شبکه های کامپیوتری

موسسه آموزش عالی کرمان

امین داستانپور


جمهوری اسلامی ایران
وزارت علوم، تحقیقات و فناوری

شماره : ۹۶۱۲
تاریخ : ۹۶/۱۲/۱۴
پیوست :

مؤسسه آموزش عالی کرمان - کرمان
(غیردولتی - غیرانتفاعی)

جناب آقای دکتر امین داستانپور

باسلام

نظر به تخصص و تجارب حضرتعالی بعنوان معاون پژوهش و ارتباط با صنعت مؤسسه آموزش عالی کرمان منصوب میشوید امید است در پرتو عنایات خداوند متعال در ارتقاء کمی و کیفی پژوهش مؤسسه مؤید و موفق بوده باشید.

دکتر محمدجواد خانجانی
معاونت آموزشی

وزارت علوم، تحقیقات و فناوری
مؤسسه آموزش عالی غیردولتی
غیرانتفاعی کرمان ۱۳۸۳

- کارشناسی نرم افزار کامپیوتر از دانشگاه فنی کرمان
- کارشناسی ارشد شبکه های کامپیوتری از دانشگاه **UPM**
- دکترای امنیت شبکه در فناوری اطلاعات از دانشگاه **UTM**

- Website : http://amindastanpour.webs.com
- Email : dr.amin.dastanpour@gmail.com

ارزیابی دانشجو در کلاس

- ۸ نمره میان ترم (یکشنبه ۲۷ آبان ۱۳۹۷) ساعت کلاسی
- ۱۲ نمره پایان ترم
- نمرات ارفاقی
- ۱- حضور غیاب (۱ نمره)
- ۲- پرسش و پاسخ در کلاس (۱ نمره)
- ۳- امتحان در کلاس (۱ نمره)
- ۴- پروژه (۳ نمره)

- جمعا ۶ نمره ارفاقی

پروژه (۳ نمره)

۱. ارائه کتبی (۱ نمره)
۲. ارائه شفاهی (۱ نمره)
۳. پرسش و پاسخ (۱ نمره)

- جمعا ۳ نمره ارفاقی

- موضوعات پروژه:

۱. پروتکل های شبکه
۲. برنامه های تحت شبکه
۳. امنیت شبکه با دیواره آتش
۴. امنیت شبکه با تشخیص نفوذ

ارائه کتبی

- عنوان
- یک پاراگراف چکیده
- یک صفحه مقدمه
- یک صفحه تاریخچه و آنچه که قبلا انجام شده تا به امروز
- دو صفحه روش کار کرد سیستم
- یک صفحه نتایج گرفته شده از این روش
- یک پاراگراف خلاصه مطلب
- منابع

ارائه شفاهی

- ۱۰ دقیقه ارائه از ذهن ارائه دهنده و به هیچ عنوان رو خوانی نشود
- ۵ دقیقه پرسش و پاسخ

جلسه سوم

لایه پیوند داده

ملاحظات طراحی لایه پیوند داده

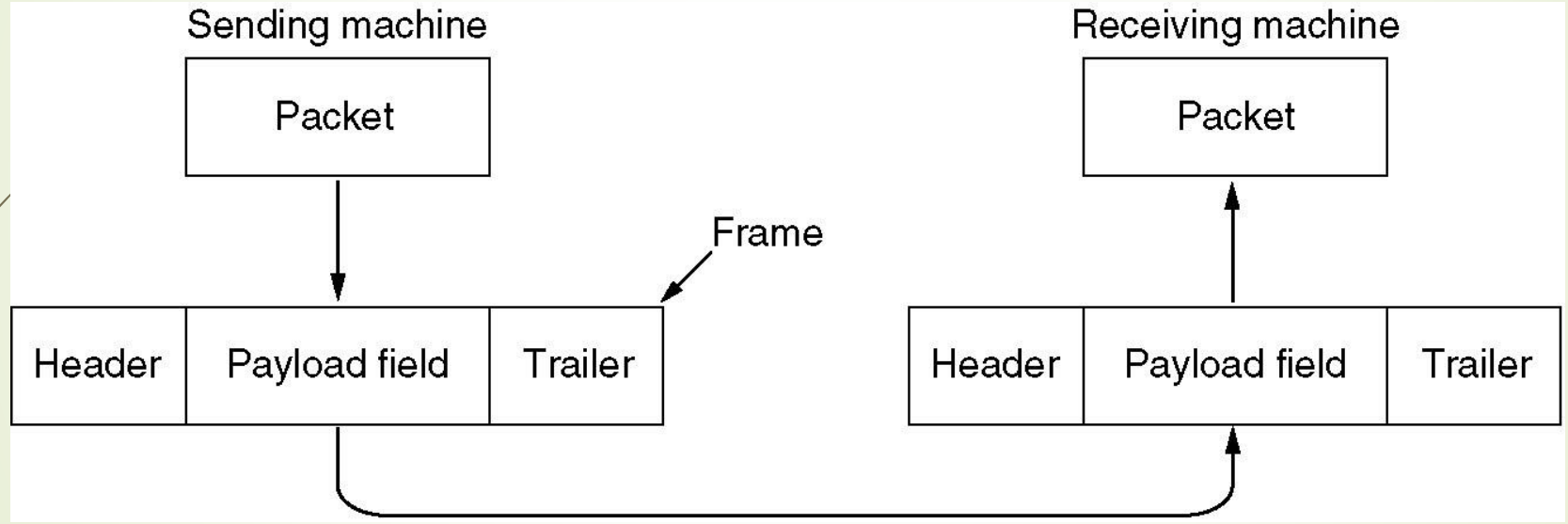
- ارایه سرویس های مشخص به لایه شبکه
- فریم بندی

۱. داده اصلی
۲. سرآیند
۳. پی آیند

کلیدی ترین وظیفه لایه پیوند داده

- مدیریت خطاهای انتقال
- تنظیم جریان داده ها

ملاحظات طراحی لایه پیوند داده(ادامه)



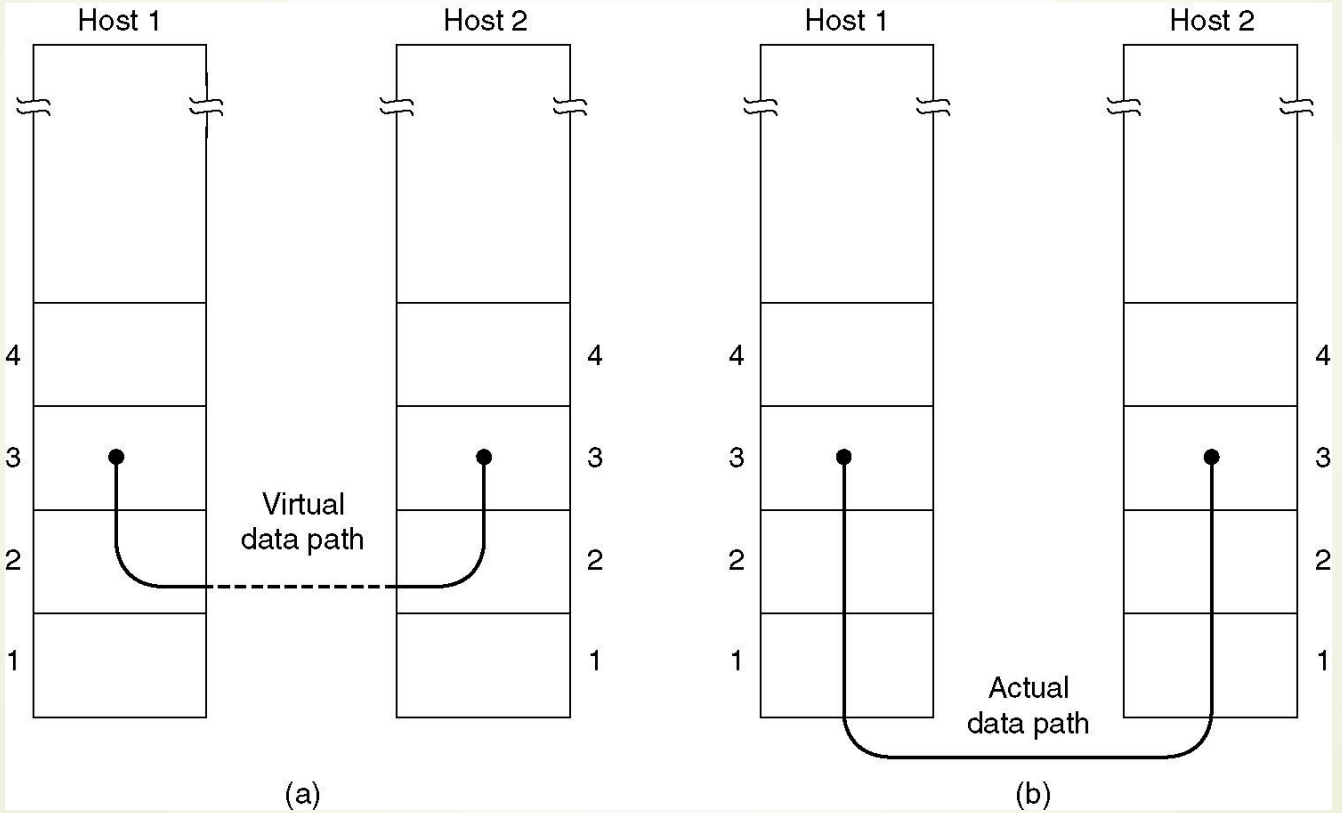
رابطه بین بسته و فریم

سرویس هایی ارایه شده به لایه شبکه

انتقال داده ها از لایه شبکه مبدا به لایه ماشین مقصد

۱. سرویس غیر متصل(connection less) بدون تصدیق دریافت(**Ack**)
۲. سرویس غیر متصل با تصدیق دریافت
۳. سرویس اتصال-گرا (connection oriented)با تصدیق دریافت

