U.C. Sistemas em Rede

Tópico 3

Nível Data Link

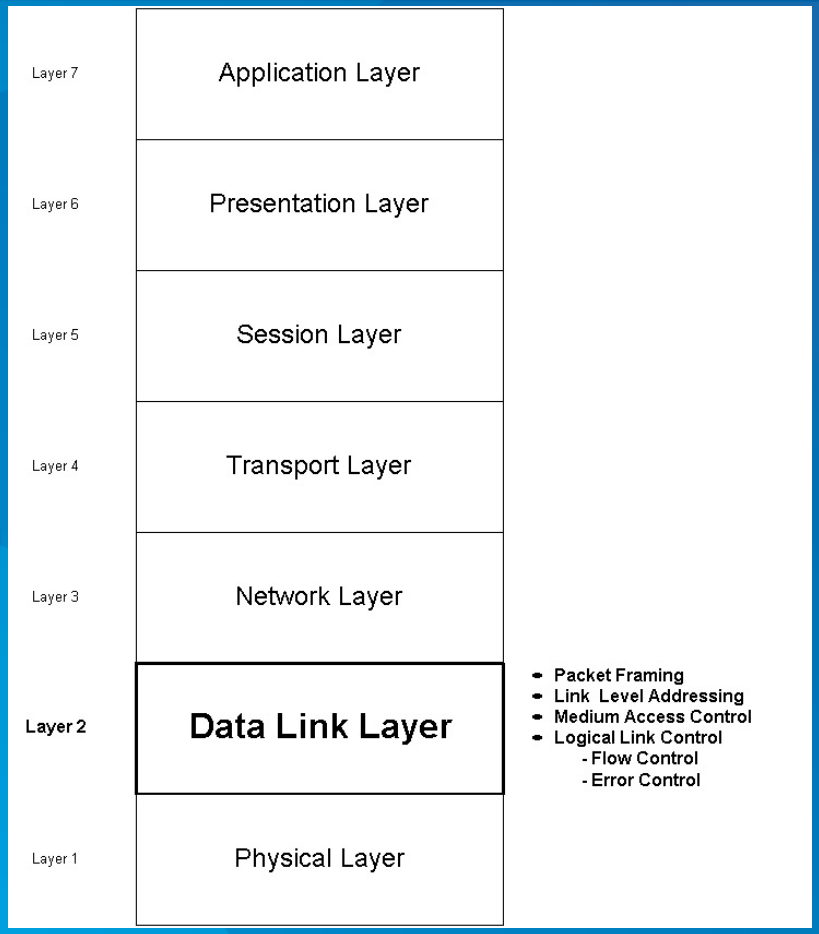
(Camada de Enlace de Dados)

Henrique S. Mamede

Sumário

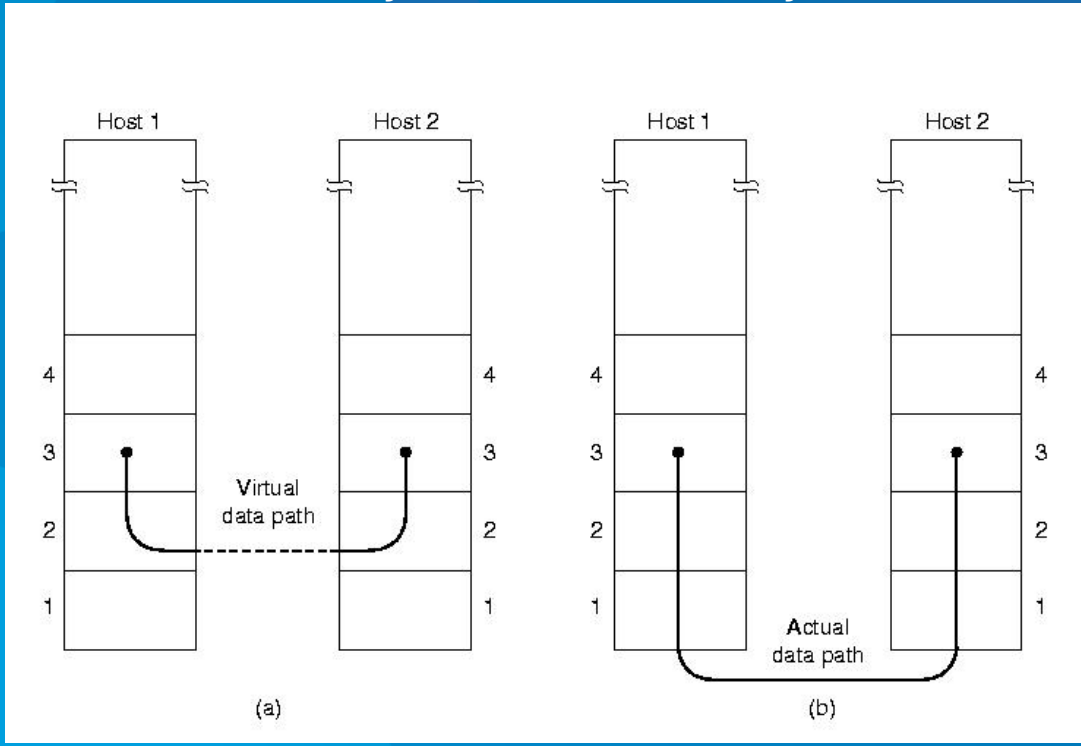
- O Desenho do Nível Data Link
- Detecção e Correcção de Erros
- Protocolos do Nível
- Protocolos de Janela Deslizante
- Especificação e Verificação de Protocolos

Onde Estamos?



Desenho do Nível Data Link

- Existem vários tipos de serviços que podem ser fornecidos aos níveis de Rede.
- A figura recorda a diferença entre comunicações reais e virtuais entre níveis.



Serviços para o nível de Rede

- Mecanismos de Entrega

	UN-Acknowledged	Acknowledged
Connection-Less	“Best Effort”	Better Quality
Connection Oriented		Reliable Delivery

Serviços para o nível de Rede

- Serviço *connectionless* sem confirmação – Melhor Esforço
  - O receptor não devolve confirmações ao emissor, que fica sem forma de saber se dada *frame* foi entregue com sucesso. Quando é apropriado um tal serviço?
    - Quando níveis superiores podem recuperar de erros com pouca perda de desempenho. Quando os erros são tão pouco frequentes que não existe ganho no nível de data link a executar a recuperação. É tão simples como ter níveis superiores a lidar com perdas ocasionais de pacotes.
    - Para aplicações em tempo-real exigindo semânticas tipo “antes nunca que tarde”. Dados atrasados pode ser pior que ausência de dados. P.e., deve um avião calcular o ângulo adequado de flaps das asas com dados atrasados de altitude e velocidade do vento quando dados mais actuais já estão disponíveis?

