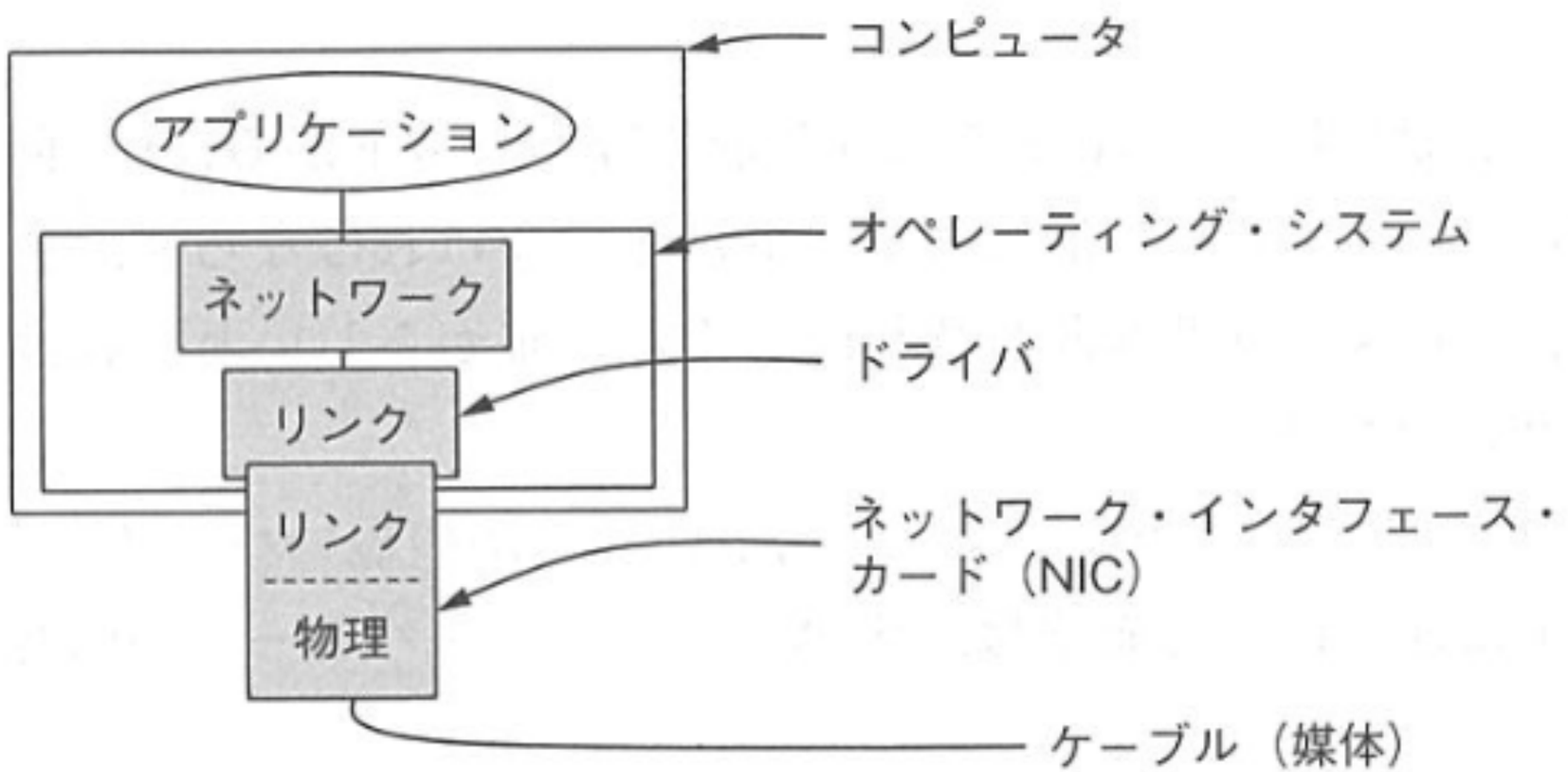
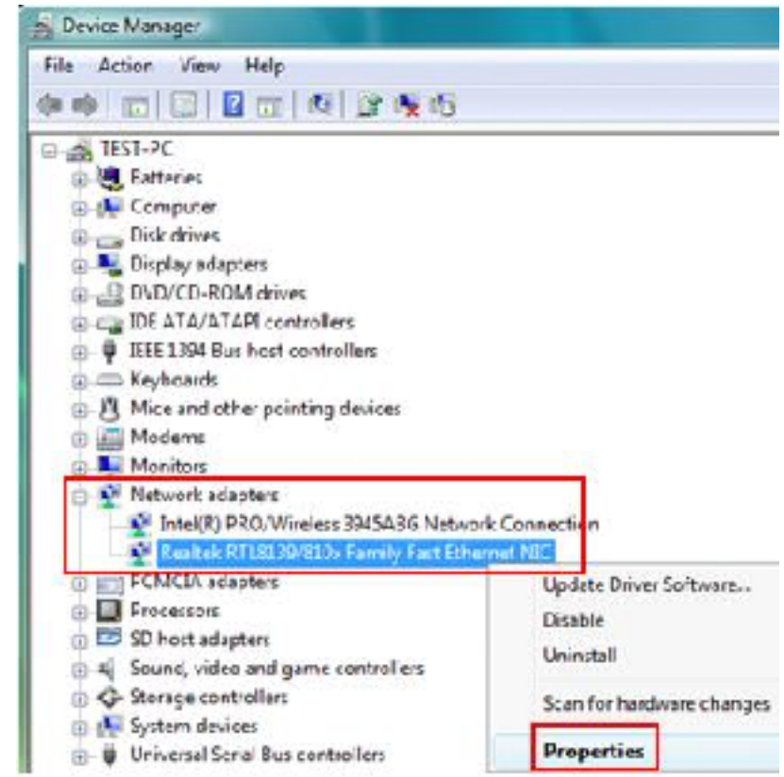
講義日程

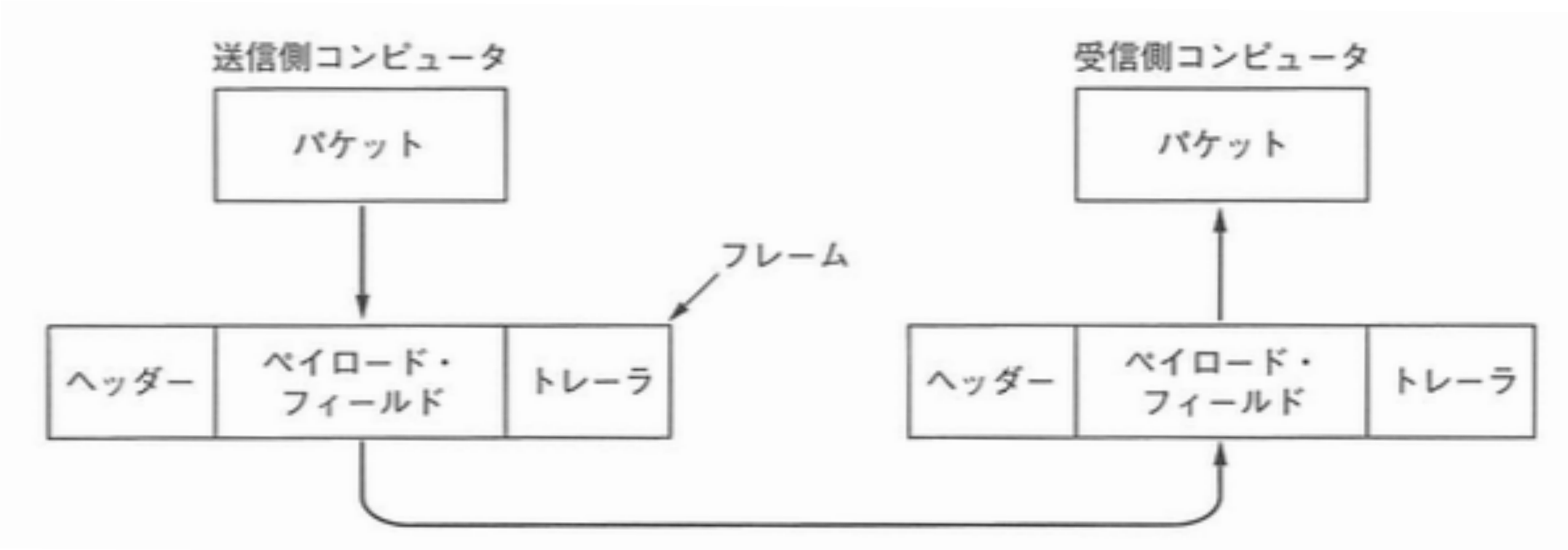
		授業計画		課題
04/11	第１回	計算機ネットワークの基本概念 ハードウェア・ソフトウェア，参照モデル	１章	ネットワークの種類と参照モデルを理解し プロトコル階層と各層の設計課題
04/14	第２回	物理層１ 有線伝送と無線伝送	２章	物理チャネルの特性を理解し データ通信の理論的基礎を理解
04/18	第３回	物理層２ デジタル変調と多重化	２章	ベースバンド伝送と通過帯域伝送， 電話網，携帯電話システムを説明できる
04/21	第４回	データリンク層１ 誤りの検出・訂正	３章	誤りの検出・訂正のしくみを理解し 検出・訂正符号の計算ができる
04/25	第５回	データリンク層２ データリンク・プロトコル	３章	データリンク・プロトコルの種類， 各プロトコルを定量的に評価できる
04/28	第６回	メディア・アクセス副層１ ブロードキャスト・チャネル	４章	多重アクセス・プロトコルを理解し データ・レートを計算できる
05/02	第７回	メディア・アクセス副層２ 無線 LAN，Bluetooth, RFID	４章	個別のプロトコル・スタックを理解し データリンク層スイッチングを理解
05/09	第８回	理解度確認総合演習 (中間試験) 第１回から第７回までの内容の演習形式による確認		第１回から第７回までの理解度確認と 到達度自己評価

