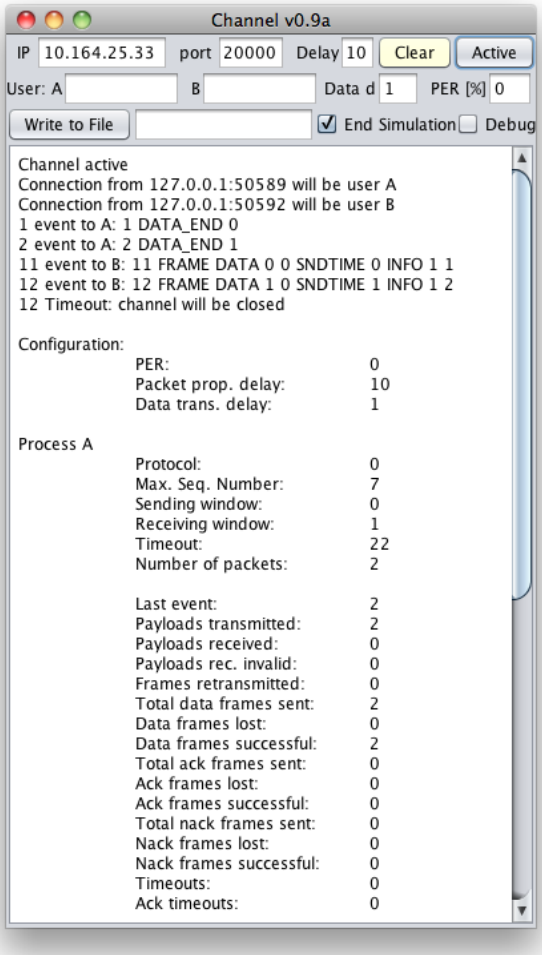
- *Protocol* (*protocolo*) – realiza o protocolo de nível lógico e emula o nível rede, controlando o envio e receção de pacotes de dados numerados sequencialmente. A parte do nível lógico será feita pelos alunos;
- *Channel* (*canal*) – liga duas instâncias de *Protocol*, emulando o tempo de propagação e perdendo tramas com uma certa probabilidade de perda.

Após arrancar, o *canal* aceita duas ligações TCP de dois programas *Protocol*, começando a partir daí a simulação. O canal realiza um sequenciador de eventos discretos, recebendo os comandos gerados pelos protocolos e gerando os eventos relacionados com o nível físico e com os temporizadores apresentados anteriormente ordenados de acordo com o tempo de simulação. O tempo de simulação é medido em unidades de tempo virtuais, designadas de *tics*.

Tanto o *protocolo* como o *canal* fornecidos com o enunciado escrevem resumidamente (ou exaustivamente, no modo *debug*) os eventos e comandos que são gerados, e o conteúdo da fila de espera com eventos à espera de serem disparados.

3. DESENVOLVIMENTO DO PROGRAMA

3.1. APLICAÇÃO CANAL



A aplicação *canal* é fornecida totalmente realizada. Consiste numa aplicação em Java com a interface gráfica representada à direita. Quando se prime o botão "*Active*" o *canal* arranca um *ServerSocket* no IP e porto indicados, ficando preparado para receber ligações de *protocolos*, que serão designados respetivamente de *protocol* (*User*) A e B.

A aplicação *canal* permite realizar várias configurações do cenário de simulação:

- "*Delay*" – tempo de propagação de tramas no canal;
- "*Data d*" – duração de uma trama de dados (tempo entre o envio da trama *Data* e a geração do evento *Data_end*);
- "*PER [%]*" – (*Packet Error Rate*) taxa média de perda de tramas no canal, que pode afetar todas as tramas transmitidas com igual probabilidade;
- "*End Simulation*" – controla se o canal termina automaticamente uma simulação quando não recebe eventos durante um segundo, ou se é deixada a decisão ao utilizador, carregando no botão "*Active*".
- "Write to File" – controla se se escrevem as mensagens ecoadas para o ecrã num ficheiro.