سرویس هایی ارایه شده به لایه شبکه(ادامه)



(a)ارتباط مجازی (b)ارتباط واقعی

سرویس هایی ارایه شده به لایه شبکه
(ادامه-۲)

سرویس غیر متصل بدون تصدیق دریافت(مانند سیستم پست)

- اتصال منطقی وجود ندارد
- این سرویس برای نرخ پایین خطا مناسب است
- تشخیص خطا ندارد
- برای ترافیک realtime مناسب است
- مقابله با خطا در لایه بالاتر
- اغلب LANها و سرویس صدا

سرویس هایی ارایه شده به لایه شبکه
(ادامه-۳)

سرویس غیر متصل با تصدیق دریافت

- اتصال منطقی بین مبدا و مقصد وجود ندارد
- دریافت فریم ها از سوی مقصد تصدیق می شود
- مناسب برای کانال های غیر قابل اعتماد
- سیستم های بیسیم

سرویس هایی ارایه شده به لایه شبکه
(ادامه-۴)

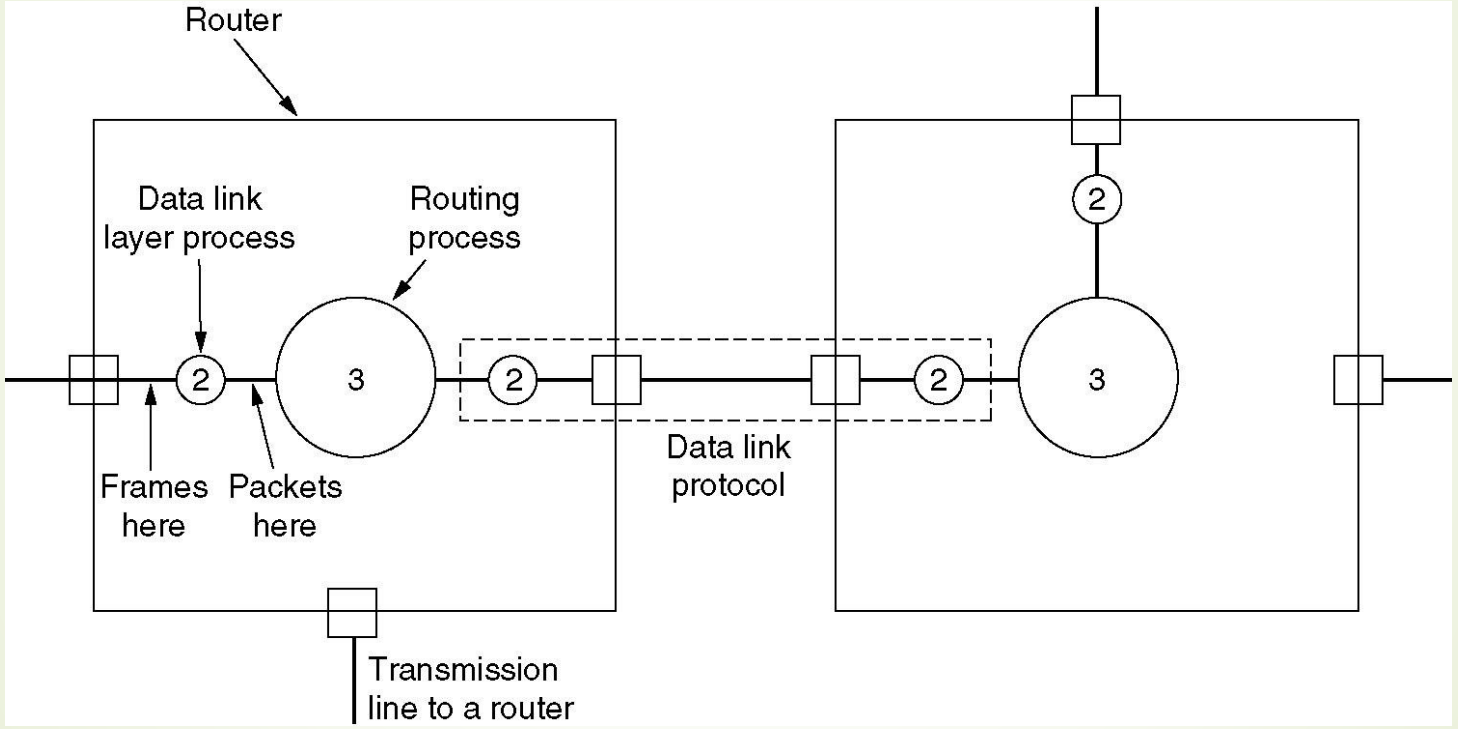
سرویس اتصال گرا با تصدیق دریافت

- بهترین سرویس لایه پیوند داده
- شماره گذاری فریم
- تصدیق دریافت توسط ماشین مقصد
- ارسال داده ها در سه مرحله

۱. برقراری اتصال و مقداردهی متغیر های لازم
۲. انتقال فریم ها
۳. قطع اتصال و آزاد سازی منابع

سرویس هایی ارایه شده به لایه شبکه

(ادامه-۵)