データ・リンク層


コンピュータ
アプリケーション
オペレーティング・システム
ネットワーク
ドライバ
リンク
リンク
物理
ネットワーク・インタフェース・
カード（NIC）
ケーブル（媒体）


Device Manager
File Action View Help
TEST-PC
Batteries
Computer
Disk drives
Display adapters
DVD/CD-ROM drives
IDE ATA/ATAPI controllers
IEEE 1394 Bus host controllers
Keyboards
Mice and other pointing devices
Modems
Monitors
Network adapters
Intel(R) PRO/Wireless 3945ABG Network Connection
Realtek RTL8139/810x Family Fast Ethernet NIC
PCMCIA adapters
Processors
SD host adapters
Sound, video and game controllers
Storage controllers
System devices
Universal Serial Bus controllers
Update Driver Software...
Disable
Uninstall
Scan for hardware changes
Properties

データ・リンク層の仮定



1. 送るべきデータは無限にあり、常に用意されている
2. コンピュータはクラッシュしない
3. ペイロードは全てデータ
4. チェックサムを計算できる仕組みが用意されている

protocol.h

```
#define MAX_PKT 1024                                /* パケットの大きさを決める */

typedef enum {false, true} boolean;                 /* 論理型 */
typedef unsigned int seq_nr;                        /* 順序番号または確認通知番号 */
typedef struct {unsigned char data[MAX_PKT];} packet; /* パケットの定義 */
typedef enum {data, ack, nak} frame_kind; /* frame_kindの定義 */

typedef struct {                                    /* この層ではフレームが転送される */

  frame_kind kind;                                  /* このフレームの種類は? */
  seq_nr seq;                                       /* 順序番号 */
  seq_nr ack;                                       /* 確認通知番号 */
  packet info;                                      /* ネットワーク層のパケット */
 } frame;
```

seq_nr : フレームの番号

packet : ネットワーク層とデータ・リンク層の間で交換される情報の単位

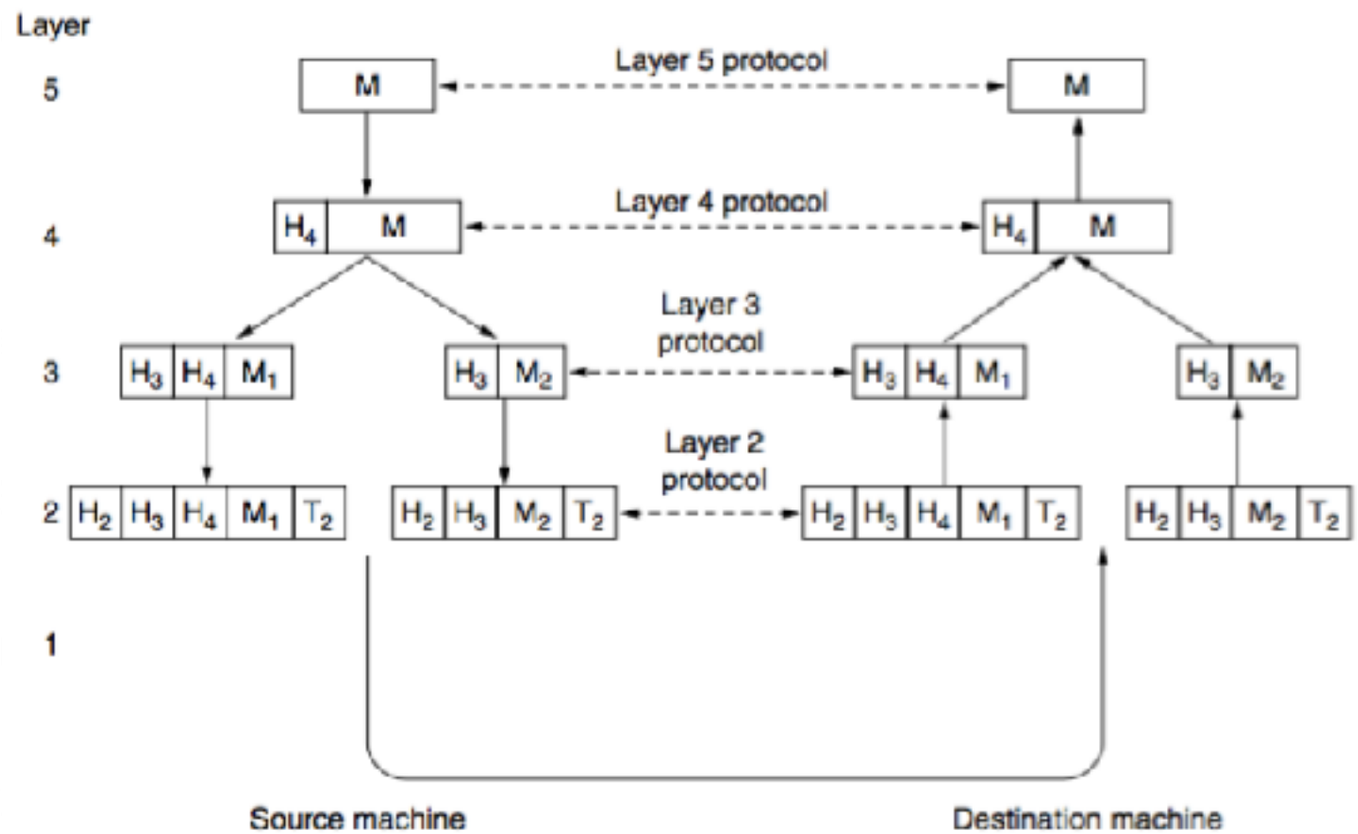
frame : 最初の３つが制御情報、最後の１つが送りたいデータ

kind : フレーム中にデータがあるかどうか

protocol.h

```
/* イベントが起こるのを待つ。イベントのタイプをeventに返す */
void wait_for_event(event_type *event);
```
event = {*timeout*, *cksum_err*, *frame_arrival*, *network_layer_ready*}

```
/* チャネル上に伝送すべきパケットをネットワーク層から取り出す */
void from_network_layer(packet *p);

/* 入力したフレームからの情報をネットワーク層に届ける */
void to_network_layer(packet *p);

/* 入力フレームを物理層に取りにいき、rにコピーする */
void from_physical_layer(frame *r);

/* フレームを伝送するため物理層に渡す */
void to_physical_layer(frame *s);

/* クロックの動作を開始し、タイムアウト・イベントを有
void start_timer(seq_nr k);

/* クロックを止め、ダイムアウト・イベントを無効にする
void stop_timer(seq_nr k);

/* 補助タイマーを起動し、ack_timeoutイベントを有効
void start_ack_timer(void);

/* 補助タイマーを止め、ack_timeoutイベントを無効に
void stop_ack_timer(void);

/* ネットワーク層がnetwork_layer_readyイベントを起こすことを許可する */
void enable_network_layer(void);

/* ネットワーク層がnetwork_layer_readyイベントを起こすことを禁止する */
void disable_network_layer(void);

/* マイクロincはインライン展開される。kを循環的に増やす */
#define inc(k) if(k < MAX_SEQ) k = k + 1; else k = 0
```



ユートピア的単方向プロトコル

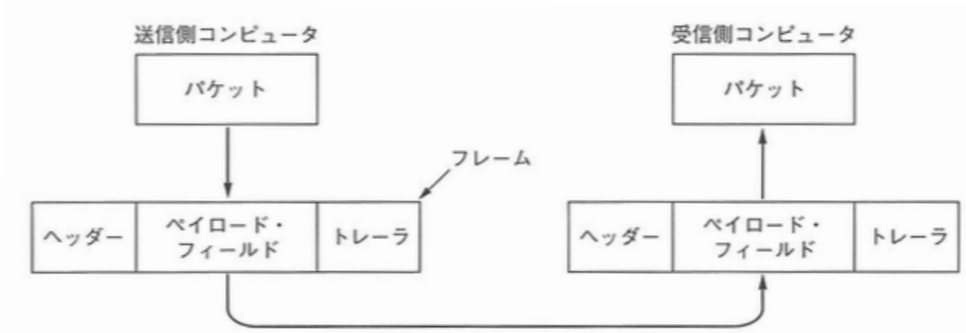
仮定

1. ネットワーク層は送受信側ともに準備できている
2. 通信チャネルはフレームを壊したり失ったりしない
3. データは１方向にのみ伝送される
4. 処理時間は無視できる
5. 無限のバッファ領域が利用可能である