No final da simulação é escrito um relatório com o valor das várias configurações usadas no *canal* e nos *protocolos*, e com várias medidas de desempenho do sistema para os *protocolos* A e B. Destacam-se entre estas medidas:

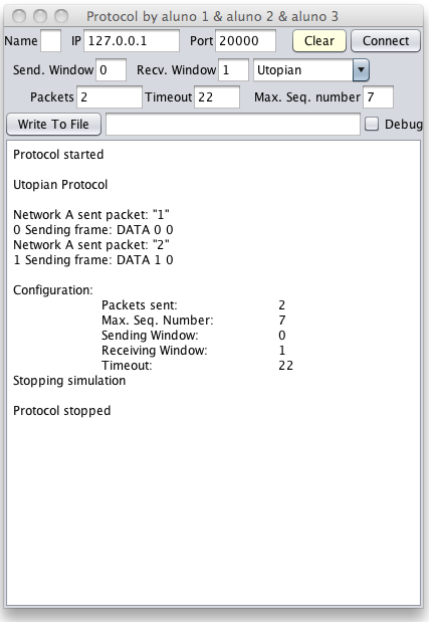
- *Last event* – tempo do último evento que o *protocolo* recebeu/originou, traduzindo-se no atraso total que ocorreu para enviar e receber todos os dados;
- *Payloads transmitted* – número de pacotes transmitidos pelo nível rede;
- *Payloads received* – número de pacotes recebidos no nível rede;
- *Payloads rec. invalid* – número de pacotes recebidos fora de ordem no nível rede;
- *Frames retransmitted* – número de tramas de dados retransmitidas (tem de ser o programador a atualizar explicitamente este contador);
- *Total data frames sent* – número total de tramas de dados enviadas;
- *Data frames lost* – número de tramas de dados perdidas por erros no canal;
- *Data frames successful* – número de tramas de dados recebidas com sucesso;
- *Total ack frames sent* – número total de tramas ACK enviadas;
- *Ack frames lost* – número de tramas ACK perdidas por erros no canal;
- *Ack frames successful* – número de tramas ACK recebidas com sucesso;
- *Total nack frames sent* – número total de tramas NACK enviadas;
- *Nack frames lost* – número de tramas NACK perdidas por erros no canal;
- *Nack frames successful* – número de tramas NACK recebidas com sucesso;
- *Timeouts* – número de eventos Timeout recebidos;
- *Ack timeouts* – número de eventos ACK_Timeout recebidos.

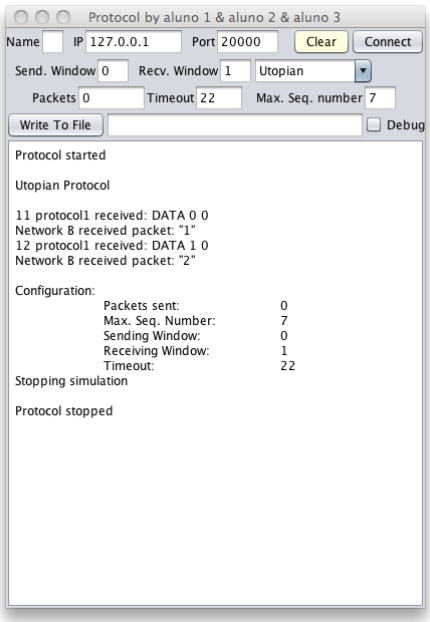
O desempenho dos protocolos realizados vai ser medido usando estas métricas, destacando-se o atraso e o rácio do número total de pacotes transmitidos por pacote de dados. Sugere-se que para as várias variantes do protocolo se meça o desempenho para: i) um cenário sem erros e com timeout ajustado (PER=0 e Timeout=22 tics); ii) cenário com erros e com timeout ajustado (PER=50% e Timeout=22 tics); e iii) cenário com erros e timeout longo (PER=50% e Timeout=44 tics).