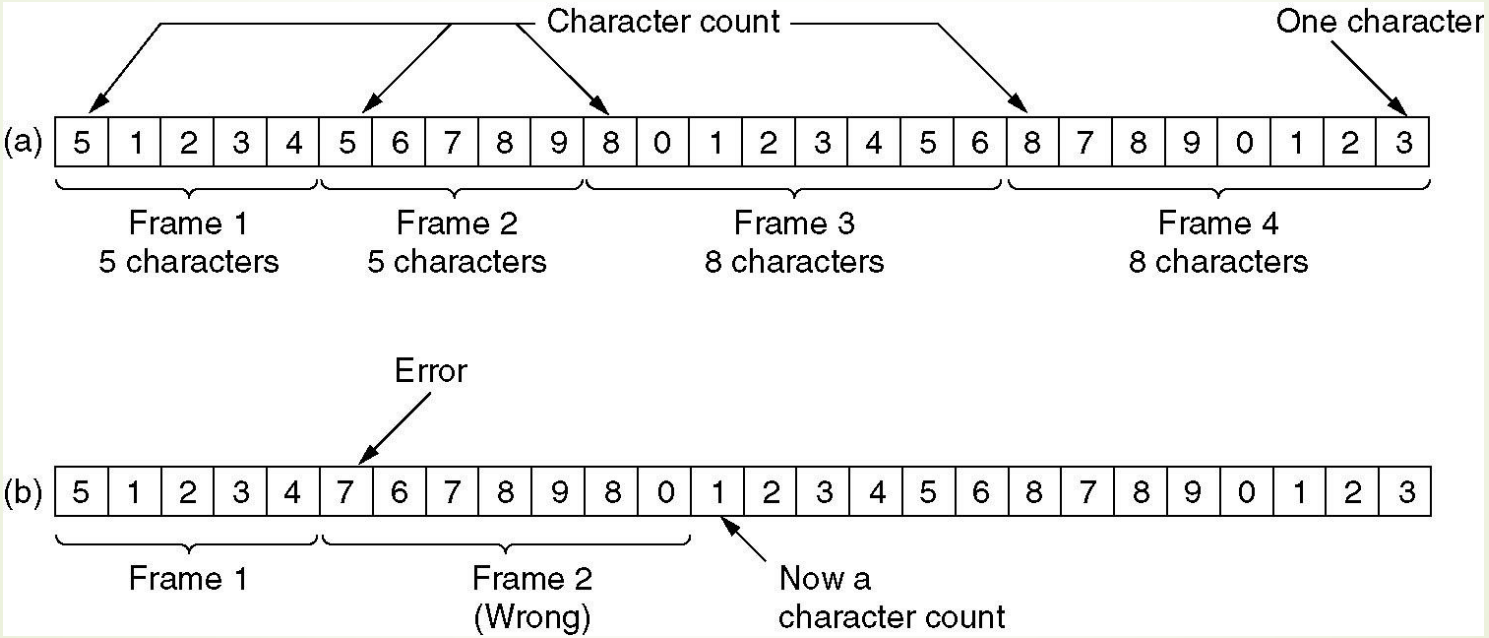
محل فعالیت لایه پیوند داده

فریم بندی

- تعریف فریم بندی
- روش های کشف خطا

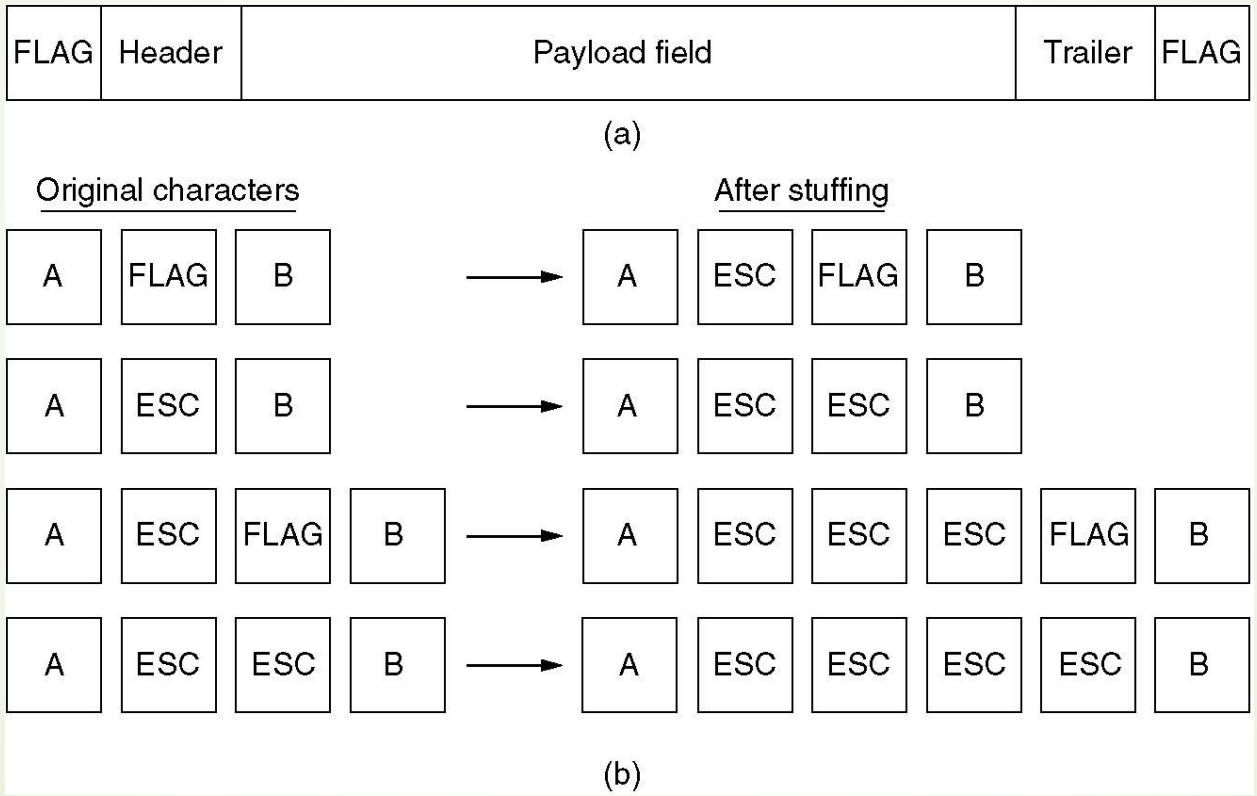
۱. شمارش کارکترها
۲. بایت های پرچم، با درج بایت (byte/character stuffing)
۳. پرچم های شروع وپایان، با درج بیت (bit stuffing)
۴. حالت های غیر مجاز کدگذاری لایه فیزیکی

فریم بندی(ادامه)



استریم کارکترها (a) بدون خطا (b) باخطا

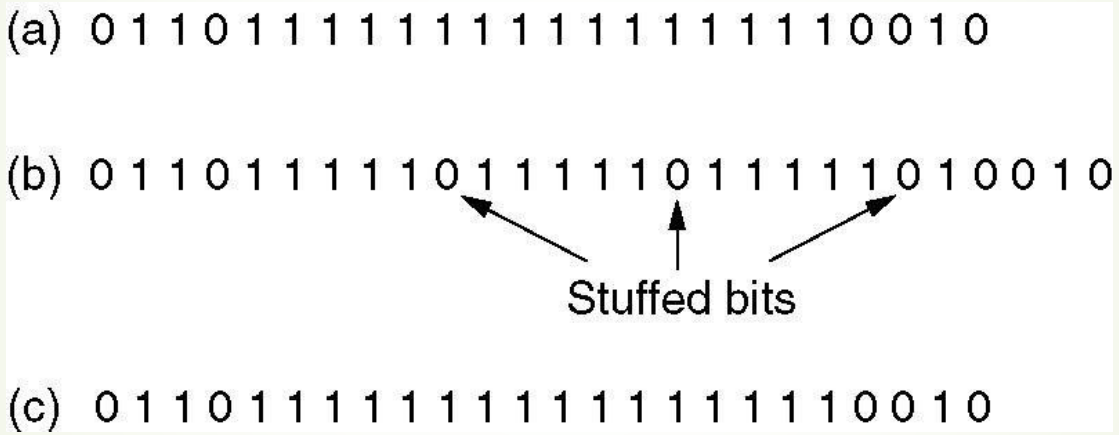
فریم بندی(ادامه-۲)



(a) تعیین ابتدا و انتهای فریم با استفاده از بایت پرچم

(b) چهارتوالی بایت قبل و بعد از درج بایت

فریم بندی(ادامه-۳)



درج بیت: (a) داده اولیه (b) داده هابصورتی که روی خط فیزیکی ارسال می شود. (c) داده ها به صورت که درگیرنده دریافت می شود.

کنترل خطا و کنترل جریان

کنترل خطا

- چگونگی ارسال فریم ها به ترتیب وسالم
- اطلاع از رسیدن بسته به مقصد
- تایمر برای حل مشکل فریم ناپدید شده

کنترل جریان

- کنترل جریان براساس بازخور
- کنترل جریان براساس نرخ

کشف و تصحیح خطا

- کدهای تصحیح خطا (error-correcting code)
  - قابل استفاده در کانال های پر از خطا مانند لینک های بیسیم
- کدهای کشف خطا (error-detecting codes)
  - قابل استفاده در کانال های قابل اطمینان مانند فیبرنوری
- فاصله همینگ
  - به ازای کشف d خطا به کدی بافاصله همینگ d+1
  - طراحی کدی با m بیت داده اصلی و r بیت افزونگی به منظور کشف تمام خطاهای تک بیتی

کشف و تصحیح خطا (ادامه)

کلمه کد = داده چک کننده(افزونگی) + داده اصلی

$n = m + r$

وزن کد(W): تعداد بیت‌های ۱ موجود در کد

فاصله همینگ (d):

- تعداد بیت‌های ۱ متفاوت در دو کد
- با XOR دو کد به دست می‌آید
- فاصله همینگ یک مجموعه کد = حداقل فاصله همینگ آن مجموعه

C_1 = 10011011

C_2 = 10101101 XOR

$W(00110110) = 4$

$D(C_1, C_2) = 4$

کنترل خطا

- پیش رو (Forward Error Correction) : کدهای افزونه برای تشخیص و تصحیح خطا
- پس رو(Backward Error Correction): کدهای افزونگی فقط برای تشخیص خطا

کشف خطا

- سوال:

اگر فاصله همینگ در یک مجموعه کد برابر ۵ باشد. چند بیت خطا قابل تشخیص است؟ چند بیت خطا قابل تصحیح است؟ چرا؟

کشف خطا

جواب:

- برای تشخیص d خطا به کدی با فاصله همینگ d+1 نیاز است
- برای تصحیح d خطا به کدی با فاصله همینگ 2d+1 نیاز است

تشخیص خطا

روش Block Sum Check

R6	b6	b5	b4	b3	b2	b1	b0
	1	0	1	0	0	1	0
	0	1	0	0	1	0	0
	0	1	1	0	0	1	1
	1	0	0	1	0	0	1
	0	1	1	0	1	1	1

تشخیص خطا

روش Block Sum Check

R6	b6	b5	b4	b3	b2	b1	b0
0	1	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	0	1	1
0	1	0	0	1	0	0	1
0	0	1	1	0	1	1	1
0	0	1	1	1	0	1	1

تشخیص خطا

- روش Block Sum Check

R6	b6	b5	b4	b3	b2	b1	b0
0	1	0	1	0	0	1	0
1	0	**1**	0	0	**1**	0	0
1	0	1	1	0	0	1	1
0	1	**0**	0	1	**0**	0	1
0	0	1	1	0	1	1	1
0	0	1	1	1	0	1	1

سوال

برای ایجاد کدی با m بیت داده اصلی و r بیت داده افزونه که بتواند یک بیت خطا را تصحیح کند، حداقل r چقدر باید باشد؟

جواب

- کلمه کد: $n=m+r$
- تعداد پیام های مجاز 2^m
- برای تشخیص و تعیین محل وقوع یک بیت خطا: هر پیام به n کد غیر مجاز با فاصله یک نیاز دارد
- حداقل تعداد کدهای لازم که نبایستی همپوشانی داشته باشند: $2^m(n+1)$
- کل ترکیبات ممکنه: 2^n

$$\begin{cases} 2^m(n+1) \le 2^n \\ n = m + r \end{cases} \Rightarrow m + r + 1 \le 2^r$$

همینگ کد

برای تصحیح خطاهای تک بیتی بکار می‌رود

- شماره گذاری بیت‌ها از چپ به راست
- شماره هایی که توانی از دو هستند (۱،۲،۴،۸،...) بیتهای افزونه چک کننده (r)
  و مابقی (۳،۵،۶،۷،۹،...) داده اصلی(m)

مثال: توازن بیت داده ۱۳ توسط بیت‌های افزونه ۸+۴+۱ چک می‌شود

همینگ کد برای تصحیح خطاهای فورانی

تصحیح حداکثر k بیت خطای پشت سرهم

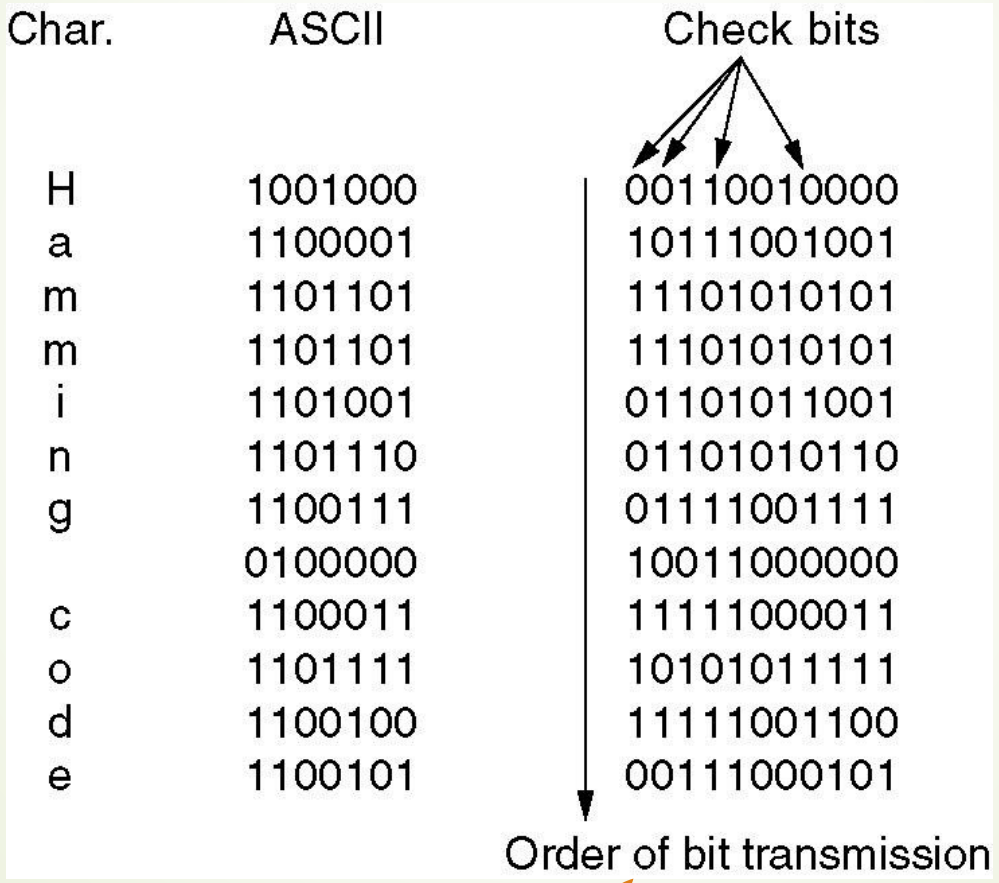
- قرار دادن k کلمه کد m+r بیتی کنار هم (سطر به سطر)
- ارسال ستونی به جای ارسال سطری