フロー制御不要

誤り訂正不要

ユートピア的単方向プロトコル

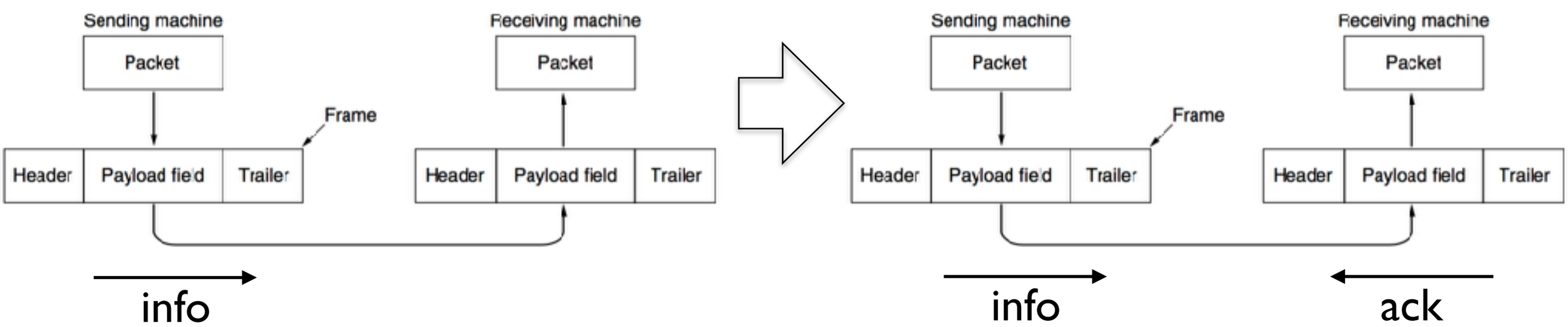



```
typedef enum {frame_arrival} event_type;
#include “protocol.h”

void sender1(void) {
  frame s;                          /* 出力フレームのためのバッファ */

  packet buffer;                    /* 出力パケットのためのバッファ */

  while (true) {
    from_network_layer(&buffer);    /* 送信すべきものを取ってくる */

    s.info = buffer;                /* 伝送のためSにコピーする */

    to_physical_layer(&s);          /* 送信する */

  }
}

void receiver1(void) {
  frame r;
  event_type event;                 /* waitにより設定されるが、ここでは用いない */

  while (true) {
    wait_for_event(&event);         /* 唯一の可能性はframe_arrival */

    from physical layer(&r);        /* 入力したフレームを取ってくる */

    to_network_layer(&r.info);      /* ネットワーク層にデータを渡す */

  }
}
```

誤りのないチャネル用の単方向逐次確認プロトコル

仮定

1. ネットワーク層は送受信側ともに準備できている
2. 通信チャネルはフレームを壊したり失ったりしない
3. データは１方向にのみ伝送される
4. 処理時間は無視できる
5. ~~無限のバッファ領域が利用可能である~~



誤りのないチャネル用の単方向逐次確認プロトコル

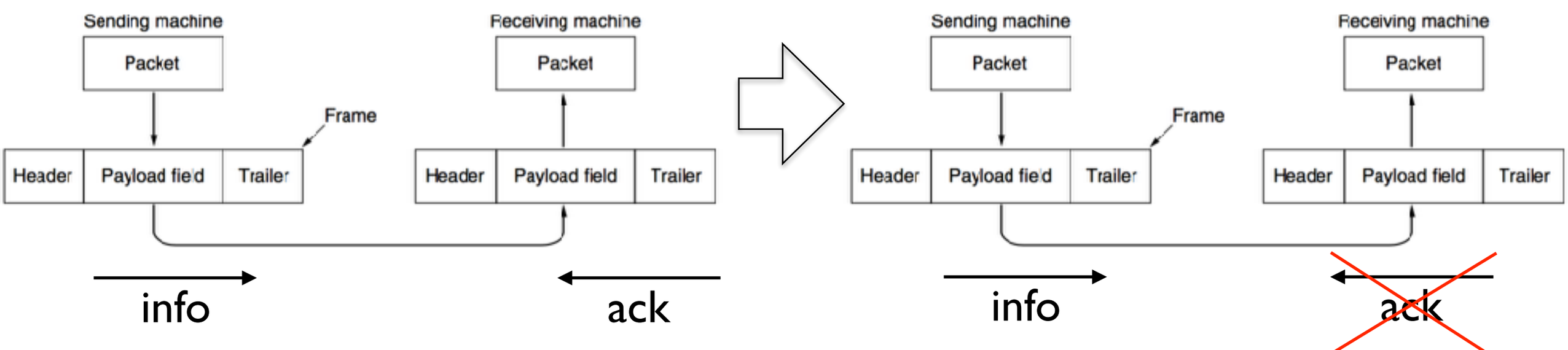
```
typedef enum {frame_arrival} event_type;
#include “protocol.h"

void sender2(void) {
  frame s;                          /* 出力フレームのためのバッファ */
  packet buffer;                    /* 出力パケットのためのバッファ */
  event_type event;                 /* frame_arrivalが唯一の可能性 */
  while (true) {
    from_network_layer(&buffer);    /* 送信すべきものを取ってくる */
    s.info = buffer;                /* 伝送のためsにコピーする */
    to_physical_layer(&s);          /* 送信する */
    wait_for_event(&event);         /* 許可をもらうまで先に進まない */
  }
}

void receiver2(void) {
  frame r, s;                       /* フレームのためのバッファ */
  event_type event;                 /* frame_arrivalが唯一の可能性 */
  while (true) {
    wait_for_event(&event);         /* frame_arrivalが唯一の可能性 */
    from_physical_layer(&r);        /* 入力フレームを取ってくる */
    to_network_layer(&r.info);      /* ネットワーク層にデータを渡す */
    to_physical_layer(&s);          /* 送信者を起こすためのダミーフレームを送る */
  }
}
```

雑音があるチャネル用の単方向逐次確認プロトコル

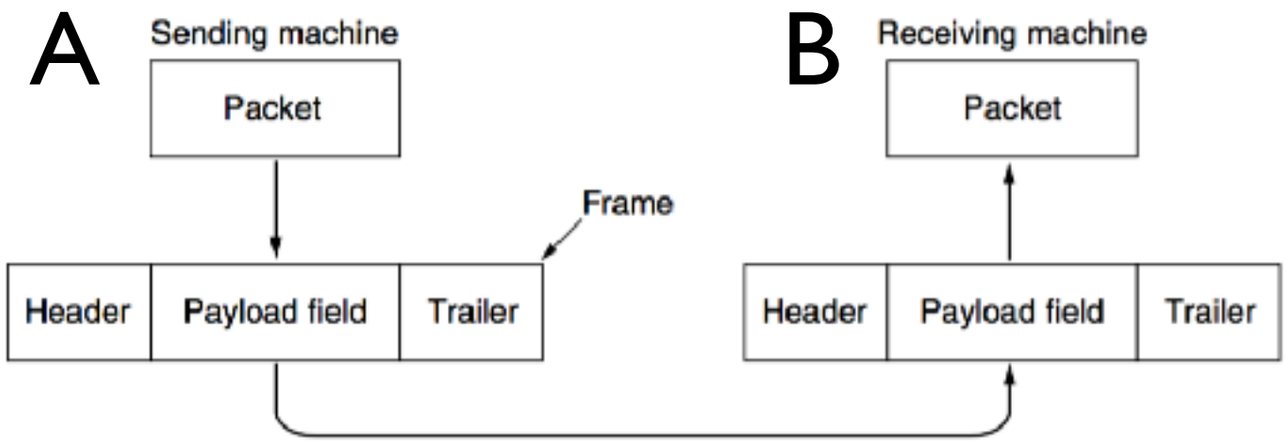
仮定

1. ネットワーク層は送受信側ともに準備できている
2. ~~通信チャネルはフレームを壊したり失ったりしない~~
3. データは１方向にのみ伝送される
4. 処理時間は無視できる
5. ~~無限のバッファ領域が利用可能である~~

タイマーが必要



雑音があるチャネル用の単方向逐次確認プロトコル



タイマーだけでは不十分

1. AがBにフレームを送信
2. Bがフレームを受信
3. BがAに確認通知を送信
4. 確認通知が届かない
5. Aがタイムアウト
6. Aがフレームを再送
7. Bがフレームを二重に受信 ⟶ エラー

```
typedef struct {                          /* この層ではフレームが転送される */
  frame_kind kind;                        /* このフレームの種類は? */
  seq_nr seq;                             /* 順序番号 */
  seq_nr ack;                             /* 確認通知番号 */
  packet info;                            /* ネットワーク層のパケット */
} frame;
```

自動再送要求：ARQ (Automatic Repeat reQuest)　もしくは

再送付き肯定確認通知：PAR (Positive Acknowledgement with Retransmission)