Pode correr o *canal* a partir do terminal com o comando: `java -jar Channel.jar`

3.2. APLICAÇÃO PROTOCOLO

O trabalho consiste exclusivamente na realização dos protocolos de nível lógico. Tudo o resto é fornecido completamente realizado. As duas figuras abaixo representam os nós *protocolo* A e B com as mensagens geradas de acordo com a figura do canal apresentada anteriormente.





A interface gráfica permite definir a qual canal é feita a ligação, escolhendo o IP e o porto. A simulação arranca quando se prime o botão "*Connect*" e o canal gera o evento de início de simulação.

Através da interface gráfica, é possível definir:

- *Packets* – o número de pacotes que vão ser enviados durante a simulação;
- *Max. Seq. number*– o número máximo de sequência (geralmente dado por 2^n-1);
- *Send Window* – o tamanho da janela de transmissão;
- *Recv Window* – o tamanho da janela de receção (é sempre 1 neste trabalho);
- *Timeout* – o tempo de espera por uma confirmação antes de reenviar uma trama de dados.

Existe também uma *combo box* que permite escolher o protocolo de nível lógico: *Utopian*; *Stop & Wait*; *Go-Back-N*; e *Go-Back-N* com *Nack*. O objetivo do trabalho é desenvolver o código para os três últimos protocolos.

O programa fornecido é composto por três pacotes, cada um com as classes seguintes:

- Pacote `terminal`:
  - *Terminal.java* (completa) – Classe principal com interface gráfica, que faz a gestão de sincronismo dos vários objetos usados;
  - *Connection.java* (completa) – Thread que trata uma ligação TCP ao canal;
  - *NetworkLayer.java* (completa) – Classe que realiza a interface com o nível rede;
- Pacote `simulator`:
  - *Frame.java* (completa) – Classe que guarda e serializa tramas;
  - *Event.java* (completa) – Classe que guarda e serializa eventos;
  - *Log.java* (completa) – Interface que define a função *Log*;
- Pacote `protocol`:
  - *Simulator.java* (completa) – Interface que define todos os comandos que podem ser usados na realização de um protocolo de nível lógico;
  - *Callbacks.java* (completa) – Interface que define todos os métodos que têm de ser implementados na realização de um protocolo de nível lógico;
  - *Protocol1.java* (completa) – Realização do protocolo de nível lógico Utópico;
  - *Protocol2.java* (**a completar**) – Realização do protocolo de nível lógico Stop & Wait;
  - *Protocol3.java* (**a completar**) – Realização do protocolo de nível lógico Go-Back-N;
  - *Protocol4.java* (**a completar**) – Realização do protocolo de nível lógico Go-Back-N com NACKs;

Os alunos apenas vão modificar estes três últimos ficheiros, com a realização dos protocolos pretendidos, utilizando principalmente os métodos definidos nas interfaces `Simulator` e `Callbacks`, descritos na secção seguinte.

3.2.1. Comandos

No código da classe *Protocol*, que é fornecida aos alunos, pode-se invocar sobre o objeto `sim` os seguintes métodos (que foram definidos na `interface Simulator`). O propósito de cada um está explicado de seguida:

- Obter a dimensão da janela de transmissão:

```
int get_send_window();
```

- Obter a número máximo de sequência:

```
int get_max_sequence();
```

- Obter o valor do timeout:

```
long get_timeout();
```

- Obter o tempo atual de simulação:

```
long get_time();
```

- Enviar a trama `frame` para o nível físico (i.e. para o canal):

```
void to_physical_layer(simulator.Frame frame);
```

- Armar um temporizador associado à chave `key`. Caso seja armado duas vezes com a mesma chave, cancela o relógio anterior:

```
void set_timer(long delay, int key);
```

- Cancelar o temporizador associado à chave `key`:

```
void cancel_timer(int key);
```