اگر حداکثر k خطای فورانی اتفاق بیافتد، یعنی در هر کلمه حداکثر یک بیت خطا اتفاق افتاده است

r بیت لازم برای تصحیح t بیت خطا

$$2^r \geq 1+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{t}$$

کشف و تصحیح خطا(ادامه)



استفاده از کد همینگ برای تصحیح خطاهای فورانی

کد افزونگی چرخه‌ای (CRC)

قوانین:

- استفاده از تقسیم مبنای دو
- جمع و تفریق پیمانه دو (XOR)
- نمایش عدد با چند جمله‌ای (۱۰۱۱۱ = $x^4+x^2+x^1+1$)
- توافق فرستنده و گیرنده بر یک چند جمله ای $G(x)$
- کم ارزش‌ترین و پر ارزش‌ترین بیت‌های $G(x)$ باید یک باشد

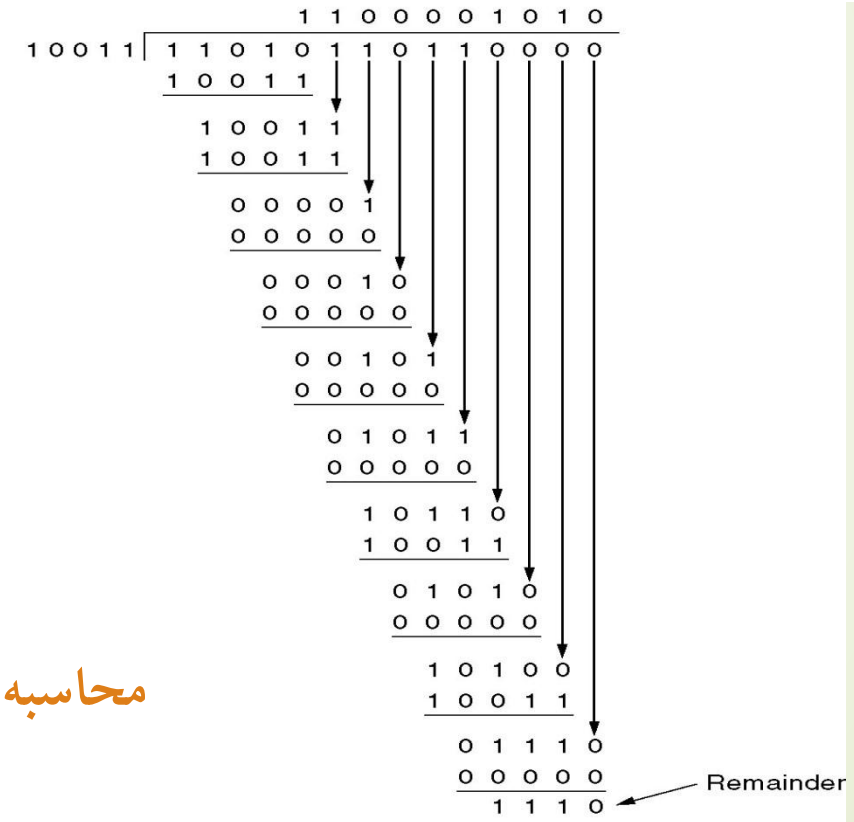
کد افزونگی چرخه‌ای (ادامه)

روال تولید CRC:

- اضافه کردن r بیت صفر به سمت راست داده اصلی(m)
  - r یکی کمتر از تعداد بیت‌های G(x)
- تقسیم داده جدید بر G(x)
- افزودن باقیمانده تقسیم به صورت r بیتی به سمت راست داده اصلی و ارسال

کد افزونگی چرخه‌ای (ادامه)

Frame : 1 1 0 1 0 1 1 0 1 1
Generator: 1 0 0 1 1
Message after 4 zero bits are appended: 1 1 0 1 0 1 1 0 1 1 0 0 0 0



Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 1 0

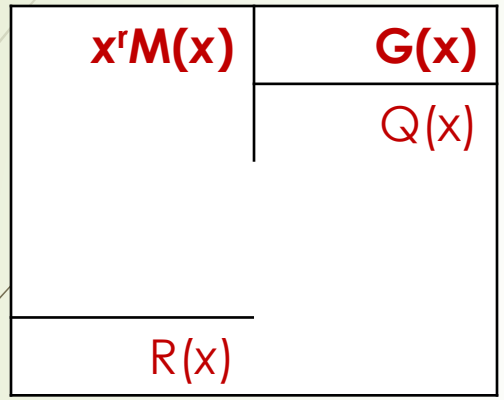
محاسبه مجموع چک کد چند جمله ای

کد افزونگی چرخه‌ای (ادامه)

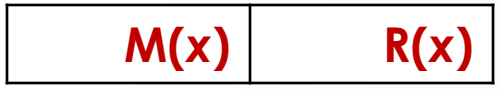
- اگر در گیرنده کد ارسالی بر $G(x)$ تقسیم شود باقیمانده باید صفر شود در غیر این‌صورت خطا

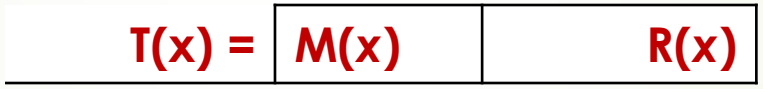
- سوال: در روش CRC چه هنگام خطا در گیرنده غیر قابل تشخیص خواهد بود؟

کد افزونگی چرخه‌ای (ادامه)

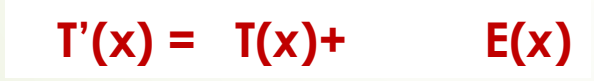


- کد ارسالی





- با بروز خطای E(x) داریم



- با انجام عمل تقسیم در گیرنده ...

قابلیت تشخیص کد CRC

- **کلیه خطاهای تک بیتی قابل تشخیص هستند ($E(x) = x^i$)**
  - $E(x)$ باید حداقل دو جمله داشته باشد تا قابل شناسایی نباشد
- **اگر $G(x)$ بر $x+1$ بخش‌پذیر باشد آنگاه تمامی خطاهای فرد قابل تشخیص هستند.** چرا؟

می‌توان ثابت کرد که هیچ چند جمله‌ای با تعداد جملات فرد بر $x+1$ بخش‌پذیر نیست.
اگر تعداد خطاها فرد باشد، چند جمله‌ای خطا فرد است لذا بر $x+1$ بخش‌پذیر نخواهد بود

اثبات: برهان خلف، $E(x)$ یک چند جمله‌ای با تعداد جملات فرد که بر $x+1$ بخش‌پذیر است آنگاه

$$E(x) = (x+1)Q(x)$$

$$E(1) = (1+1)Q(1) \quad \Rightarrow \quad odd = even\ !$$

قابلیت تشخیص کد CRC(ادامه)

- اگر G(x) حداقل دارای ۳ جمله باشد آنگاه تمامی خطاهای دو بیتی قابل تشخیص
- اگر باقیمانده r بیتی باشد آنگاه تمام خطاهای فورانی با طول کوچک‌تر یا مساوی r

- اکثر خطاهای فورانی با طول بیشتر از r
  - احتمال عدم تشخیص خطاهای فورانی به طول $r+1$ برابر $1/(2^{r-1})$
  - احتمال عدم تشخیص خطاهای فورانی به طول بیشتر از $r+1$ برابر $1/2^r$

توابع مولد استاندارد در CRC

$$CRC-12: \quad x^{12}+x^{11}+x^{3}+x^{2}+x+1$$

$$CSC-16: \quad x^{16}+x^{15}+x^{2}+1$$

$$CRC-CCITT: \quad x^{16}+x^{12}+x^{5}+1$$

- CRC-12 وقتی بکار میرود که طول کاراکتر ۶ بیت باشد
- CSC-16 وقتی بکار میرود که طول کاراکتر ۸ بیت باشد
- در CRC-CCITT تمام خطاهای منفرد، مضاعف، تمام خطاهایی با تعداد بیت فرد، تمام خطاهای پیوسته به طول ۱۶ و یا کمتر، درصد بالایی از خطاهای ۱۷و ۱۸ بیتی و بیشتر قابل تشخیص است

پروتکل های ساده لینک داده

فرض های اساسی در مدل ارتباطی زیر بنایی

۱. مستقل بودن پروسس های فعال
۲. ارتباط لایه ها از طریق ردوبدل کردن پیام
۳. سرویس اتصال-گرای قابل اعتماد
۴. ماشین مبدا منبع بی پایانی از داده ها دارد
۵. کامپیوتر ها هیچ گاه از کار نمی افتند
۶. فقط داده خالص به لایه شبکه داده می شود

تعاریف پروتکل به زبان C

```
#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;
```

تعاریف پروتکل به زبان C (ادامه)

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

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

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

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

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

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

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

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

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

پروتکل های ساده لینک داده

پروتکل یکطرفه نامحدود

۱. داده ها فقط در یک جهت منتقل می شوند.
۲. لایه شبکه در گیرنده و فرستنده آماده به کار.
۳. زمان پردازش نادیده گرفته می شود.
۴. بافر محدودیتی ندارد.
۵. کانال ارتباطی کامل و بدون خطا.