Simplex Stop-and-Wait Protocol with Error

(課題７)

```
#define MAX SEQ 1                          /* プロトコル 3 では１でなければならない */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void) {
  seq_nr next_frame_to_send;               /* 次に出力するフレームの順序番号 */
  frame s;                                 /* 作業変数 */
  packet buffer;                           /* 出力パケットのためのバッファ */
  event_type event;
  next_frame_to_send = 0;                  /* 出力順序番号の初期化 */
  from_network_layer(&buffer);             /* 最初のパケットを取り出す */
  while (true) {
    s.info = buffer;                       /* 伝送するフレームを作成する */
    s.seq = next_frame_to_send;            /* フレームに順序番号を挿入する */
    to_physical_layer(&s);                 /* 送信する */
    start_timer(s.seq);                    /* 応答に時間がかかりすぎると、タイムアウトとなる */
    wait_for_event(&event);                /* frame arrival, cksum_err, timeout */
    if (event == frame_arrival) {
      from_physical_layer(&s);             /* 確認通知を得る */
      if (s.ack == next_frame_to_send) {
        stop_timer(s.ack);                 /* タイマーを止める */
        from_network_layer(&buffer);       /* 次に送るものを得る */
        inc(next_frame_to_send);           /* next_frame_to_sendを反転する */
      }
    }
  }
}
```

Simplex Stop-and-Wait Protocol with Error

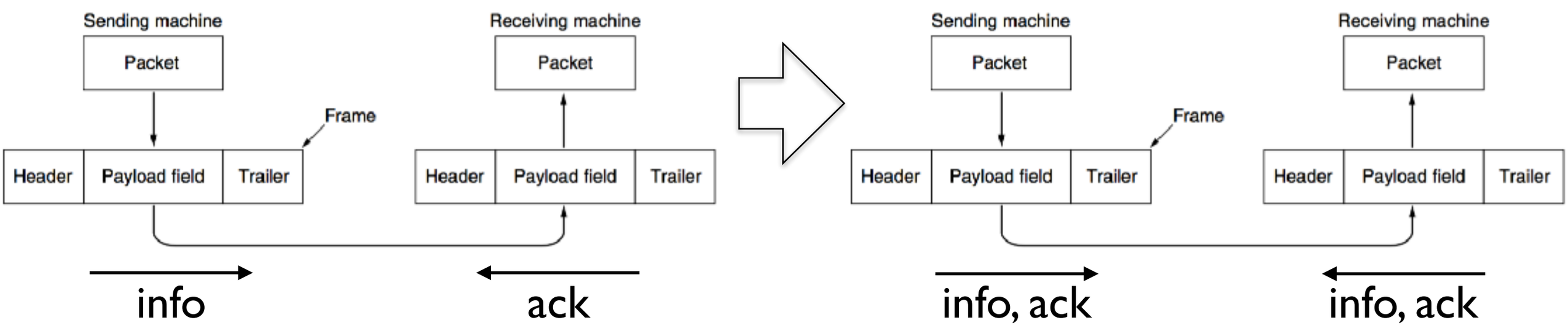
```
void receiver3(void) {
  seq_nr frame_expected;
  frame r, s;
  event_type event;
  frame_expected = 0;
  while (true) {
    wait_for_event(&event);              /* 可能性は：frame_arrival, cksum_err */
    if (event == frame_arrival) {        /* 有効なフレームが到着した */
      from_physical_layer(&r);           /* 新しく届いたフレームを取ってくる */
      if (r.seq == frame_expected) {     /* これが待っていたもの */
        to_network_layer(&r.info);       /* ネットワーク層にデータを渡す */
        inc(frame_expected);             /* 次回はもう一つの順序番号を期待する */
      }
      s.ack = 1 - frame_expected;        /* どのフレームを確認通知するのか告げる */
      to_physical_layer(&s);             /* 確認通知を送る */
    }
  }
}
```

```
typedef struct {                                  /* この層ではフレームが転送される */

  frame_kind kind;                                /* このフレームの種類は? */
  seq_nr seq; 0,1,0,1,0,1,...                     /* 順序番号 */
  seq_nr ack; 0,1,0,1,0,1,...                     /* 確認通知番号 */
  packet info;                                    /* ネットワーク層のパケット */
} frame;
```

One Bit Sliding Window Protocol (Duplex Stop-and-Wait)

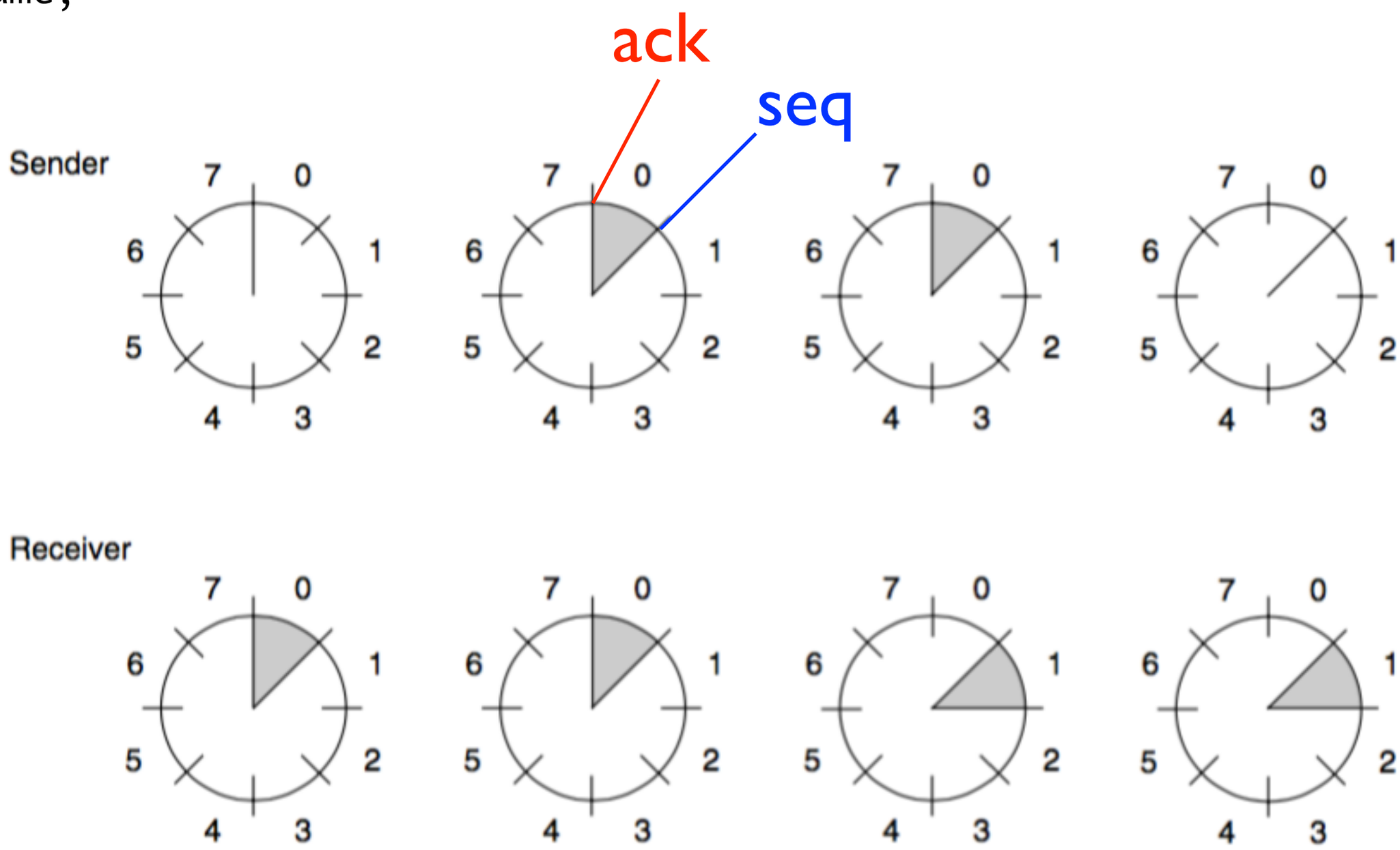
仮定

1. ネットワーク層は送受信側ともに準備できている
2. ~~通信チャネルはフレームを壊したり失ったりしない~~
3. ~~データは1方向にのみ伝送される~~
4. 処理時間は無視できる
5. ~~無限のバッファ領域が利用可能である~~

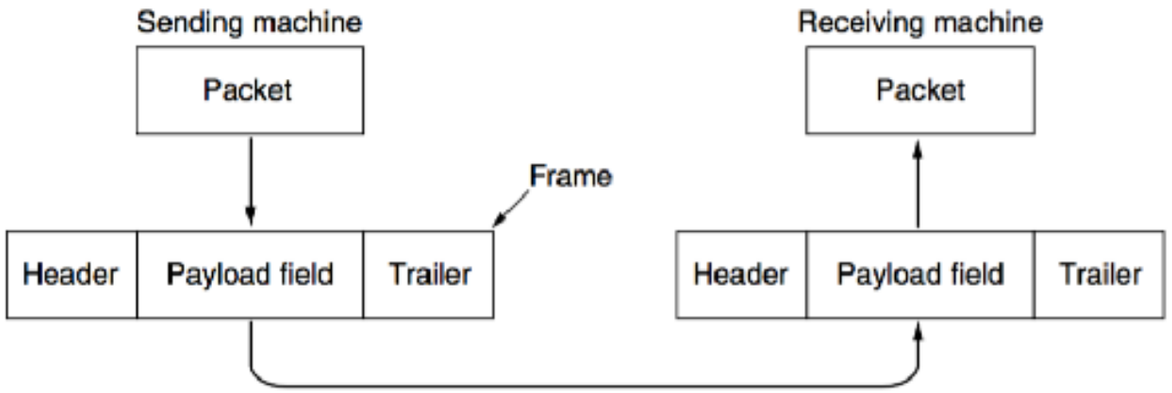


3 Bit Sliding Window Protocol

```
typedef struct {                                  /* この層ではフレームが転送される */

  frame_kind kind;                                /* このフレームの種類は? */
  seq_nr seq; 0,1,2,3,4,5,6,7,0,...               /* 順序番号 */
  seq_nr ack; 0,1,2,3,4,5,6,7,0,...               /* 確認通知番号 */
  packet info;                                    /* ネットワーク層のパケット */
} frame;
```