Serviços para o nível de Rede

- Serviço *connectionless* com confirmação – Entrega Confirmada
  - O receptor devolve uma *frame* de confirmação ao emissor indicando que uma *frame* de dados foi recebida. O emissor mantém o estado da ligação, mas não transmite necessariamente frames não confirmadas.
  - Da mesma forma, o receptor pode enviar *frames* recebidas para níveis superiores na ordem em que chegam, independentemente da ordem original de envio.
  - Tipicamente, a cada *frame* é atribuído 1 nº único sequencial, que o receptor devolve numa *frame* de confirmação para indicar a que *frame* se refere a confirmação. O emissor deve retransmitir *frames* não confirmadas (i.e., perdidas ou danificadas).

Serviços para o nível de Rede

- Serviço *connection-oriented* com confirmação – Entrega Fiável
  - As frames são entregues ao receptor de forma fiável e na mesma ordem em que foram originalmente geradas.
  - O estado da ligação antém informação da ordem de envio e sobre que frames necessitam de retransmissão. P.e., estado do receptor inclui que frames foram recebidas, quais não o foram, etc.

Framing

- O nível de Data Link traduz o fluxo de bits do nível físico em unidades discretas (mensagens) denominadas *frames*. Como pode a *frame* ser transmitida de forma a que o receptor possa detectar as fronteiras de cada *frame*? Ou seja, como pode o receptor reconhecer o início e o fim de cada *frame*? Discutimos 4 formas:
  - Contagem de Caracteres
  - Bit Stuffing
  - Character Stuffing
  - Encoding Violations

Framing - Contagem de Caracteres

- Primeiro campo no cabeçalho do *frame* indica comprimento do mesmo. Assim receptor sabe dimensão do *frame* e pode determinar onde termina o mesmo.
- Desvantagem: receptor perde sincronização quando os bits ficam baralhados. Se os bits ficarem corrompidos durante a transmissão, o receptor pode considerar que a *frame* contém menos (ou mais) bits do que a realidade.
- Embora o *checksum* determine as *frames* incorrectas, o receptor terá dificuldade em re-sincronizar para o início de uma nova *frame*. Esta técnica não é actualmente utilizada por existirem outras

Framing - Bit Stuffing

- A ideia é usar padrões de bits reservados para indicar o início e fim da *frame*. P.e., utilizar a sequência de 4 bits 0111 para delimitar *frames* consecutivas. A *frame* consistirá de tudo o que estiver entre 2 delimitadores.
- **Problema:** o que sucede se o delimitador reservado por acaso aparecer na própria frame? Se esse padrão não for removido dos dados, o receptor pensará que a frame são 2 frames mais pequenas!
- **Solução:** Usar bit stuffing. Dentro da *frame*, substituír cada ocorrência de 2 bits 1's consecutivos por 110. Ou seja, adicionar um bit zero após cada par de 1's nos dados. Tal evita 3 bits 1's a aparecerem no interior da *frame*.

Framing - Bit Stuffing (2)

- O receptor converte dois bits 1 consecutivos colocando um bit 0 imediatamente a seguir, reconhecendo a sequência 0111 como o fim da *frame*.
- Exemplo: A *frame* "1 0 1 1 1 0 1" será transmitida sobre o nível físico como "0 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1".
- Nota: Utilizando esta técnica, localizar o início/fim da *frame* é fácil, mesmo quando as *frames* estão danificadas.
- O receptor re-sincronizará rapidamente com o emissor relativamente ao início/fim das *frames*, mesmo quando os bits no interior da mesma ficam corrompidos.
- A principal desvantagem com este técnica é a inserção de bits adicionais no fluxo de dados, desperdiçando largura de banda.

Framing - Character Stuffing

- Mesma ideia que bit stuffing, mas operando em bytes ao invés de bits.
- Usa caracteres reservados para indicar início/fim da *frame*. P.e., usar a sequência de caracteres DLE STX (Data-Link Escape, Start of TeXt) para sinalizar o início da *frame*, e a sequência DLE ETX (End of TeXt) para indicar o fim da *frame*.

- **Problema**: O que sucede se a sequência de caracteres ocorrer na *frame*?
- **Solução**: Usar character stuffing dentro da *frame*, substituíndo cada ocorrência de DLE com a sequência DLE DLE. O receptor só tem de reverter o processo, substituíndo cada ocorrência de DLE DLE por um único DLE.

Framing - Character Stuffing (2)

- Exemplo: se a frame contiver "A B DLE D E DLE", os caracteres transmitidos no canal serão "DLE STX A B DLE DLE D E DLE DLE DLE ETX".

- Desvantagem: um octeto é a mais pequena unidade que pode ser operada; nem todas as arquitecturas são orientadas a 8-bits.

Framing - Encoding Violations

- Envio de sinal que não é conforme com qualquer representação legal de bits. Na codificação Manchester, p.e., bits 1 são representados por uma sequência high-low, e bits 0 por sequências low-high. O início/fim de uma *frame* pode ser representada pelo sinal low-low ou high-high.
- A vantagem das encoding violations é que não é exigida largura de banda adicional. O standard IEEE 802.4 usa esta aproximação.
- Por fim, alguns sistemas usam uma combinação destas técnicas. IEEE 802.3, p.e., tem quer um campo de comprimento da *frame*, quer padrões de início/fim da *frame*.

Controlo de Erros

- Procura assegurar que todas as *frames* são eventualmente entregues (possivelmente por ordem) no destinatário. São necessários três componentes para realizar tal tarefa:
  - **Confirmações**
  - **Temporizadores**
  - **Números de Sequência**

Controlo de Erros - Confirmações

- Entrega fiável é alcançado usando o paradigma das “confirmações com retransmissão”.
- O receptor devolve uma frame de confirmação (ACK) ao emissor indicando a recepção correcta de uma frame.
- Em alguns sistemas, o receptor também devolve uma confirmação negativa (NACK) para frames incorrectamente recebidas.
- Tal é apenas uma indicação ao emissor de forma que possa retransmitir uma frame de imediato sem ter de aguardar pela expiração de um temporizador.

Controlo de Erros - Temporizadores

- Um problema em que os esquemas de ACK/NACK falham é na recuperação de uma frame perdida em que, como resultado, falham em solicitar um ACK ou um NACK.
- Que sucede se um ACK ou NACK se perde?
  - São usados temporizadores de retransmissão para reenviar frames que não produzem um ACK. No envio de uma frame, é inicializado um temporizador para expirar algum tempo após a devolução esperada do ACK. Se o temporizador finalizar, retransmitir a frame.