پروتکل یکطرفه نامحدود

```
/* 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 quickly.
   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 */
  }                                      * Tomorrow, and tomorrow, and tomorrow,
                                           Creeps in this petty pace from day to day
                                           To the last syllable of recorded time
                                               - Macbeth, V, v */
}

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 */
  }
}
```

پروتکل های ساده لینک داده(ادامه)

پروتکل توقف-انتظار یکطرفه

۱. داده ها در یک جهت منتقل می شوند(دوطرفه ناهمزمان)
۲. کانال ارتباطی کامل و بدون خطا
۳. برگردان بازخور به عنوان مجوز ارسال فریم بعدی
۴. پروتکل توقف-انتظار (stop-and-wait)

پروتکل توقف-انتظار یکطرفه

```
/* 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);            /* bye bye little frame */
      wait_for_event(&event);           /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                           /* buffers for frames */
  event_type event;                     /* frame_arrival is the only possibility */
  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_layer(&s);            /* send a dummy frame to awaken sender */
  }
}
```

پروتکل های ساده لینک داده(ادامه-۲)

پروتکل یکطرفه برای کانال های نویز دار

۱. داده ها در یک جهت منتقل می شوند.(دوطرفه ناهمزمان)

۲. کانال ارتباطی می تواند دارای نویز باشد.

۳. فریم ها دارای یک شماره ترتیبی هستند.

۴. نام های دیگر این پروتکل **PAR** (تصدیق دریافت مثبت با ارسال مجدد) یا **ARQ** (درخواست تکرار خودکار)

پروتکل یکطرفه برای کانال های نویز دار

```
/* 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;                                          /* scratch variable */
  packet buffer;                                    /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;                           /* initialize outbound sequence numbers */
  from_network_layer(&buffer);                      /* fetch first packet */
  while (true) {
       s.info = buffer;                             /* construct a frame 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);                      /* frame_arrival, cksum_err, timeout */
       if (event == frame_arrival) {
            from_physical_layer(&s);                /* get the acknowledgement */
            if (s.ack == next_frame_to_send) {
                 stop_timer(s.ack);                 /* turn the timer off */
                 from_network_layer(&buffer);       /* get the next one to send */
                 inc(next_frame_to_send);           /* invert next_frame_to_send */
            }
       }
  }
}
```

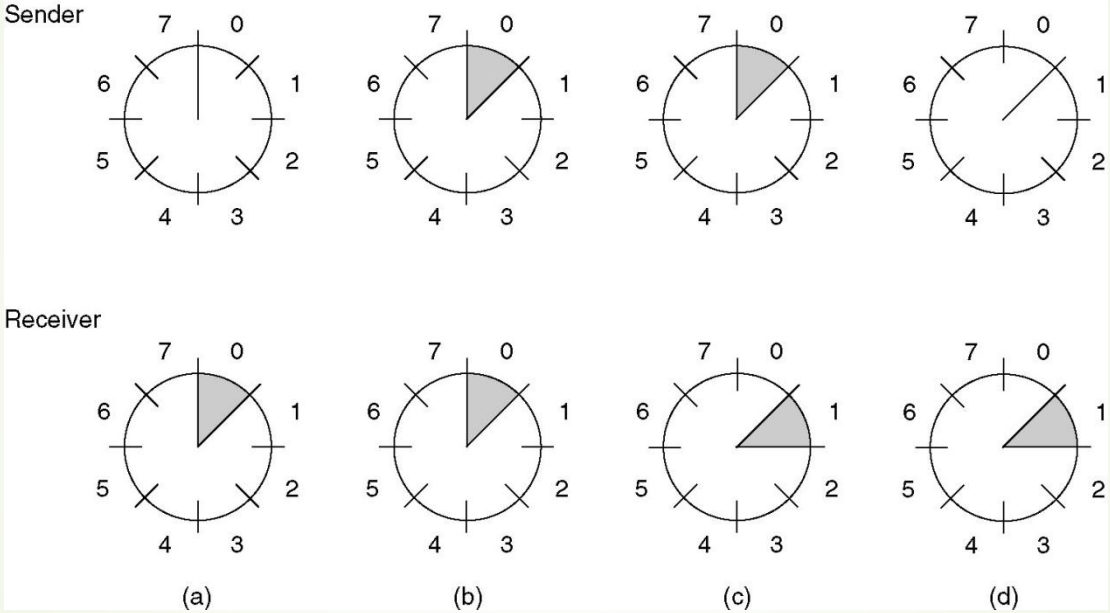
پروتکل یکطرفه برای کانال های نویز دار (ادامه)

```
void receiver3(void)
{
 seq_nr frame_expected;
 frame r, s;
 event_type event;

 frame_expected = 0;
 while (true) {
      wait_for_event(&event);                  /* possibilities: frame_arrival, cksum_err */
      if (event == frame_arrival) {            /* a valid frame has arrived. */
            from_physical_layer(&r);           /* go get the newly arrived frame */
            if (r.seq == frame  expected) {    /* this is what we have been waiting for. */
                  to_network_layer(&r.info);   /* pass the data to the network layer */
                  inc(frame_expected);         /* next time expect the other sequence nr */
            }
            s.ack = 1 – frame_expected;        /* tell which frame is being acked */
            to_physical_layer(&s);             /* send acknowledgement */
      }
 }
}
```

پروتکل های پنجره لغزنده

- کانال های دوطرفه و راه های دستیابی به آن
- سوار(کول) کردن (piggyback)، مزایا و معایب
- اصطلاحات پنجره دریافت، پنجره ارسال
- پروتکل پنجره لغزنده ۱-بیتی
- پروتکل "N تا به عقب برگرد"
- پروتکل تکرارانتخابی

پروتکل های پنجره لغزنده(ادامه)



یک پنجره لغزنده یک واحدی، با شماره ترتیبی ۳-بیتی. (a) درشروع کار.(b) بعد از ارسال اولین فریم. (c) بعد از آنکه اولین فریم دریافت شد.(d) بعد از آنکه فرستنده اولین تصدیق دریافت را گرفت

پروتکل های پنجره لغزنده (ادامه-۲)

پروتکل پنجره لغزنده ۱-بیتی

- پنجره گیرنده و پنجره فرستنده
- پروتکل توقف-انتظار
- وضعیت ارسال اولین فریم بطور همزمان
- نقش کلیدی فیلد تصدیق

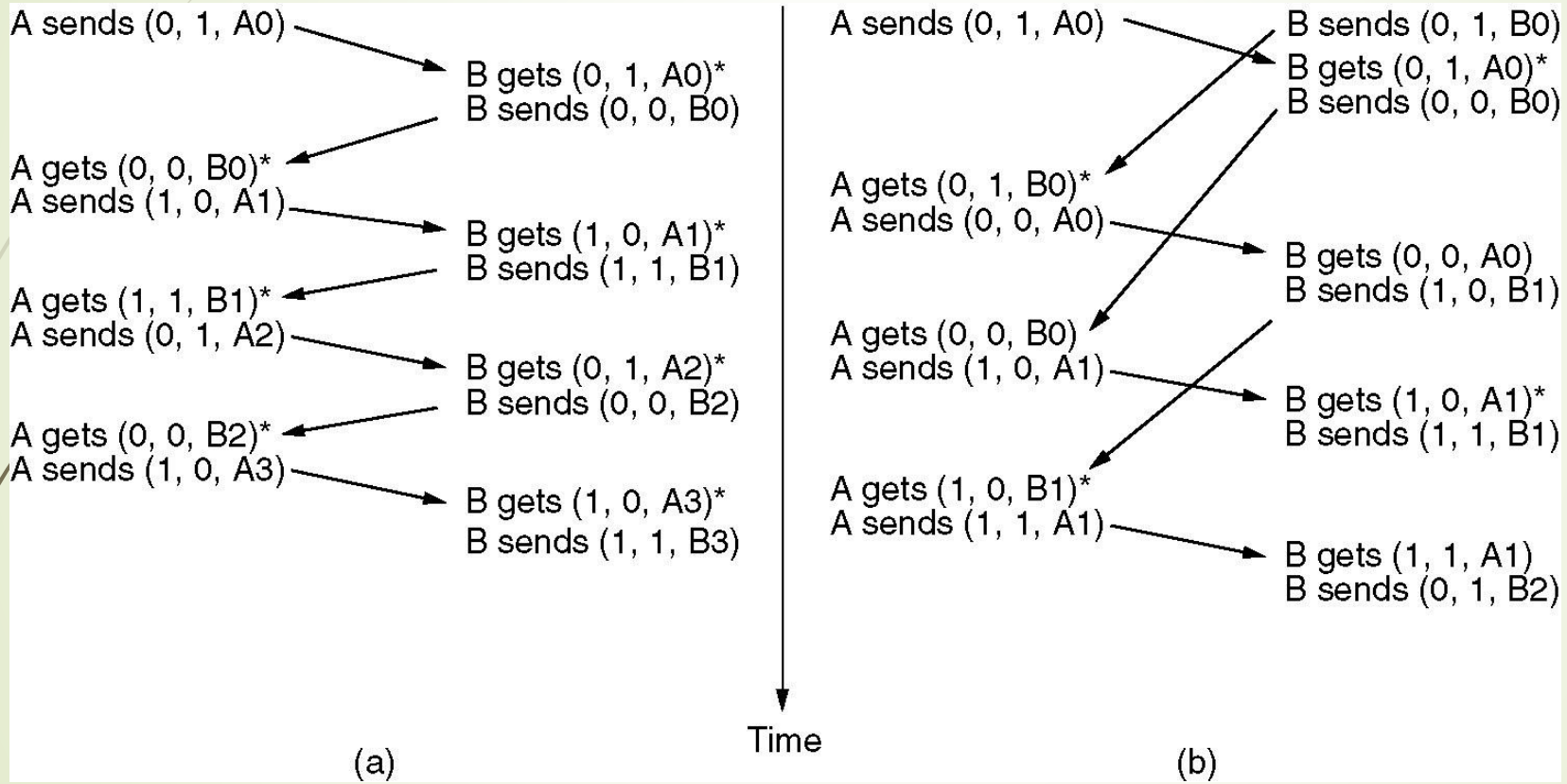
پروتکل پنجره لغزنده ۱-بیتی


```
/* Protocol 4 (sliding window) is bidirectional. */
#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;                               /* frame expected next */
  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 */
```