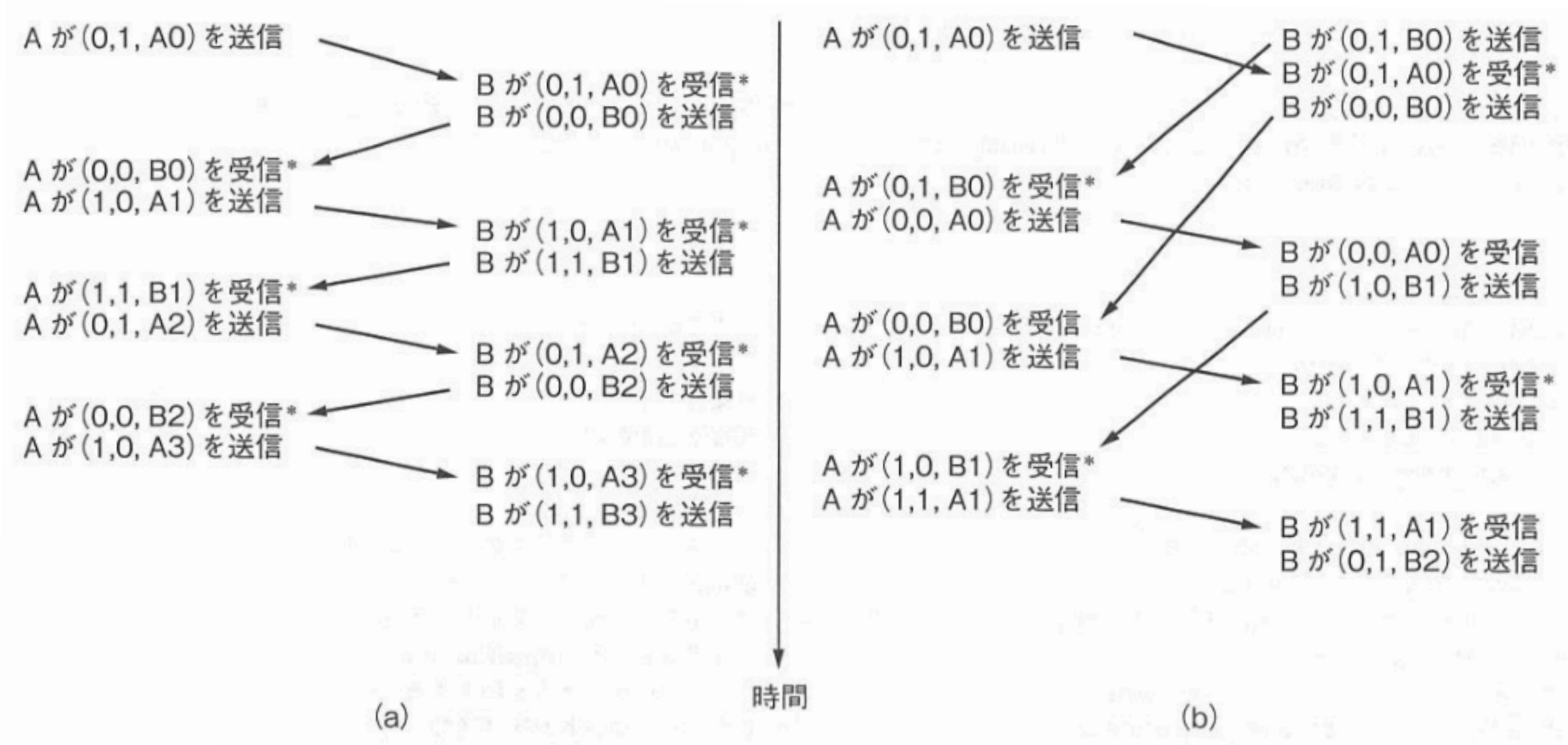


One Bit Sliding Window Protocol




```
#define MAX SEQ 1 /* プロトコル4は１でなければならない */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void protocol4 (void) {
  seq_nr next_frame_to_send;            /* 0または1だけ */
  seq_nr frame_expected;                /* 0または1だけ */
  frame r, s;                           /* 作業変数 */
  packet buffer;                        /* 現在送信中のパケット */
  event_type event;
  next_frame_to_send = 0;               /* 出力ストリーム上の次のフレーム */
  frame_expected = 0;                   /* 次に届く予定のフレーム */
  from_network_layer(&buffer);          /* ネットワーク層からのパケットを取り出す */
  s.info = buffer;                      /* 最初のフレームの送信準備 */
  s.seq = next_frame_to_send;           /* フレームに順序番号を挿入する */
  s.ack = 1 – frame_expected;           /* ピギーバックされた確認通知 */
  to_physical_layer(&s);                /* フレームを送信する */
  start_timer(s.seq);                   /* タイマーの動作を開始 */
  while (true) {
    wait_for_event(&event);             /* frame arrival, cksum err, or timeout */
    if (event == frame_arrival) {       /* フレームが損傷なしに到着した */
      from_physical_layer(&r);          /* それを取ってくる */
      if (r.seq == frame expected) {    /* 入力フレーム・ストリームを処理する */
        to_network_layer(&r.info);      /* ネットワーク層にパケットを渡す */
        inc(frame_expected);            /* 次に期待する順序番号を反転する */
      }
      if (r.ack == next_frame_to_send) { /* 出力フレーム・ストリームを処理する */
        stop_timer(r.ack);              /* タイマーを止める */
        from_network_layer(&buffer);    /* ネットワーク層から新しいパケットを取り出す */
        inc(next_frame_to_send);        /* 送信者の順序番号を反転する */
      }
    }
    s.info = buffer;                    /* 出力フレームを作成する */
    s.seq = next_frame_to_send;         /* 順序番号を挿入する */
    s.ack = 1 – frame_expected;         /* 最後に受信したフレームの順序番号 */
    to_physical_layer(&s);              /* フレームを送信する */
    start_timer(s.seq);                 /* タイマーの動作を開始する */
  }
}
```