- Armar o temporizador de ACKs:

```
void set_ack_timer();
```

- Cancelar o temporizador de ACK:

```
void cancel_ack_timer();
```

- Parar a simulação:

```
void stop();
```

Para permitir uma correta contabilização do número de tramas de dados retransmitidas, deverá ser chamado o método `count_retransmission` cada vez que é retransmitida uma trama:

```
void count_retransmission();
```

3.2.2. Eventos

Pense nos eventos como algo que provoca a invocação dos métodos de *callback*. Estes métodos vão ser implementados pelos alunos na classe *Protocol*, e foram definidos na `interface Callbacks`:

- Evento de início de simulação:

```
void start_simulation(long time);
```

- Evento de fim de transmissão da trama de dados com o nº de sequência `seq`:

```
void handle_Data_end(long time, int seq);
```

- Evento de disparo do temporizador associado à chave `key`:

```
void handle_Timer(long time, int key);
```

- Evento de disparo do temporizador de ACK:

```
void handle_ack_Timer(long time);
```

- Evento de receção da trama `frame` do nível físico:

```
void from_physical_layer(long time, simulator.Frame frame);
```

- Evento de fim de simulação:

```
void end_simulation(long time);
```

Em todos os eventos, é recebido o tempo de simulação atual, em `time`.

3.2.3. Nível rede

A classe `terminal.NetworkLayer`, define os métodos de troca de pacotes com o nível Rede, e está instanciada no objeto `net`:

- Obter um novo pacote do nível rede (caso já não exista mais nenhum para enviar, retorna `null`):

```
String from_network_layer();
```

- Enviar um novo pacote para o nível rede (caso exista um erro no conteúdo devolve `false`):

```
boolean to_network_layer(String packet);
```

3.2.4. Geração e leitura de Tramas

As tramas são objetos da classe `simulator.Frame (import Simulator.Frame)`. Esta classe contém os campos já explicados anteriormente (seq, ack, info, etc.) mais um chamado `kind`. Para além disso, tem métodos estáticos para criar novas instâncias de objetos, e métodos normais para aceder aos vários campos. O campo `kind` define o tipo de trama tendo três valores para uma trama válida: `Frame.DATA_FRAME`, `Frame.ACK_FRAME` ou `Frame.NACK_FRAME`; e o valor `Frame.UNDEFINED_FRAME`, quando ela não está inicializada.

Para obter o valor de `kind`, pode-se usar o método `kind()`.

Para criar uma nova trama de cada um dos três tipos é possível usar os métodos:

```
public static Frame new_Data_Frame(int seq, int ack, String info);
public static Frame new_Ack_Frame(int ack);
public static Frame new_Nack_Frame(int nack);
```

Para aceder aos campos usam-se os métodos:

```
public int seq();       // for DATA_FRAME
public String info();   // for DATA_FRAME
public int ack();       // for DATA_FRAME, ACK_FRAME, NACK_FRAME
public long snd_time(); // tempo em que foi enviado
```

Também é possível obter uma descrição textual do conteúdo da trama:

```
public String kindString();   // devolve o nome do tipo de trama
public String toString();     // devolve o nome do tipo e o resumo do conteúdo
```

Para permitir o transporte das tramas através de sockets TCP, foram definidas duas funções para converter o conteúdo para string e restaurar o conteúdo a partir de uma string (i.e. fazer a serialização do objeto). Estas funções não vão ser usadas pelos alunos, mas geram a representação que é mostrada no log da aplicação.

```
public String frame_to_str();
public boolean str_to_frame(String line, Log log);
```

3.2.5. Protocolo Utópico

A classe *Protocol1* foi programada inteiramente para servir de exemplo para os alunos. Ela realiza o protocolo utópico descrito anteriormente. A classe acrescenta (em relação à `interface Callbacks`) duas variáveis de estado, dois métodos para os números de sequência e mais um método para centralizar o envio de tramas para o nível físico.