پروتکل پنجره لغزنده ۱-بیتی(ادامه)


```
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 seq number expected next */
        }

        if (r.ack == next_frame_to_send) {  /* handle outbound frame stream. */
            stop_timer(r.ack);              /* turn the timer off */
            from_network_layer(&buffer);    /* fetch new pkt from network layer */
            inc(next_frame_to_send);        /* invert senderís sequence number */
        }
    }
    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 */
 }
}
```

پروتکل پنجره لغزنده ۱-بیتی(ادامه-۲)



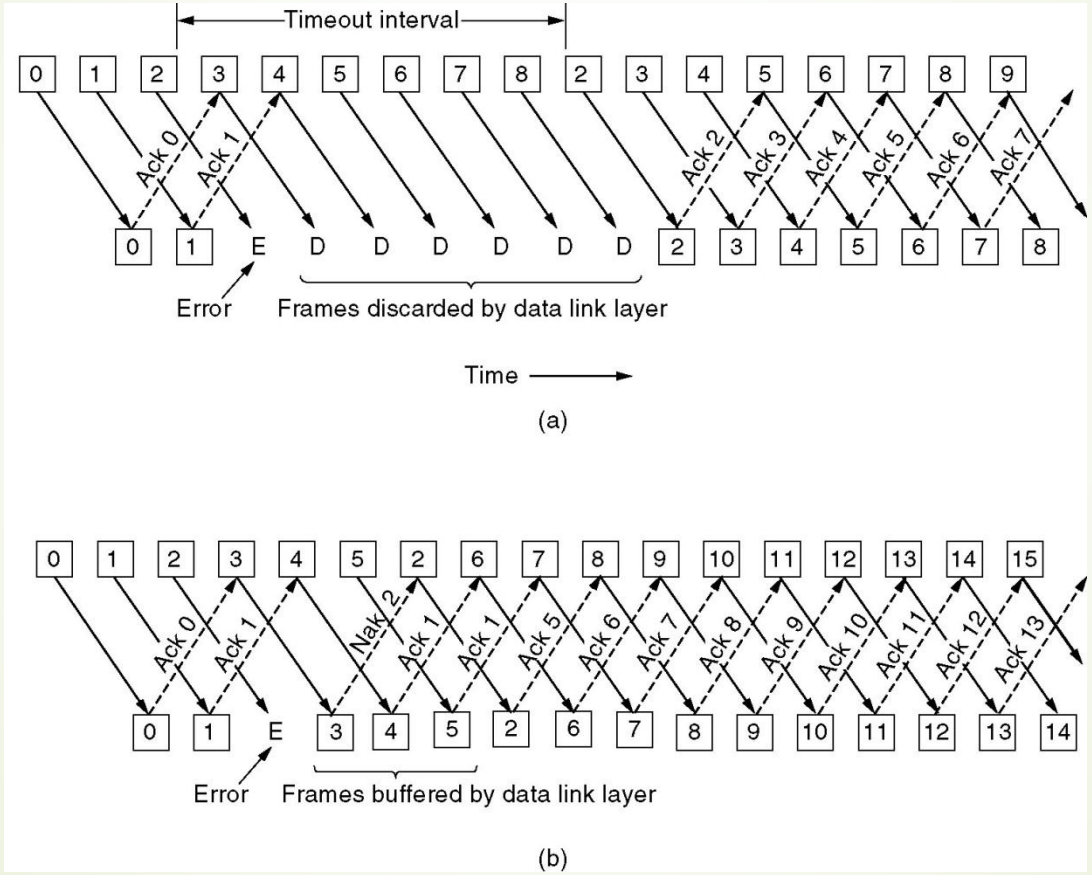
دوسناریوی پروتکل ۴ (a) حالت عادی (b) حالت غیرعادی. اعداد داخل پرانتز از چپ به راست عبارتند از : seq، ackو شماره بسته. بسته هایی که پذیرفته و به لایه شبکه تحویل می شوند، با * مشخص شده اند.

پروتکل های پنجره لغزنده

- پروتکل "N تا به عقب برگرد"
  - فرستادن W فریم به جای یک فریم
  - اگر ***پهنای باند*** × ***تاخیر رفت وبرگشت*** عددی بزرگ باشد
    آنگاه ***پنجره ارسال بزرگ***
  - تکنیک لوله کشی
    - $l / (l + bR)$ = بهره خط در روش توقف و انتظار
    - اگر $bR < l$ کارایی خط زیر 50%
    - مقابله با خطا : ۱.رهیافت N تا به عقب برگرد
      ۲.تکرار انتخابی

l:طول فریم(بیت) *b*:پهنای باند(بیت بر ثانیه) *R*:تاخیر رفت و برگشت (ثانیه)

مثال



مقابله با خطا در خط لوله .تاثیر خطا وقتی که (a) اندازه پنجره دریافت گیرنده ۱ است، و (b) پنجره دریافت بزرگ است.

پروتکل N" تا به عقب برگرد" (ادامه)

```
/* Protocol 5 (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"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <=b < c circularly; false otherwise. */
  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 */
}
```

پروتکل "N تا به عقب برگرد"(ادامه-۲)

```
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 */
```

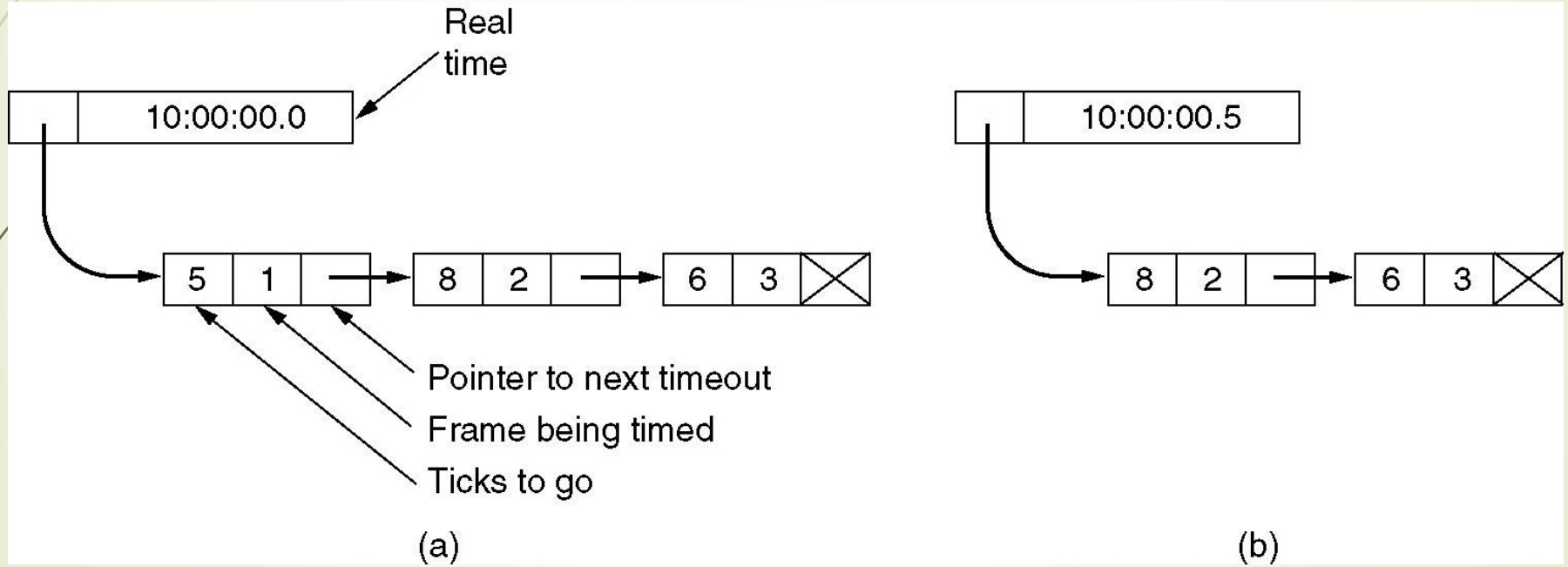
پروتکل "N تا به عقب برگرد"(ادامه-۳)

```
while (true) {
  wait_for_event(&event);              /* four possibilities: see event_type above */

  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 + 1;    /* 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 */
          }
```

پروتکل "N تا به عقب برگرد"(ادامه-۴)

```
            /* Ack n implies n – 1, n – 2, etc.  Check for 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;

        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 = 1; 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 */
            }

    }

    if (nbuffered < MAX_SEQ)
            enable_network_layer();
    else
            disable_network_layer();
  }
}
```