Controlo de Erros – Números de Sequência

- Retransmissão introduz a possibilidade de frames duplicadas.
- Para suprimir duplicados, adicionar números de sequência a cada frame, para que o receptor possa distinguir entre frames novas e repetições de frames antigas.

Controlo de Fluxo

- Lida com o controlo de velocidade do transmissor de forma a igualar a velocidade do receptor. Tipicamente é um processo dinâmico, já que a velocidade de recepção depende de factores como a carga e disponibilidade de espaço de buffer.
- Uma solução consiste em o receptor dar créditos ao transmissor. Por cada crédito, o transmissor pode enviar uma frame. Desta forma, o receptor controla a taxa de transmissão pela entrega dos créditos.
- Inicialização do Link:
  - Em alguns casos o serviço da camada data link deve ser "aberto" antes de ser utilizado:
    - A camada data link usa operações abertas para atribuição de espaço de buffer, controlo de blocos, acordo na máxima dimensão de mensagem, etc.
    - Sincronizar e inicializar números de sequência de envio e recepção com o respectivo par no outro extremo do canal de comunicação

Detecção e Controlo de Erros

- Esta secção lida com as questões de redundância necessárias à detecção e correcção de erros nos dados.
- Serão abordadas as questões de desenho da camada de data link (DLL), protocolos, incluíndo os de janela deslizante e a respetiva especificação e verificação.

Detecção e Controlo de Erros

- Na comunicação de dados, o ruído é um facto da vida. Mais, o ruído usualmente ocorre em rajadas e não na forma de erros de bit únicos.
- Detectar e corrigir erros exige redundância – envio de informação adicional juntamente com os dados.
- Existem duas formas de abordar os erros:
  - **Códigos de Detecção de Erros:** Inclui bits de redundância suficientes para detectar erros e usar ACKs e retrnsmissões para recuperar dos mesmos.
  - **Códigos de Correcção de Erros:** Inclui redundância bastante para detectar e corrigir erros.
- Introduziremos alguns conceitos e depois abordaremos quer a detecção quer a correcção.
- Para compreender os erros, considerar o seguinte:
  - Mensagens (frames) consistem de **m** bits (mensagem) de dados e **r** bits de redundância, originando uma palavra com **n = (m + r)** bit

Detecção e Controlo de Erros

- **Distância de Hamming**. Dadas 2 palavras, podemos determinar quantos dos bits diferem. Basta fazer XOR às 2 palavras e contar a quantidade de bits 1 no resultado. A contagem é a distância de Hamming.
- Significado? Se 2 palavras estão distantes **d** bits, **d** erros são necessários para converter uma na outra.
- Um código de distância de Hamming é definido como a distância Hamming mínima entre 2 palavras de todas as possíveis.
- Em geral, todas as possíveis $\mathbf{2^m}$ palavras são legais. No entanto, escolhendo bits de check cuidadosamente, as palavras resultantes terão uma distância de Hamming maior. Quanto maior a distância de Hamming, melhor os códigos são capazes de detectar erros.
- Para detectar **d** 1-bit erros é necessária uma distância de Hamming de pelo menos **d + 1** bits. Porquê?
  - Para corrigir **d** erros é necessário **2d + 1** bits. Intuitivamente, depois de **d** erros, a palavra errada estará ainda mais próxima da mensagem original que qualquer outra palavra.

Detecção e Controlo de Erros

- **Bits de Paridade**
- Um único bit de paridade é adicionado a cada bloco de dados (cada caractere nos sistemas ASCII) de forma que o número de bits 1 seja sempre par.
  Exemplos:

  1000000(1) 1111101(0)
- A Distânca de Hammimg para a paridade é 2, e não consegue corrigir nem erros de apenas um bit, apesar de os conseguir detectar. Como exemplo, considerar um código de 10 bits utilizado para representar quatro possíveis valores:

  "00000 00000", "00000 11111", "11111 00000", and "11111 11111".

  A respectiva Distância de Hamming é 5, e conseguimos corrigir 2 erros de bit único:

  Por exemplo, "10111 00010" torna-se "11111 00000" mudando apenas 2 bits.

  No entanto, se o transmissor envia "11111 00000" e o receptor vê "00011 00000", o receptor não conseguirá corrigir o erro adequadamente.
- Por fim, neste exemplo temos a garantia de apanhar todos os erros de 2 bits, mas podemos fazer melhor: se "00111 00111" contém 4 erros de um único bit, conseguimos reconstruír correctamente o bloco.

Detecção e Controlo de Erros

- Qual é o menor nº de bits necessários para corrigir erros de bit único? Vejamos um código contendo **n = m +r** bits que corrige todos os erros de bit único (recordar que **m** é o número de data bits (mensagem) e **r** é o número de bits redundantes) :
- Existem **2m** mensagens legais (i.e., padrões legais de bits).
- Cada uma das m mensagens tem n palavras ilegais a uma distância 1 de si. Ou seja, se sistematicamente invertermos cada bit na palavra de n-bit correspondente, obtemos palavras ilegais à distância 1 da original. Assim, cada mensagem exige n + 1 bits dedicados (n que estão à distância de 1 bit e 1 que é a mensagem).
- O número total de padrões de bits é **(n + 1) * 2m ≤ 2n**. Ou seja, todas as **(n+1) * 2m** mensagens codificadas devem ser única, e não podem existir menos mensagens que as **2n** palavras possíveis.
- Uma vez que **n = m + r** , obtemos:

  (m + r + 1) * 2m ≤ 2m+r ou

  (m + r + 1) ≤ 2r
- Esta fórmula dá o limite inferior absoluto relativo ao número de bits necessários para detectar (e corrigir) erros de 1-bit.

Detecção e Controlo de Erros

- A correcção de erros é relativamente dispendiosa, quer computacionalmente quer em largura de banda.
- Por exemplo, 10 bits de redundância são necessários para corrigir 1 erro de single-bit numa mensagem de 1000-bit. Em contraste, detectar um erro de single bit exige apenas um single-bit, independentemente da dimensão da mensagem. Os códigos de detecção de erros mais populares são baseados em códigos polinomiais ou de redundância cíclica (***c****yclic* ***r****edundancy* ***c****odes* - CRCs).
- Permitem-nos confirmar correctamente frames recebidas e descartar frames incorrectas.
- No livro existem vários exemplos!!