One Bit Sliding Window Protocol



同じフレームが何度も送信されることがある

一括再送(Go-back-N)プロトコル

仮定

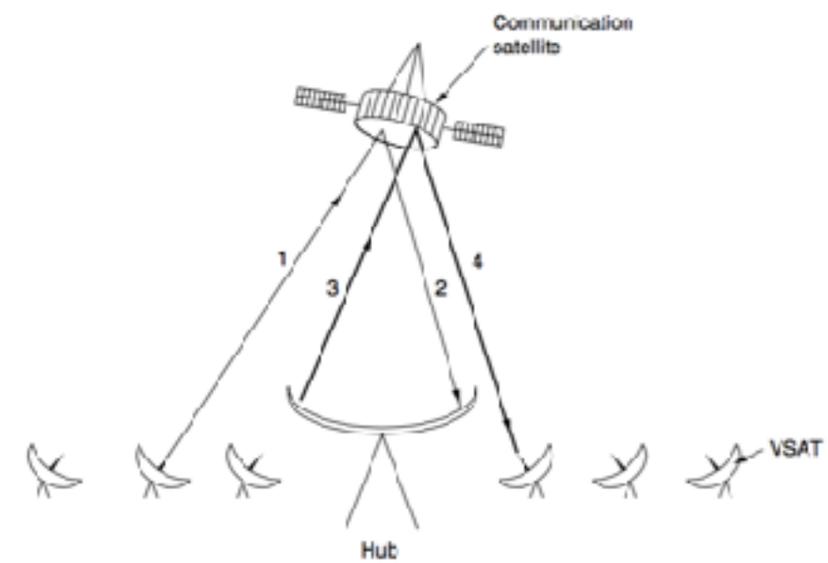
~~1. ネットワーク層は送受信側ともに準備できている~~

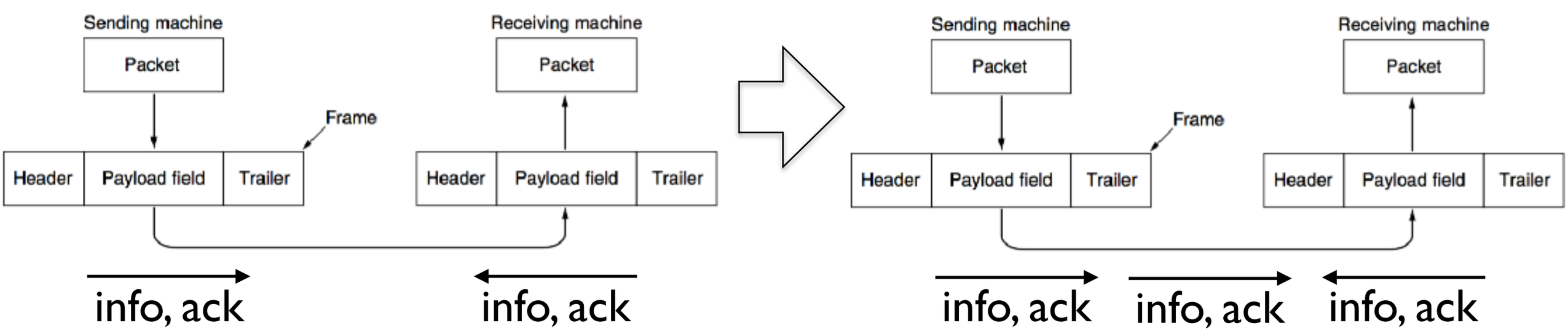
~~2. 通信チャネルはフレームを壊したり失ったりしない~~

~~3. データは1方向にのみ伝送される~~

~~4. 処理時間は無視できる~~

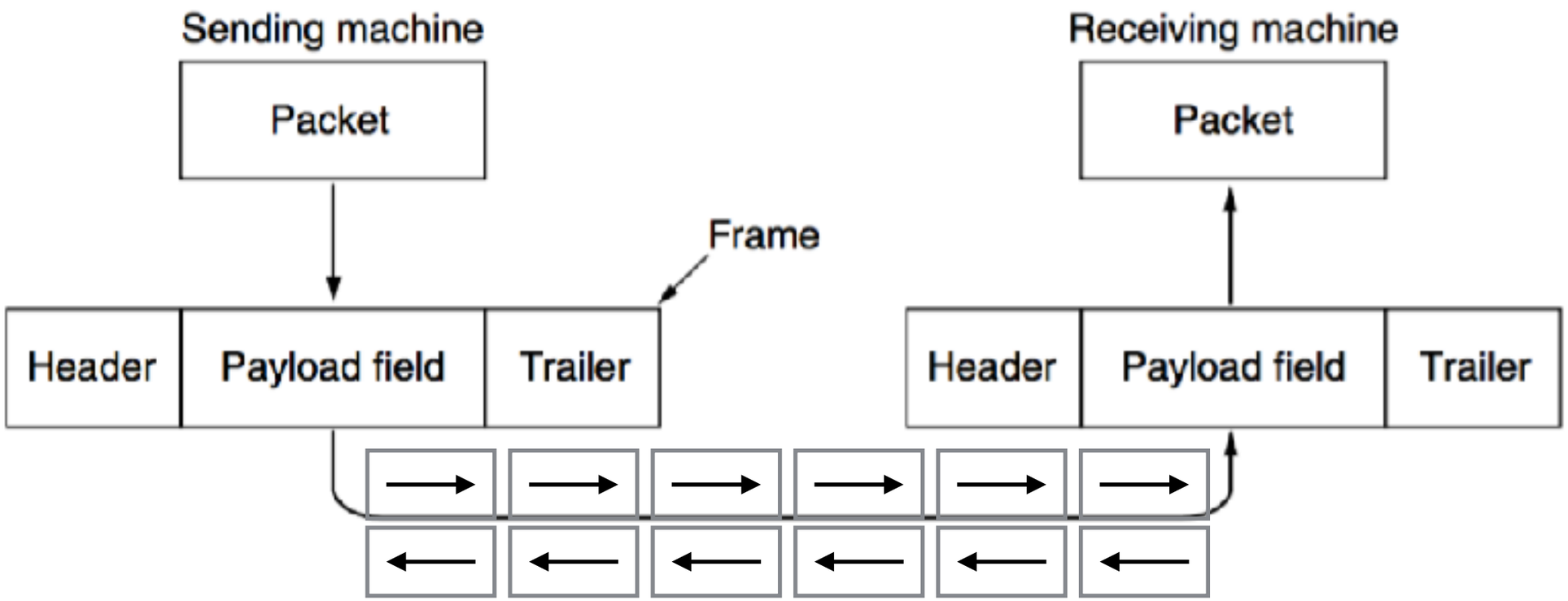
~~5. 無限のバッファ領域が利用可能である~~





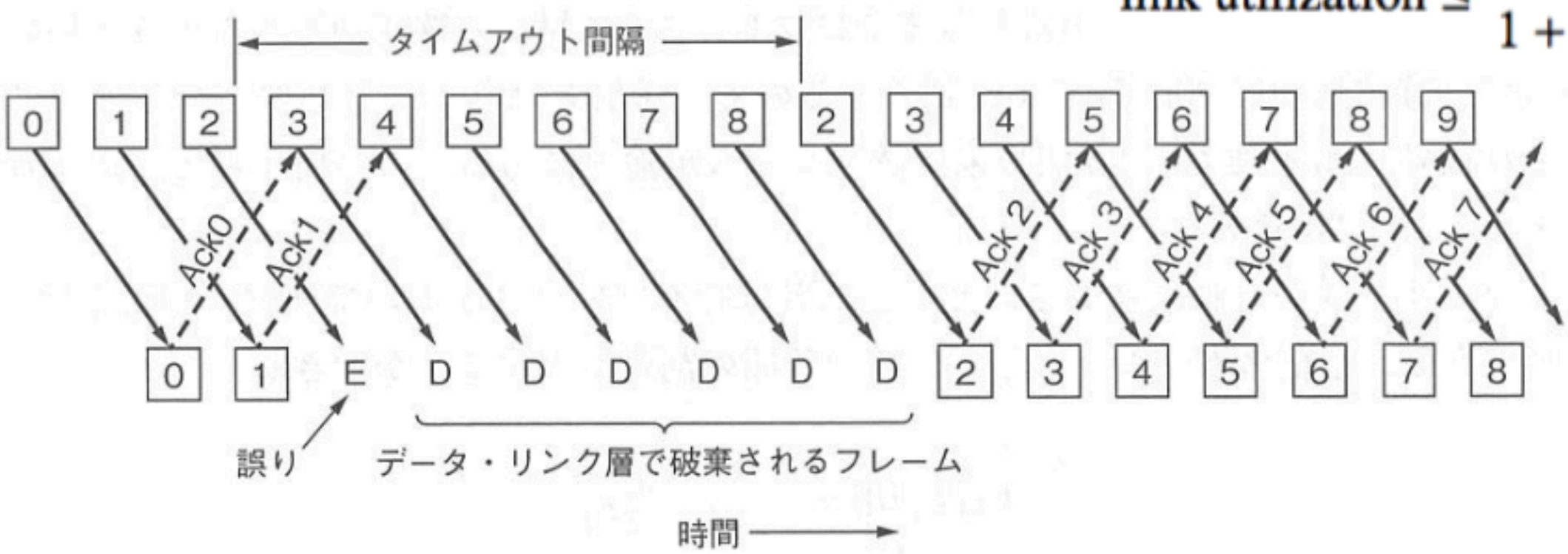
一括再送(Go-back-N)プロトコル

BD (Bandwidth Delay product) = (Bandwidth) x (One way transit time)