As variáveis de estado servem para controlar os números de sequência usados nas tramas:

```
private int next_frame_to_send; // Número da próxima trama de dados a enviar
private int frame_expected;     // Número esperado na próxima trama a receber
```

Os dois métodos servem para incrementar um número de sequência de forma circular, que considera o número máximo de sequência usado na simulação:

```
int incr_seq(int n);
int decr_seq(int n);
```

O método de envio de tramas é privado e é o seguinte:

```
private boolean send_next_data_packet() {
  String packet= net.from_network_layer();   // Get a packet from the network layer
  if (packet != null) {
    next_frame_to_send= incr_seq(next_frame_to_send);  // Increment the seq. number
    // Create a new Frame object
```

```
      Frame frame = Frame.new_Data_Frame(next_frame_to_send, frame_expected, packet);
      sim.to_physical_layer(frame);    // Send the frame to the physical layer
      return true;
    }
    return false;  // Failed; no more packets to send
  }
```

Este método é chamado:

- No arranque da simulação:

```
  public void start_simulation(long time) {
    sim.Log("\nUtopian Protocol\n\n");
    send_next_data_packet();    // Start sending the first data frame
  }
```

- Cada vez que termina o envio da trama de dados anterior:

```
  public void handle_Data_end(long time, int seq) {
    send_next_data_packet();    // Send the next data frame
  }
```

A receção de tramas é feita no método `from_physical_layer`, que neste caso faz um pouco mais do que a versão original do livro – verifica se o número de sequência é o esperado, e avança o número esperado para a próxima trama.

```
  public void from_physical_layer(long time, Frame frame) {
    sim.Log(time + " protocol1 received: " + frame.toString() +"\n");
    if (frame.kind() == Frame.DATA_FRAME) {     // Check the frame kind
      if (frame.seq() == frame_expected) {    // Check the sequence number
        net.to_network_layer(frame.info()); // Send the frame to the network layer
        frame_expected = incr_seq(frame_expected);
      }
    }
  }
```

Os restantes métodos da interface `Callbacks` não são usados neste protocolo e limitam-se escrever que foram invocados.

3.3. METAS

Uma sequência para o desenvolvimento do trabalho poderá ser:

1. Programar o protocolo *Stop&Wait* na classe `Protocol2`. Comece por estudar a realização do protocolo utópico, na classe `Protocol1`, para perceber o que pode reutilizar;
2. Programar o protocolo *Go-Back-N* com um único temporizador na classe `Protocol3`. Comece por estudar a realização que fez do protocolo *Stop&Wait*, na classe `Protocol2`, para perceber o que pode reutilizar;
3. Completar a programação do protocolo *Go-Back-N*, introduzindo a utilização de vários temporizadores em paralelo para as diferentes tramas e o temporizador de ACK na classe `Protocol3` (faça *backup* da primeira versão);
4. Programar o protocolo *Go-Back-N* com Nacks na classe `Protocol4`, partindo da classe `Protocol3`.

TODOS os alunos devem tentar concluir **pelo menos a fase 2**. Na primeira semana do trabalho é feita uma introdução geral do trabalho, devendo-se concluir a fase 1. No fim da segunda semana devem estar perto de concluir a fase 2. No fim da terceira semana deve ter terminado a fase 3. No fim da quarta e última semana devem tentar realizar o máximo de fases possível, tendo sempre em conta que é preferível fazer menos e bem (a funcionar e sem erros), do que tudo e nada funcionar.

POSTURA DOS ALUNOS

Cada grupo deve ter em consideração o seguinte:

- Não perca tempo com a estética de entrada e saída de dados
- Programe de acordo com os princípios gerais de uma boa codificação (utilização de indentação, apresentação de comentários, uso de variáveis com nomes conformes às suas funções...) e
- Proceda de modo a que o trabalho a fazer fique equitativamente distribuído pelos dois membros do grupo.