Protocolos Data Link Layer (DLL)

- **Como podem duas camadas DLL comunicar de forma a assegurar confiabilidade?**
- **Veremos alguns protocolos complexos que demonstram como tal é feito.**

Protocolos Data Link Layer (DLL)

Protocolos DLL Elementares

- A DLL fornece os seguintes serviços à camada de Rede (Network Layer – NL):
  - Data entregue a DLL pela NL num módulo, é entregue à NL em outro módulo por aquela DLL.
  - O par remoto NL deve receber a mensagem idêntica gerada pelo emissor (i.e., se a DLL adiciona informação de controlo, o cabeçalho deve ser removidoantes da mensagem ser passada à NL).
  - A NL pode querer assegurar que todas as mensagens que envia são correctamente entregues (sem perdas nem corrupção). Note-se que erros arbitrários podem resultar na perda quer das frames de dados quer das frames de controlo.
  - A NL pode querer que as mensagens sejam entregues ao par remoto exactamente na mesma ordem em que são enviadas.

  - Nota: Nem sempre é claro que queremos que os protocolos de DLL forneçam este tipo de serviço. Contudo, o modelo de referência sugere que a DLL disponibilize tal serviço.

Protocolos Data Link Layer (DLL)

- Método a Utilizar:
  - Descrever sucessivos protocolos de complexidade crescente para fornecer, por ordem, entrega confiável à camada de rede.
- Ambiente:
  - Assumir que DLL se executa como um processo com rotinas para comunicar com a NL acima e a camada física abaixo.
  - **Frames** são a unidade de transmissão. Consistem de data mais bits de controlo (informação de cabeçalho).
  - Ver as estruturas de dados e protótipos nos próximos slides.
  - De especial interesse é:
    - *typedef struct frame;*
    - *void wait_for_event(event_type *event);*
  - *wait_for_event()* suspende o processo até que ocorra um evento. Eventos possíveis incluem pedidos da NL, da camada física ou do timer.

Protocolos Data Link Layer (DLL)

```
#define MAX PKT 1024                                  /* determines packet size in bytes */
typedef enum {false, true} boolean;                   /* boolean type */
typedef unsigned int seq_nr;                          /* sequence or ack numbers */

typedef struct {
    unsigned char data[MAX PKT];
} packet;                                             /* packet definition */
typedef enum {data, ack, nak} frame_kind;             /* frame kind definition */

typedef struct {                                      /* frames are transported in this layer */
    frame_kind kind;                                  /* what kind of a frame is it? */
    seq_nr seq;                                       /* sequence number */
    seq_nr ack;                                       /* acknowledgement number */
    packet info;                                      /* the network layer packet */
} frame;
```

Protocolos Data Link Layer (DLL)

```
/* 1. Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event );

/* 2. Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer( packet *p);

/* 3. Deliver information from an inbound frame to the network layer. */
void to_network_layer( packet *p);

/* 4. Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer( packet *p);

/* 5. Pass the frame to the physical layer for transmission. */
void to_physical_layer( packet *p);

/* 6. Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* 7. Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* 8. Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* 9. Stop the auxiliary timerand disable the ack_timeout event. */
void stop_ack_timer(void);

/* 10. Allow the network layer to cause a network_layer_event. */
void  enable_network_layer( void );

/* 11. Forbid the network layer from causing a network_layer_event. */
void  disable_network_layer( void );
```

Protocolos Data Link Layer (DLL)

- Suposições
  - Transmissão de dados apenas numa direcção (simplex).
  - Não existem erros no canal físico.
  - O emissor/receptor pode gerar/consumir uma quantidade infinita de dados.
  - Sempre pronto para enviar/receber.
  - Ver o código no próximo slide

Protocolos Data Link Layer (DLL)

```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
sender to receiver. The communication channel is assumed to be error free,
and the receiver is assumed to be able to process all the input infinitely fast.
Consequently, the sender just sits in a loop pumping data out onto the line as
fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender1(void)
{
     frame          s;                  /* buffer for an outbound frame */
     packet         buffer;             /* buffer for an outbound packet */
     while (true) {
          from_network_layer(&buffer);  /* go get something to send */
          s.info = buffer;              /* copy it into s for transmission */
          to_physical_layer(&s);        /* send it on its way */
     }
}
void receiver1(void)
{
     frame          r;
     event_type     event;              /* filled in by wait, but not used here */
     while (true) {
          wait_for_event(&event);       /* only possibility is frame arrival */
          From_physical_layer(&r);      /* go get the inbound frame */
          To_network_layer(&r.info);    /* pass the data to the network layer */
     }
}
```

Protocolos Data Link Layer (DLL)

- Pressupostos:
  - Não se assume que o receptor possa processar dados infinitamente rápido.
  - Emissor envia uma frame e espera pelo *acknowledgment* (stop & wait.)
  - O conteúdo da frame de *acknowledgment* não tem qualquer importância.
  - Transmissão de dados é unidireccional, mas deve ter uma linha bi-direccional. Pode ter um canal físico half-duplex (uma direcção de cada vez).
- Ver o código no próximo slide

Protocolos Data Link Layer (DLL)

```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
sender to receiver. The communication channel is once again assumed to be error
free, as in protocol 1. However, this time, the receiver has only a finite buffer
capacity and a finite processing speed, so the protocol must explicitly prevent
the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender2(void)
{
     frame          s;                  /* buffer for an outbound frame */
     packet         buffer;             /* buffer for an outbound packet */
     event_type     event;              /* frame_arrival is the only possibility */
     while (true) {
          from_network_layer(&buffer);  /* go get something to send */
          s.info = buffer;              /* copy it into s for transmission */
          to_physical_layer(&s);        /* send it on its way */
          wait_for_event(event(&event); /* do not proceed until given the go ahead */
     }
void receiver2(void)
{
     frame          r, s;
     event_type event;                  /* filled in by wait, but not used here */
     while (true) {
          wait_for_event(&event);       /* only possibility is frame arrival */
          From_physical_layer(&r);      /* go get the inbound frame */
          To_network_layer(&r.info);    /* pass the data to the network layer */
          to_physical_layers);          /* send a dummy frame to awaken sender */
     }
}
```

Protocolos Data Link Layer (DLL)

- Protocolo simplex para canal com ruído
  - Pressupostos:
    - O canal tem ruído pode haver perda de frames (nunca chegam)
    - Aprox. simples consiste em adicionar um time-out ao emissor de forma que se nenhum ACK após certo período de tempo, tem de retransmitir a frame
    - Cenário de um bug que pode ocorrer:
      1. ***A*** transmite frame 1
      2. ***B*** recebe ***A1***
      3. ***B*** gera ACK
      4. ACK é perdido
      5. ***A*** times-out, retransmite
      6. ***B*** obtém cópia duplicada de ***A1*** (que envia para camada de rede)
    - Usar número de sequência. Quantos bits? 1-bit é suficiente para este caso simples porque se preocupa apenas com duas frames sucessivas
    - Ack positivo com retransmissão (PAR): emissor espera por ack positivo antes de avançar para o próximo item