w=2BD+1

$$\text{link utilization} \le \frac{w}{1+2BD}$$



一括再送(Go-back-N)プロトコル

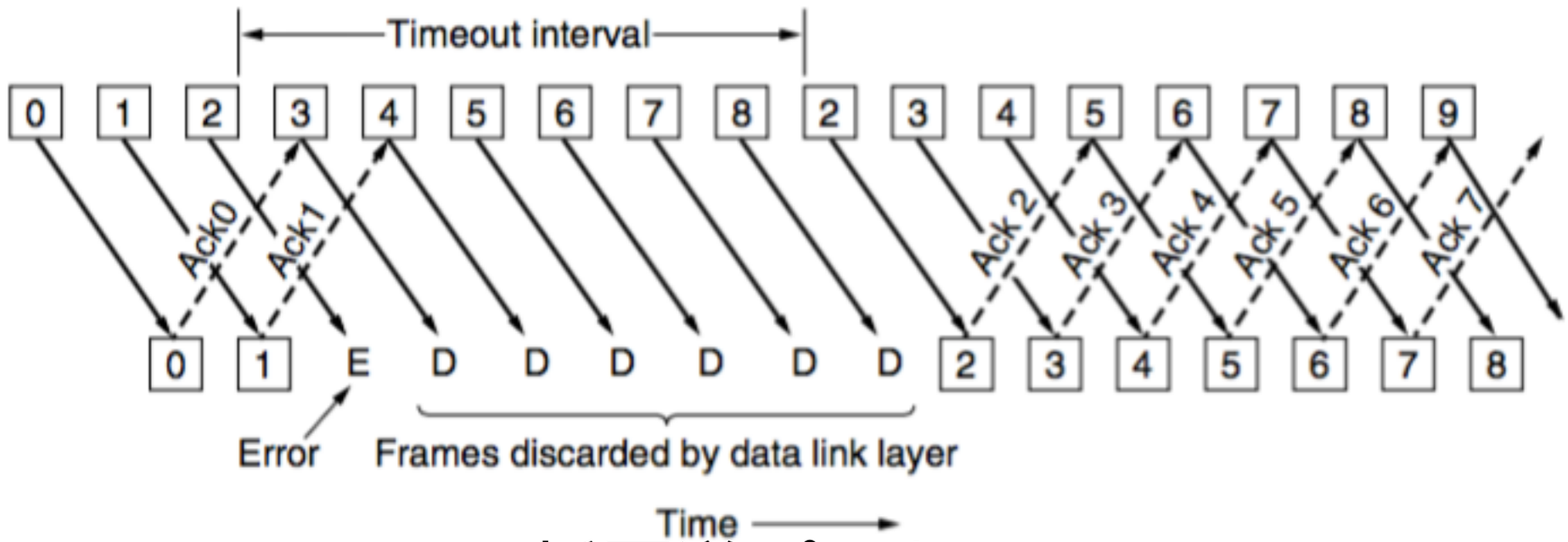
(課題８)

```
#define MAX_SEQ 7
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) { /* 巡回的にa <= b < cであればtrueを返す。そうでなければfalseを返す。 */
  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[]) { /* データ・フレームを作成して送信する */
  frame s;                                                    /* 作業変数 */
  s.info = buffer[frame_nr];                                  /* パケットをフレームに挿入する */
  s.seq = frame_nr;                                           /* 順序番号をパケットに挿入する */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);         /* ピギーバックされた確認通知 */
  to_physical_layer(&s);                                      /* フレームを送信する */
  start_timer(frame_nr);                                      /* タイマーの動作を起動する */
}

void protocol5(void) {
  seq_nr next_frame_to_send;                                  /* MAX SEQ > 1（出力ストリームで用いる） */
  seq_nr ack_expected;                                        /* まだ未確認の最も古いフレーム */
  seq_nr frame_expected;                                      /* 入力ストリームの次に期待するフレーム */
  frame r;                                                    /* 作業変数 */
  packet buffer[MAX_SEQ+1];                                   /* 出力ストリームのためのバッファ */
  seq_nr nbuffered;                                           /* 現在使用中の出力バッファの数 */
  seq_nr i;                                                   /* バッファの入れるの中を指すのに用いる */
  event_type event;
  enable_network_layer();                                     /* network_layer_readyイベントを許可する */
  ack_expected = 0;                                           /* 次に期待する入力確認通知 */
  next_frame_to_send = 0;                                     /* 次に出力するフレーム */
  frame_expected = 0;                                         /* 期待する入力フレームの番号 */
  nbuffered = 0;                                              /* 最初はバッファされているパケットはない */
```

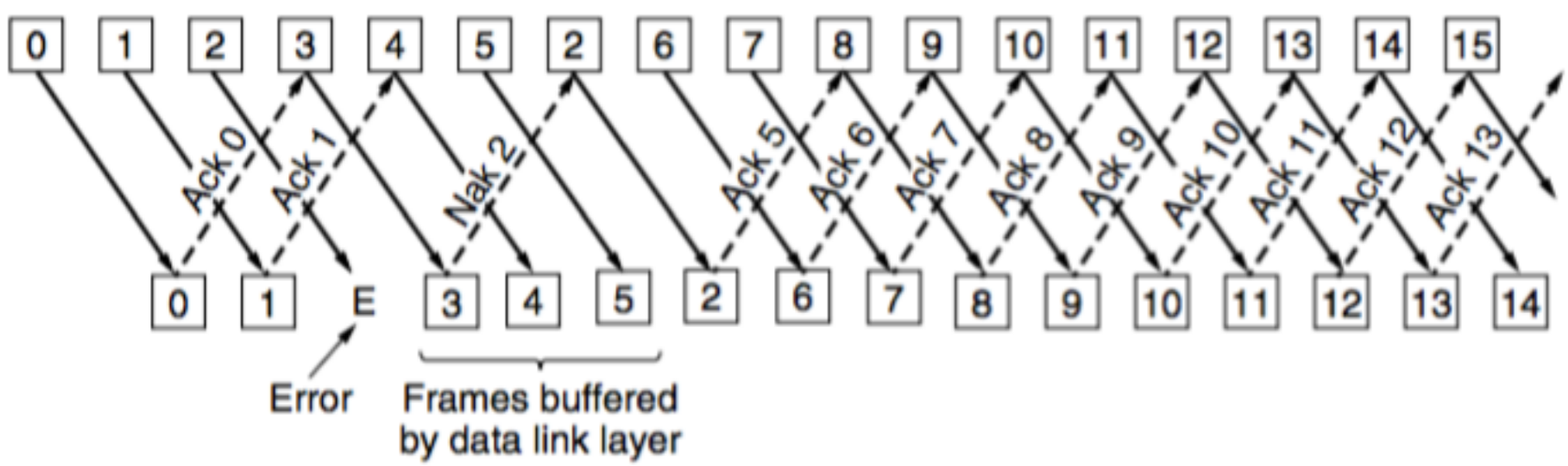
Go-back-N protocol

```
while (true) {
  wait_for_event(&event);                                          /* 四つの可能性がある: 上記のevent_typeを参照 */
  switch(event) {
    case network_layer_ready:                                      /* ネットワーク層に送信すべきパケットがある */
      from_network_layer(&buffer[next_frame_to_send]);             /* 新しいパケットを取り出す */
      nbuffered = nbuffered + 1;                                   /* 送信者のウィンドウを広げる */
      send_data(next_frame_to_send, frame_expected, buffer);       /* フレームを送信する */
      inc(next_frame_to_send);                                     /* 送信者のウィンドウの上端を進める */
      break;
    case frame_arrival:                                            /* フレームまたは制御フレームが到着した */
      from_physical_layer(&r);                                     /* 物理層から入力フレームを取ってくる */
      if (r.seq == frame_expected) {                               /* フレームが期待したものなら受理する */
        to_network_layer(&r.info);                                 /* ネットワーク層にパケットを渡す */
        inc(frame_expected);                                       /* 受信者のウィンドウの下端を進める */
      }
      while (between(ack_expected, r.ack, next_frame_to_send)) {   /* ピギーバックされた確認通知を処理する */
        nbuffered = nbuffered - 1;                                 /* バッファされたフレームが一つ減る */
        stop_timer(ack_expected);                                  /* フレームが損傷なく到着。タイマーを停止 */
        inc(ack_expected);                                         /* 送信者のウィンドウを縮める */
      }
      break;
    case cksum_err:
      break;                                                       /* 正しくないフレームは無視する */
    case timeout:                                                  /* 問題発生。全ての未確認フレームを再送する */
      next_frame_to_send = ack_expected;                           /* ここから再送開始 */
      for (i = 1; i <= nbuffered; i++) {
        send_data(next_frame_to_send, frame_expected, buffer);     /* フレームを再送 */
        inc(next_frame_to_send);                                   /* 次のものを送る準備をする */
      }
    }
    if (nbuffered < MAX_SEQ)
      enable_network_layer();
    else
      disable_network_layer();
  }
}
}
```

選択的再送プロトコル



一括再送プロトコル



選択的再送プロトコル

(課題9,10)

```
#define MAX_SEQ 7                                       /* 2^n - 1でなければならない */
#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;                                  /* nakはまだ送信していない */
seq_nr oldest_frame = MAX_SEQ + 1;                      /* 初期値はシミュレータのためだけ */

static boolean between(seq_nr a, seq_nr b, seq_nr c) {
  return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[]) {
  frame s;                                              /* 作業変数 */
  s.kind = fk;                                          /* kindはdata, ack, nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                                     /* 意味があるのはデータ・フレームだけ */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;                        /* フレーム当たり否定確認通知は一つだけ */
  to_physical_layer(&s);                                /* フレームを送信する */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop ack_timer();                                     /* 個別の確認通知フレームの必要なし */
}

void protocol6(void) {
  seq_nr ack_expected;                                  /* 送信者のウィンドウの下端 */
  seq_nr next_frame_to_send;                            /* 送信者のウィンドウの上端 + 1 */
  seq_nr frame_expected;                                /* 受信者のウィンドウの下端 */
  seq_nr too_far;                                       /* 受信者のウィンドウの上端 + 1 */
  int i;                                                /* バッファ・プールへの添え字 */
  frame r;                                              /* 作業変数 */
  packet out_buf[NR_BUFS];                              /* 出力ストリームのためのバッファ */
  packet in_buf[NR_BUFS];                               /* 入力ストリームのためのバッファ */
  boolean arrived[NR_BUFS];                             /* 入力ビットマップ */
  seq_nr nbuffered;                                     /* 現在使用中の出力バッファの数 */
  event_type event;
  enable_network_layer();                               /* 初期化 */
  ack_expected = 0;                                     /* 入力ストリームで次に期待する確認通知 */
  next_frame_to_send = 0;                               /* 次の出力フレームの番号 */
  frame_expected = 0;
  too_far = NR_BUFS;
```