شبیه سازی چند تایمر بوسیله نرم افزار



پروتکل های پنجره لغزنده

پروتکل تکرار انتخابی

- مناسب برای خطوط پر نویز
- بافر کردن فریم های سالم بعد از یک یا چند فریم معیوب
- پنجره ارسال متغیر ولی پنجره دریافت ثابت
- مساله همپوشانی
- تصدیق دریافت منفی

پروتکل تکرار انتخابی

```
/* Protocol 6 (nonsequential receive) accepts frames out of order, but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7                                   /* should be 2^n – 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                              /* no nak has been sent yet */
seq_nr oldest_frame = MAX- SEQ + 1;                 /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) ll ((c < a) && (a <= b)) ll ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data, ack, or nak frame. */
  frame s;                                          /* scratch variable */

  s.kind = fk;                                      /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                                 /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;                    /* one nak per frame, please */
  to_physical_layer(&s);                            /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                                 /* no need for separate ack frame */
}
```

پروتکل تکرار انتخابی(ادامه)


```
void protocol6(void)
{
 seq_nr ack_expected;                    /* lower edge of sender's window */
 seq_nr next_frame_to_send;              /* upper edge of sender's window + 1 */
 seq_nr frame_expected;                  /* lower edge of receiver's window */
 seq_nr too_far;                         /* upper edge of receiver's window + 1 */
 int i;                                  /* index into buffer pool */
 frame r;                                /* scratch variable */
 packet out_buf[NR_BUFS];                /* buffers for the outbound stream */
 packet in_buf[NR_BUFS];                 /* buffers for the inbound stream */
 boolean arrived[NR_BUFS];               /* inbound bit map */
 seq_nr nbuffered;                       /* how many output buffers currently used */
 event_type event;

 enable_network_layer();                 /* initialize */
 ack_expected = 0;                       /* next ack expected on the inbound stream */
 next_frame_to_send = 0;                 /* number of next outgoing frame */
 frame_expected = 0;
 too_far = NR_BUFS;
 nbuffered = 0;                          /* initially no packets are buffered */
 for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```

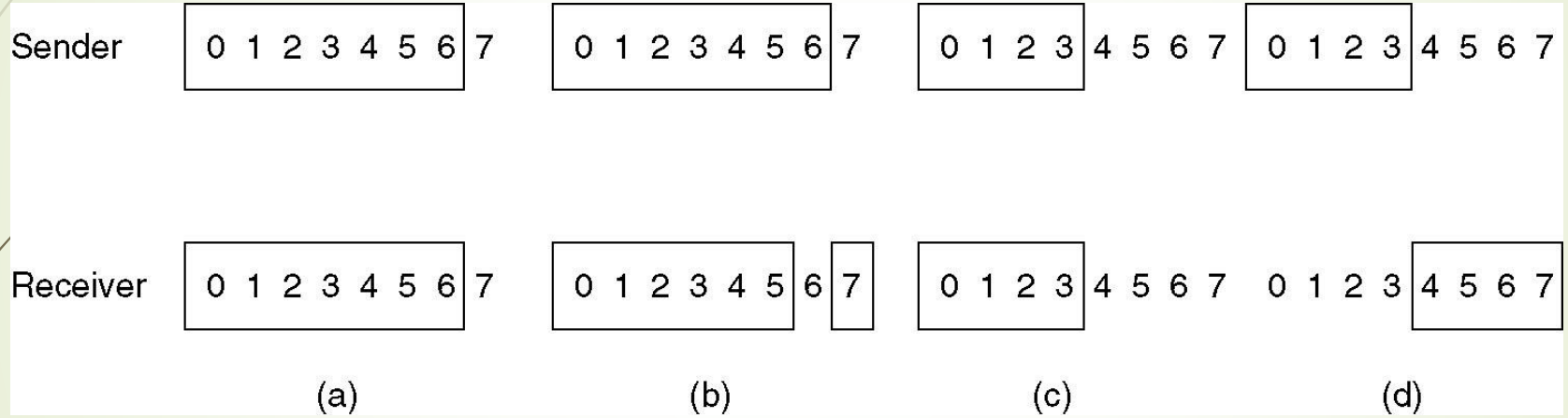
پروتکل تکرار انتخابی (ادامه-۲)

```
while (true) {
  wait_for_event(&event);                    /* five possibilities: see event_type above */
  switch(event) {
    case network_layer_ready:                /* accept, save, and transmit a new frame */
        nbuffered = nbuffered + 1;           /* expand the window */
        from_network_layer(&out  buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
        send_frame(data, next_frame_to_send, frame_expected, out_buf);/* transmit the frame */
        inc(next_frame_to_send);             /* advance upper window edge */
        break;

    case frame_arrival:                      /* a data or control frame has arrived */
        from_physical_layer(&r);             /* fetch incoming frame from physical layer */
        if (r.kind == data) {
            /* An undamaged frame has arrived. */
            if ((r.seq != frame_expected) && no_nak)
               send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
            if (between(frame_expected, r.seq, too_far) && (arrived[r.seq%NR_BUFS] == false)) {
                /* Frames may be accepted in any order. */
                arrived[r.seq % NR_BUFS] = true;      /* mark buffer as full */
                in_buf[r.seq % NR_BUFS] = r.info;     /* insert data into buffer */
                while (arrived[frame_expected % NR_BUFS]) {
                    /* Pass frames and advance window. */
                    to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                    no_nak = true;
                    arrived[frame_expected % NR_BUFS] = false;
                    inc(frame_expected);     /* advance lower edge of receiver's window */
                    inc(too_far);            /* advance upper edge of receiver's window */
                    start_ack_timer();       /* to see if a separate ack is needed */
                }
            }
        }
```

پروتکل تکرار انتخابی(ادامه-۳)

```
            if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next frame to send)
                  send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);