Protocolos Data Link Layer (DLL) - Simplex

```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */
#define MAX_SEQ 1                          /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout } event_type;
#include "protocol.h“

void sender3(void)
{
     seq_nr          next_frame_to_send;          /* Seq number of next outgoing frame */
     frame           s;                           /* buffer for an outbound frame */
     packet          buffer;                      /* buffer for an outbound packet */
     event_type      event;                       /* frame_arrival is the only possibility */
     next_frame_to_send = 0;
     from_network_layer(&buffer);                 /* go get something to send */

     while (true) {
         s.info = buffer;                         /* copy it into s for transmission */
         s.seq  = next_frame_to_send;             /* insert sequence number in frame */
         to_physical_layer(&s);                   /* send it on its way */
         start_timer( s.seq);                     /* if answer takes too long, time out */
         wait_for_event(event(&event);            /* frame arrival or cksum err, or timeout */
         if ( event == frame_arrival) {
             from_physical_layers(&s);            /* Get the ACK */
             if ( s.ack == next_frame_to_send ) {
                 from_network_layer( &buffer );   /* get the next one to send */
                 inc( next_frame_to_send );       /* invert next_frame_to_send */
             }
         }
     }
}
```

Protocolos Data Link Layer (DLL) - Simplex

```
void receiver3(void)
{
     seq_nr          frame_expected;
     frame           r, s;
     event_type      event;
     while (true) {
         wait_for_event(&event);                    /* only possibility is frame arrival */
         if ( frame == event_arrival ) {            /* A valid frame has arrived */
             from_physical_layer(&r);               /* go get the inbound frame */
             if ( r.seq == frame_expected ) {       /* This is what we've been waiting for */
                 to_network_layer(&r.info);         /* pass the data to the network layer */
                 inc(frame_expected);               /* next time expect the other seq # */
             }
          s.ack = 1 – frame_expected;
         to_physical_layer(&s);                     /* send a dummy frame to awaken sender */
         }
     }
}
```

Protocolos Data Link Layer (DLL) - Simplex

- Um problema não resolvido por este protocolo:
  - Qual o tempo certo para o timer?
  - E se for demasiado? (ineficiente)
  - E se for curto? Problema porque o ACK não contém o nº de sequência da frame que está a ser ACK'd. Assim, que frame está a ser ACK'd?

- Cenário
  1. A envia frame A0
  2. time out de A
  3. Reenvio de frame A0
  4. B recebe A0, ACKS
  5. B recebe A0 novamente, ACKS novamente
  6. A obtém A0 ACK, envia frame A1
  7. A1 perde-se
  8. A obtém segundo A0 ACK (assume que é ACK de A1), envia A2
  9. B obtém A2 (rejeita, nº de sequência incorrecto)

  - Perdem-se duas frames antes de se conseguir normalizar a situação (com A3)

Protocolos de Janela Deslizante

- Protocolos de Janela Deslizante
  - Estes métodos fornecem um realismo muito maior!
  - O método geral disponibiliza *buffering* com ACKs.

Protocolos de Janela Deslizante

- Pressupostos:
  - Usam comunicação mais realista de 2 vias.
  - Temos 2 tipos de frames (contendo um campo de “tipo”):
    - Dados
    - ACK contendo o nº de sequência da última frame correctamente recebida.
  - **Piggybacking** - adiciona ack a frames de dadosque vão na direcção inversa.
  - Problema com Piggybacking: para melhor uso da largura de banda, quanto tempo se deve esperar pela frame de dados outgoing antes de enviar o ACK.

Protocolos de Janela Deslizante

- Exemplo de um protocolo de janela deslizante:
  - Contém um nº de sequência cujo valor máximo, **MaxSeq,** é **2n - 1**.
  - Para protocolo janela deslizante stop-and-wait, **n** = 1.
- Essencialmente mesmo que Protocolo Simplex, excepto que:
  - ACKs são numeradas, o quie resolve problema time out.
  - Comunciação Two-way.

- Protocolo funciona, todas as frames entregues em ordem correcta
- Exige pouco espaço de buffer.
- Baixa utilização da linha devida a stop-and-wait.

Protocolos de Janela Deslizante

```
/* Protocol 4 (sliding window) is bi-directional and is more robust than protocol 3 */

#define MAX-SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame-arrival, cksum-err, timeout} event-type;
#include "protocol.h"

void protocol4 (void) {
  seq-nr            next-frame-to-send;           /* 0 or 1 only */
  seq-nr            frame-expected;               /* 0 or 1 only */
  frame             r, s;                         /* scratch variables */
  packet            buffer;                       /* current packet being sent */
  event-type        event;
  next-frame-to-send = 0;                         /* next frame on the outbound stream */
  frame-expected = 0;                             /* number of frame arriving frame expect */
  from-network-layer(&buffer);                    /* fetch a packet from the network layer */
  s.info = buffer;                                /* prepare to send the initial frame */
  s.seq  = next-frame-to-send;                    /* insert sequence number into frame */
  s.ack  = 1 -frame-expected;                     /* piggybacked ack */
  to-physical-layer(&s);                          /* transmit the frame */
  start-timer(s.seq);                             /* start the timer running */
```

Protocolos de Janela Deslizante

```
while (true) {
  wait-for-event(&event);                    /* frame-arrival, cksum-err, or timeout */
  if (event == frame-arrival) {              /* a frame has arrived undamaged. */
      from-physical-layer(&r);               /* go get it */
      if (r.seq == frame-expected) {

                                             * Handle inbound frame stream. */
          to-network-layer(&r.info);         /* pass packet to network layer */
          inc(frame-expected);               /* invert sequence number expected next */
      }
      if (r.ack == next-frame-to-send) {     /* handle outbound frame stream. */
          from-network-layer(&buffer);       /* fetch new pkt from network layer */
          inc(next-frame-to-send);           /* invert sender's sequence numbe
      }
  }
  s.info = buffer;                           /* construct outbound frame */
  s.seq = next-frame-to-send;                /* insert sequence number into it */
  s.ack = 1 -frame-expected;                 /* seq number of last received frame */
  to-physical-layer(&s);                     /* transmit a frame */
  start-timer(s.seq);                        /* start the timer running */
 }
}
```

Protocolos de Janela Deslizante

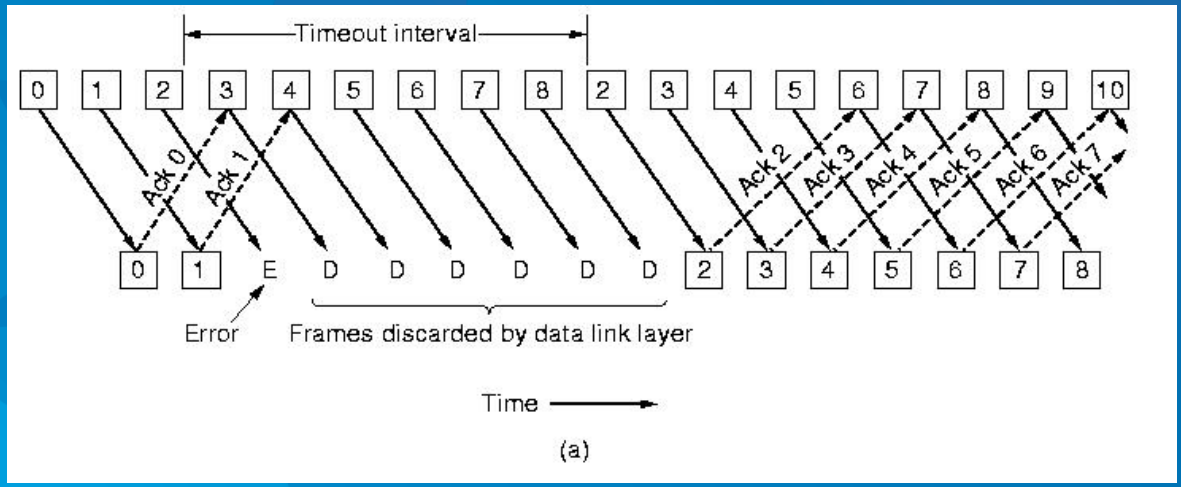
- Problema com protocolos stop-and-wait é que o emissor só tem no exterior uma frame sem ACK
- Exemplo:
  - Frames de 1000 bit
  - Canal de 1 Mbs (satélite)
  - Atraso de propagação de 270 ms
  - Frame demora 1msec (1000 bits/(1,000,000 bits/sec) = 0.001 sec = 1 msec) a enviar. Com atraso de propagação o ACK não visto no emissor até 541msec. Baixa utilização do canal!
  - Várias soluções são possíveis:
    - Utilização de frames maiores, mas com tamanho máximo limitado pela taxa de bit error do canal. Quanto maior a frame, maior a probabilidade de se danificar durante a transmissão
    - Uso de **pipelining**: permitir múltiplas frames simultâneas na transmissão

Protocolos de Janela Deslizante

- (cont.)
  - Emissor não espera para o ACK de cada frame. Envia várias frames com o pressuposto de que chegarão. Deve, no entanto, obter ACKs para cada frame
  - Que sucede se 20 frames transmitidas, e a segunda tem um erro? Frames 3-20 são ignoradas do lado do receptor? Emissor terá de retransmitir. Quais são as possibilidades?
  - Vamos ver 2 estratégias:

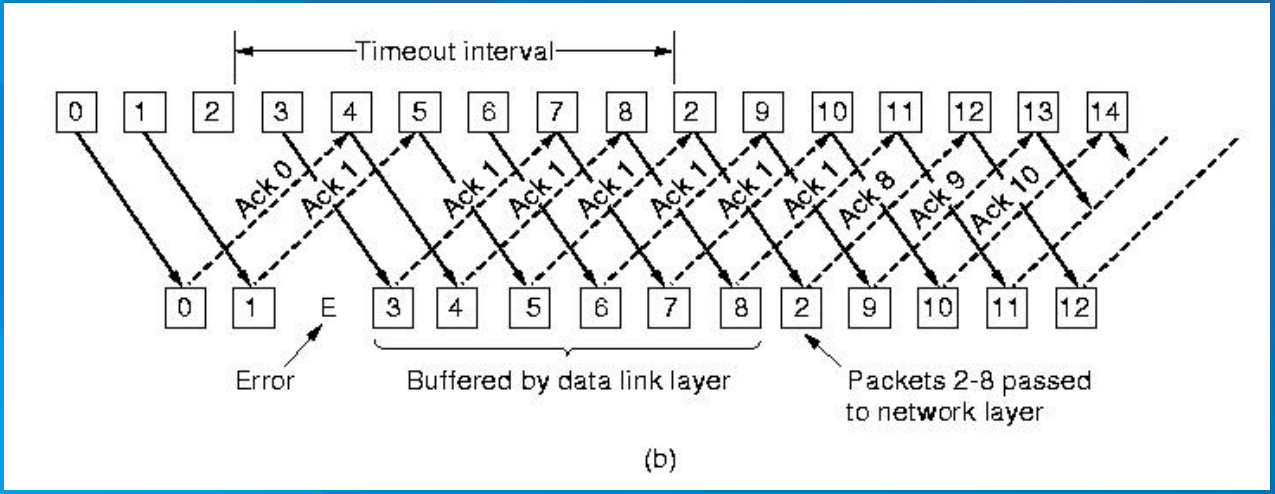
Protocolos de Janela Deslizante

- (cont.)
  - **Go back n** – equivalente a janela de tamanho 1 do receptor. Se receptor vê frames com erros ou nºs de sequência em falta, frames subsequentes são ignoradas. Não há ACKs para frames ignoradas.



Protocolos de Janela Deslizante

- (cont.)
  - **Selective repeat** – dimensão da janela do recpetor maior que 1. Armazena todas as frames recebidas depois da errada. ACK apenas após a última recebida em sequência.



Protocolos de Janela Deslizante

- (Cont.)
  - Compromisso entre largura de banda e espaço de buffer no lado do receptor.
  - Em qualquer caso é necessário espaço de buffer no lado do emissor. Não pode libertar até que o ACK seja recebido.
  - Uso de timer para cada frame unACK'ed enviada.
  - Deve ser capaz de enable/disable camada de rede porque pode não ser capaz de manipular mais dados enviados caso existam muitas frames unACK'd.

- **Regras para Dimensão da Janela**

  - Problema Potencial da dimensão da janela (dimensão janela receptor é 1):
    - **MaxSeq** é 7 (0 a 7) é válido. De que dimensão pode ser a janela do emissor?
    - Envia 0-7.
    - Recebe 0-7 (uma de cada vez) e envia ACKS
    - Todos os ACKS são perdidos
    - Mensagem 0 times out e é retransmitida
    - Receptor aceita frame 0 (porquê? – porque é a frame seguinte) e passa-a à camada de rede.
  - Portanto: – dimensão da janela do emissor deve ser inferior a MaxSeq.