            while (between(ack_expected, r.ack, next_frame_to_send)) {
                  nbuffered = nbuffered - 1;            /* handle piggybacked ack */
                  stop_timer(ack_expected % NR_BUFS);     /* frame arrived intact */
                  inc(ack_expected);                    /* advance lower edge of sender's window */
            }
            break;
      case cksum_err:
            if (no_nak) send_frame(nak, 0, frame_expected, out_buf);/* damaged frame */
            break;
      case timeout:
            send_frame(data, oldest_frame, frame_expected, out_buf);/* we timed out */
            break;
      case ack_timeout:
            send_frame(ack,0,frame_expected, out_buf);      /* ack timer expired; send ack */
   }
   if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
 }
}
```

پروتکل تکرار انتخابی(ادامه-۴)

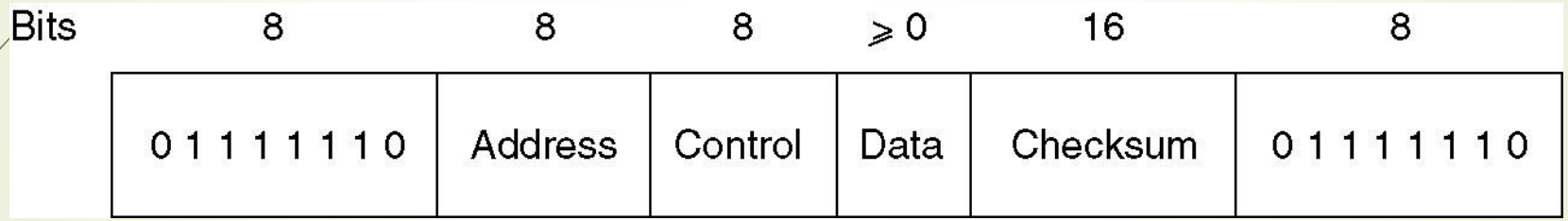


(a) پنجره های ارسال ودریافت هفت تایی در لحظه شروع. (b) بعد از رسیدن هفت فریم به مقصد، وقبل از بازگشت تصدیق دریافت به فرستنده. (c) پنجره های ارسال و دریافت چهارتایی در لحظه شروع. (d) بعد از رسیدن چهار فریم به مقصد، و قبل از بازگشت تصدیق دریافت به فرستنده

چند نمونه از پروتکل های لینک داده

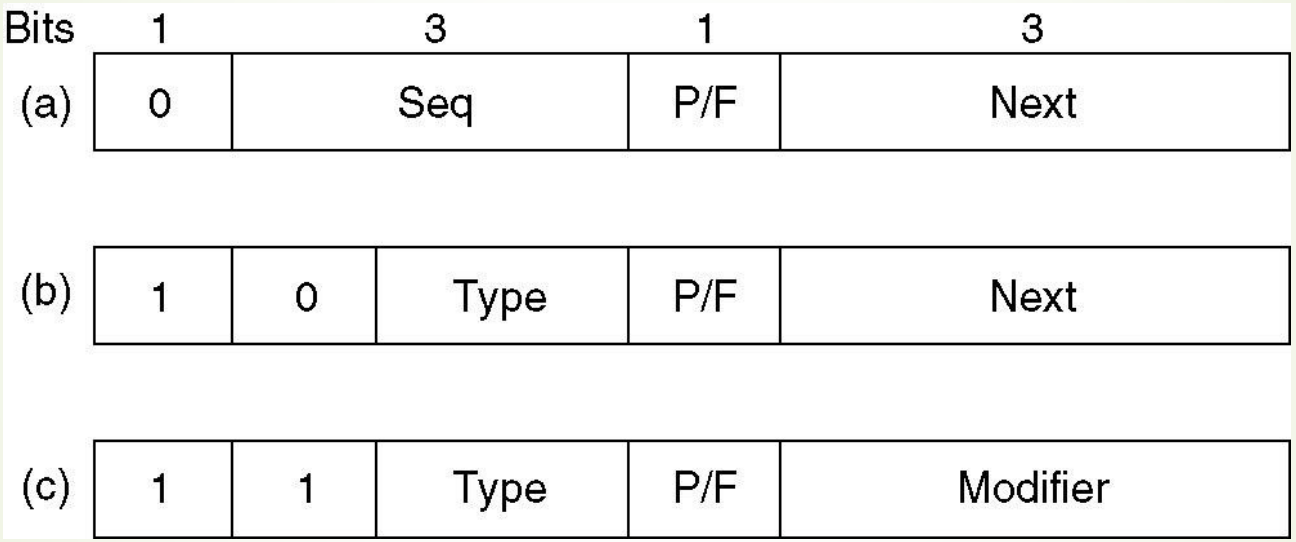
- HDLC کنترل سطح بالای لینک داده
- لایه پیوند داده در اینترنت

چند نمونه از پروتکل های لینک داده(ادامه)



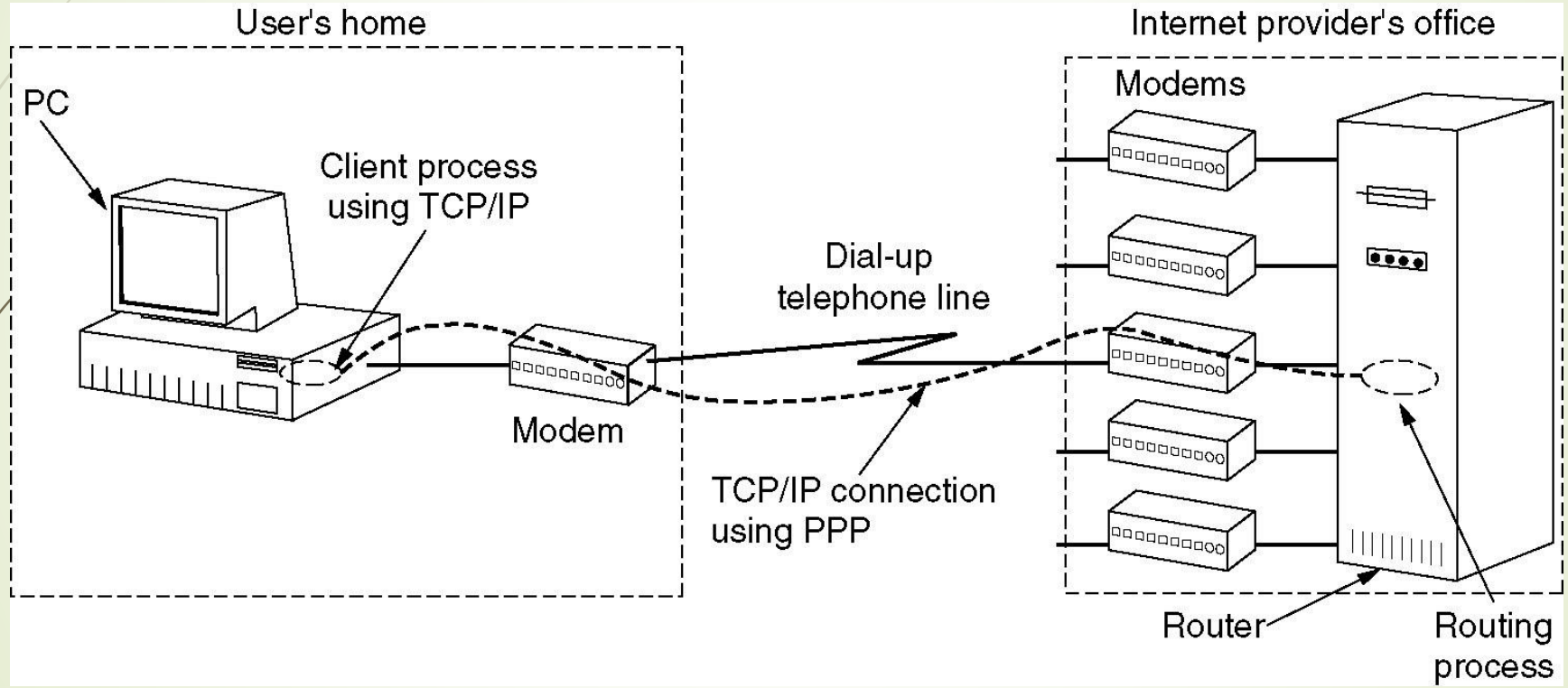
فرمت فریم در پروتکل های بیت-گرا

چند نمونه از پروتکل های لینک داده(ادامه-۲)



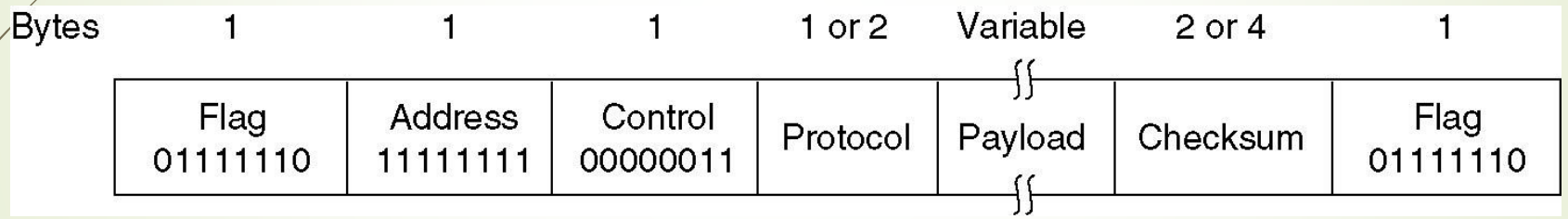
فیلد کنترل در یک (a) فریم اطلاعاتی (b) فریم سرپرستی (c) فریم بدون شماره

چند نمونه از پروتکل های لینک داده(ادامه-۳)



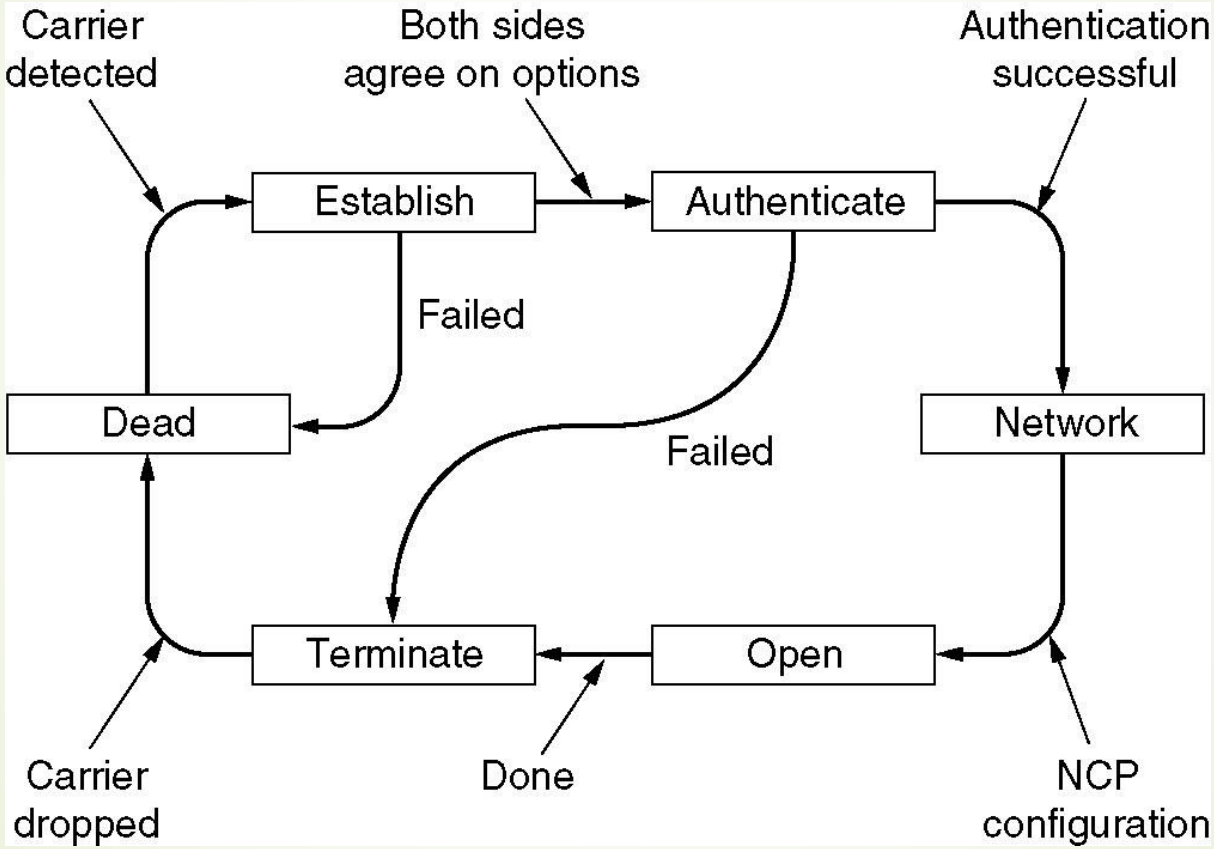
یک کامپیوتر خانگی می تواند نقش میزبان اینترنت رابازی کند.

چند نمونه از پروتکل های لینک داده(ادامه-۴)



فرمت فریم کامل PPP برای حالت بدون شماره

چند نمونه از پروتکل های لینک داده(ادامه-۵)



مراحل ساده شده برقراری و قطع خط در پروتکل PPP

چند نمونه از پروتکل های لینک داده(ادامه-۶)

Name	Direction	Description
Configure-request	I → R	List of proposed options and values
Configure-ack	I ← R	All options are accepted
Configure-nak	I ← R	Some options are not accepted
Configure-reject	I ← R	Some options are not negotiable
Terminate-request	I → R	Request to shut the line down
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request received
Protocol-reject	I ← R	Unknown protocol requested
Echo-request	I → R	Please send this frame back
Echo-reply	I ← R	Here is the frame back
Discard-request	I → R	Just discard this frame (for testing)

انواع فریم های LCP

پایان