- Veja-se como tudo isto é posto a funcionar nos slides seguintes!

Protocolos de Janela Deslizante

```
/* Protocol5 (pipelining) allows multiple outstanding frames.  The sender may
   transmit up to MAX-SEQ frames without waiting for an ack. In addition, unlike
   the previous protocols, the network layer is not assumed to have a new packet
   all the time. Instead, the network layer causes a network-layer-ready event
   when there is a packet to send. */

#define MAX-SEQ 7                                       /* should be 2^n -1 */
typedef enum {frame-arrival, cksum-err, timeout, network-layer-ready} event-type;
#include "protocol.h"
                                                        /* Return true if (a <=b < c circularly;
                                                            false otherwise. */
static boolean between(seq-nr a, seq-nr b, seq-nr c) {
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
    return(true);
  else
    return(false);
}

static void send-data(seq-nr frame-nr,
     seq-nr frame-expected, packet buffer[]) {
                                                        /* Construct and send a data frame. */
  frame   s;                                            /* scratch variable */
  s.info = buffer[frame-nr];                            /* insert packet into frame */
  s.seq = frame-nr;                                     /* insert sequence number into frame */
  s.ack = (frame-expected + MAX-SEQ) % (MAX-SEQ + 1 );      /* piggyback ack */
  to-physical-layer(&s);                                /* transmit the frame */
  start-timer(frame-nr);                                /* start the timer running */
}
```

Protocolos de Janela Deslizante

```
void protocol5(void) {
  seq-nr  next-frame-to-send;                /* MAX-SEQ > 1; used for outbound stream */
  seq-nr            ack-expected;            /* oldest frame as yet unacknowledged */
  seq-nr            frame-expected;          /* next frame expected on inbound stream */
  frame             r;                       /* scratch variable */
  packet            buffer[MAX-SEQ + 1 ];    /* buffers for the outbound stream */
  seq-nr            nbuffered;               /* # output buffers currently in use */
  seq-nr            i;                       /* used to index into the buffer array */
  event-type        event;

  enable-network-layer();                    /* allow network-layer-ready events */
  ack-expected = 0;                          /* next ack expected inbound */
  next-frame-to-send = 0;                    /* next frame going out */
  frame-expected = 0;                        /* number of frame expected inbound */
  nbuffered = 0;                             /* initially no packets are buffered */
```

Protocolos de Janela Deslizante

```
while (true) {
   wait-for-event(&event);                          /* four possibilities: see event-type */
   switch(event) {
      case network_layer_ready:                     /* the network layer has a packet to send */
                                                    /* Accept, save, and transmit a new frame. */
         from-network_layer(&buffer[next-frame-to-send]);         /* fetch new packet */
         nbuffered = nbuffered + I;                 /* expand the sender's window */
         send-data(next_frame-to-send, frame-expected, buffer);   /* transmit the frame */
         inc(next_frame-to-send);                   /* advance sender's upper window edge */
         break;
      case frame-arrival:                           /* a data or control frame has arrived */
         from_physical_layer(&r);                   /* get incoming frame from physical layer */
         if (r.seq == frame-expected) {
                                                    /* Frames are accepted only in order. */
            to_network-layer(&r.info);              /* pass packet to network layer */
            inc(frame-expected);                    /* advance lower edge of receiver's window */
         }
                                                    /* Ack n implies n- 1, n -2, etc. Check this. */
         while (between(ack-expected, r.ack, next_frame_to_send)) {
                                                    /* Handle piggybacked ack. */
            nbuffered = nbuffered -1;               /* one frame fewer buffered */
            stop-timer(ack-expected);               /* frame arrived intact; stop timer */
            inc(ack-expected);                      /* contract sender's window */
         }
         break;
```

Protocolos de Janela Deslizante

```
      case cksum-err: break;                          /* just ignore bad frames */

      case timeout:                                   /* trouble; retransmit all outstanding frames*/
          next-frame-to-send = ack-expected;          /* start retransmitting here */
          for (i = I; i <= nbuffered; i++){
              send-data(next-frame-to-send, frame-expected, buffer);    /* resend 1 frame */
              inc(next-frame-to-send);                /* prepare to send the next one */
          }
          break;
    }
    if (nbuffered < MAX-SEQ)
      enable_network_layer();
    else
      disable_network_layer();
  }
}
```

Especificação e Verificação de Protocolos

- O problema é antigo!
- Como especificar a operação de um protocolo e depois assegurar que o mesmo está a trabalhar correctamente?

Especificação e Verificação de Protocolos

- Que métodos de representaçãod e especificações existem?

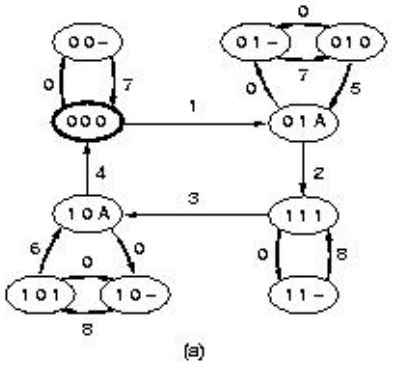
Diagramas de estado são uma forma útil de verificar que um desenho é correcto e completo.

Ver situação do slide 37.

Os estados possíveis para essa configuração são representados por (XYZ) onde:
X = <0|1>: Nº de sequência da frame a ser enviada
Y = <0|1>: Nº de sequência da frame que o receptor espera
Z = <0|1|A|->: Estado do canal; <Seq. 0|Seq. 1|ACK|empty>

(0,0,0) = emissor enviou frame 0, receptor espera 0, e frame 0 está no canal.

Na fig. o diagrama de estados correspondente->



Transition	Who runs?	Frame accepted	Frame emitted	To network layer
		(frame lost)		
0	–			–
1	R	0	A	Yes
2	S	A	1	–
3	R	1	A	Yes
4	S	A	0	–
5	R	0	A	No
6	R	1	A	No
7	S	(timeout)	0	–
8	S	(timeout)	1	–

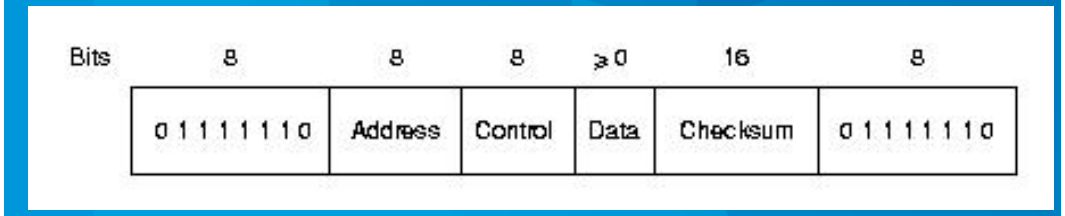
(b)

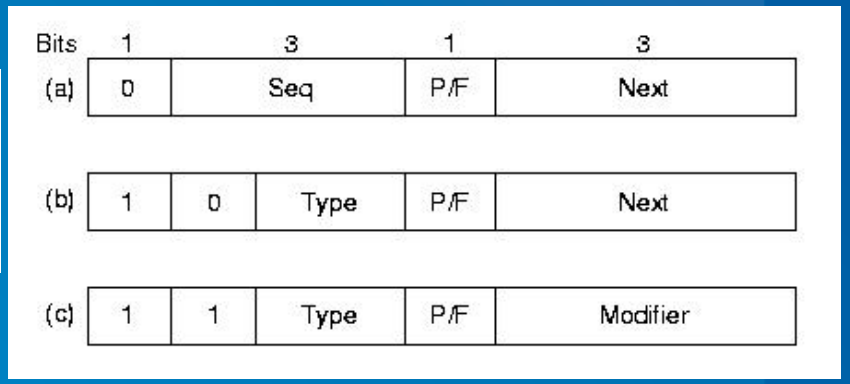
Útil para determinar:
Garantir que algumas transições não são possíveis.
Garantir que não há deadlocks
(cada estado tem uma transição para outro).

Especificação e Verificação de Protocolos - Exemplos

HDLC - HIGH LEVEL DATA LINK CONTROL

Adoptado como parte do X.25.
Rede orientada à ligação de 64Kbps utilizando quer circuitos virtuais quer permanetes.
Bit oriented (usa bit stuffing e bit delimitadores).
Nºs de sequência com 3-bit.
Até 7 frames unACK'ed podem existir em qualquer momento.
ACK's o "frame esperado" ao invés da última frame recebida.
Repare-se na informação de controlo nas duas figuras.





Especificação e Verificação de Protocolos - Exemplos

Linhas ponto-a-ponto:
Entre routers em leased lines
Dial-up a um host através de modem

PPP - Point-to-Point Protocol

Um Standard (RFCs 1661-1663)

Pode ser usado para linhas dial-up e leased router-router.

Fornece:

- Método para delinear frames. Também trata da detecção de erros.
- Link Control Protocol (LCP) para activar linhas, negociação de opções, desactivar linhas. Estes são pacotes PPP distintos.
- Network Control Protocol (NCP) para negociar opções da camada de rede.
- Similar ao HDLC, mas *character-oriented*.
- PPP não fornece transferência fiável de dados usando nºs de sequência e acks por defeito. Transferência de dados fiável pode ser pedida como opção (como parte do LCP).
- Permite a um provider internet reutilizar endereços IP. Permite a utilização de um endereço para a duração do respectivo login.

Especificação e Verificação de Protocolos - Exemplos

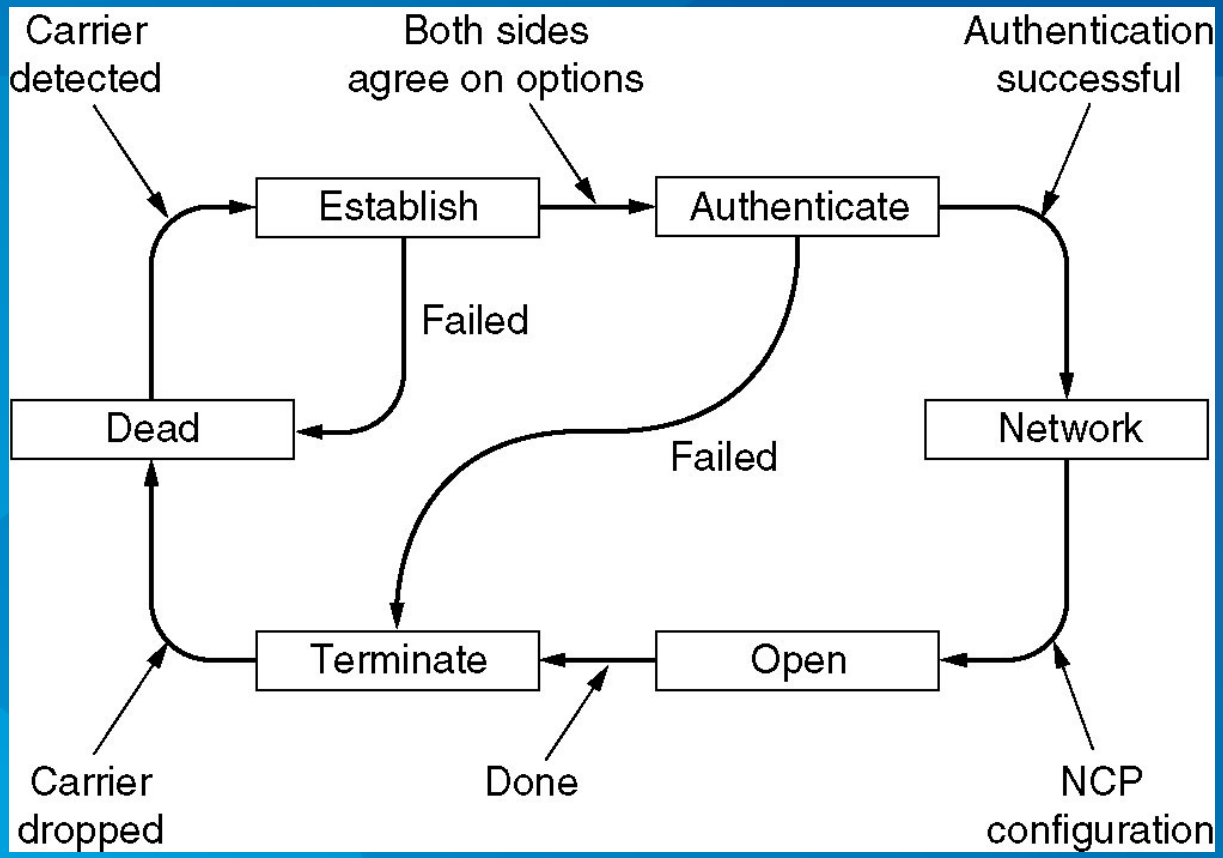


Diagrama de Activação/Desactivação de linha