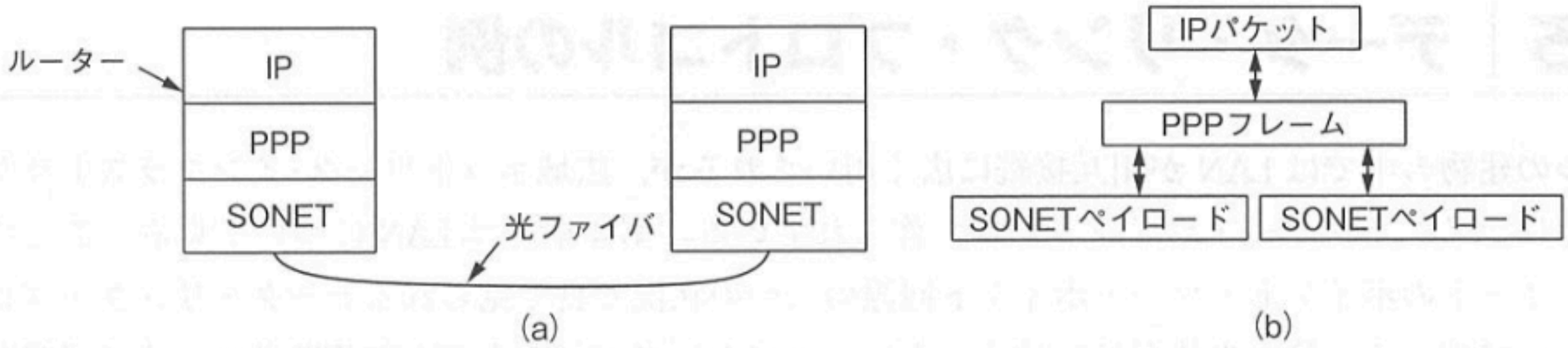
```
nbuffered = 0;                                                    /* 最初はバッファされているパケットはない */
for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
while (true) {
  wait_for_event(&event);                                         /* 五つの可能性がある。上記のevent_typeを参照 */
  switch(event) {
    case network_layer_ready:                                     /* 受理、保存、新しいフレームを送信する */
      nbuffered = nbuffered + 1;                                  /* ウィンドウを広げる */
      from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]);  /* 新しいパケットを取り出す */
      send_frame(data, next_frame_to_send, frame_expected, out_buf); /* フレームを送信する */
      inc(next_frame_to_send);                                    /* ウィンドウの上端を進める */
      break;
    case frame_arrival:                                           /* フレームまたは制御フレームが到着した */
      from_physical_layer(&r);                                    /* 物理層から入力フレームを取り出す */
      if (r.kind == data) {                                       /* 損傷のないフレームが到着した */
        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)) {
          arrived[r.seq % NR_BUFS] = true;                        /* バッファに満の印をつける */
          in_buf[r.seq % NR_BUFS] = r.info;                       /* バッファにデータを挿入する */
          while (arrived[frame_expected % NR_BUFS]) {             /* フレームを渡してウィンドウを進める */
            to_network_layer(&in_buf[frame_expected % NR_BUFS]);
            no_nak = true;
            arrived[frame_expected % NR_BUFS] = false;
            inc(frame_expected);                                  /* 受信者のウィンドウの下端を進める */
            inc(too_far);                                         /* 受信者のウィンドウの上端を進める */
            start_ack_timer();                                    /* 個別の確認通知が必要かどうか調べるため */
          }
        }
      }
      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;                                /* ピギーバックされた確認通知を処理する */
        stop_timer(ack_expected % NR_BUFS);                       /* フレームはそのまま到着した */
        inc(ack_expected);                                        /* 送信者のウィンドウの下端を進める */
      }
      break;
```

(課題9,10)

(課題9,10)

```
      case cksum err:
        if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* 損傷したフレーム */
        break;
      case timeout:
        send_frame(data, oldest_frame, frame_expected, out_buf); /* タイムアウトとなった */
        break;
      case ack_timeout:
        send_frame(ack, 0, frame_expected, out_buf);             /* 確認通知タイムアウト。再送 */
    }
    if (nbuffered < NR_BUFS) enable_network_layer();
    else disable_network_layer();
  }
}
```

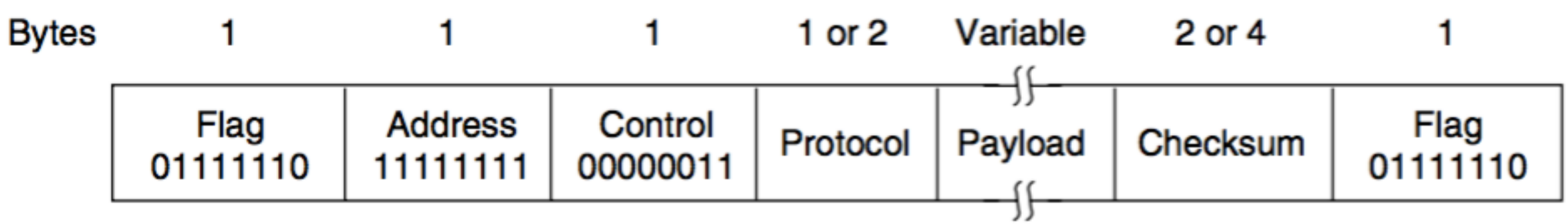
Point-to-Point Protocols (PPP) over SONET



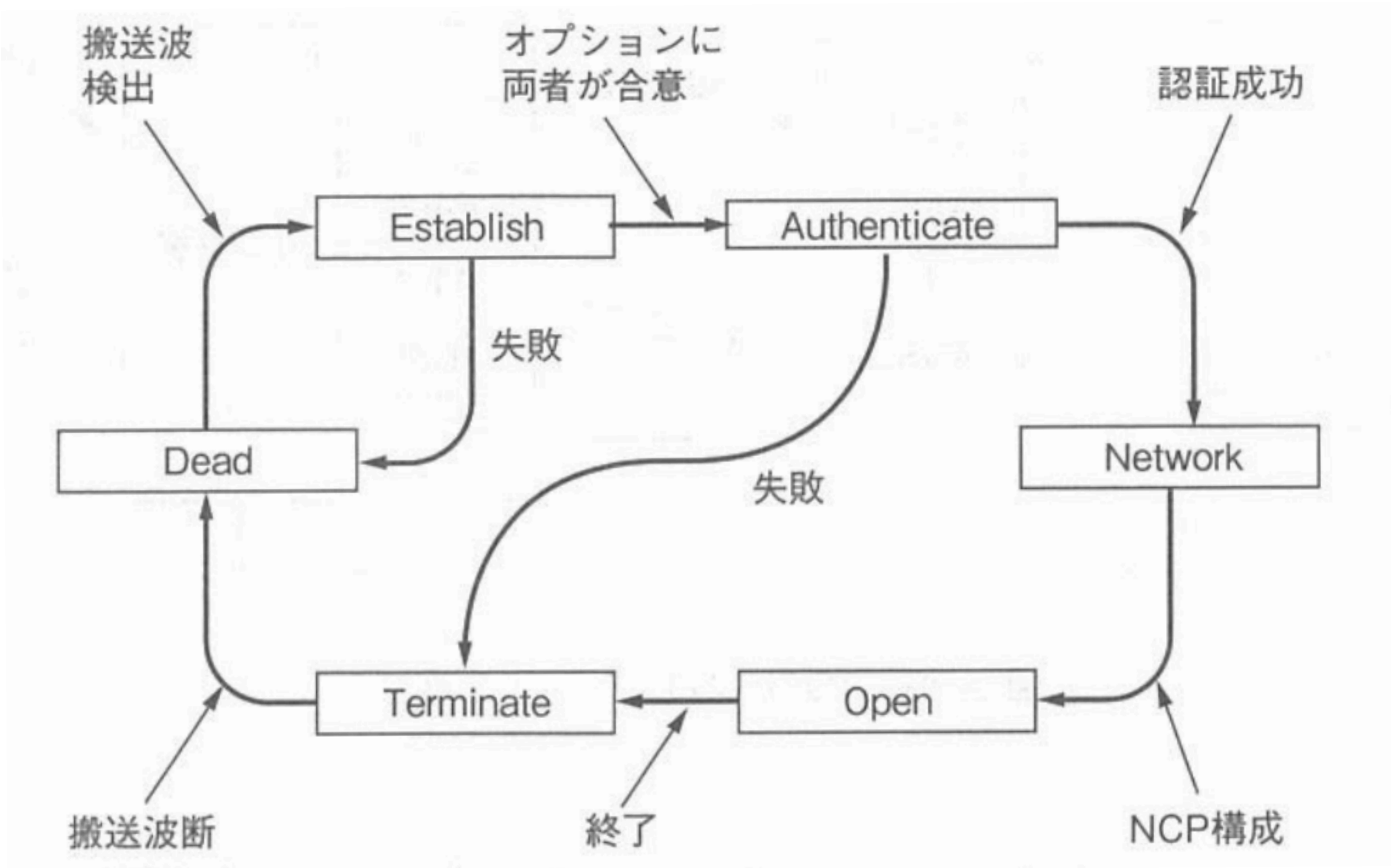
SONET 上のパケット。(a) プロトコル・スタック，(b) フレームの関係

PPPの特徴

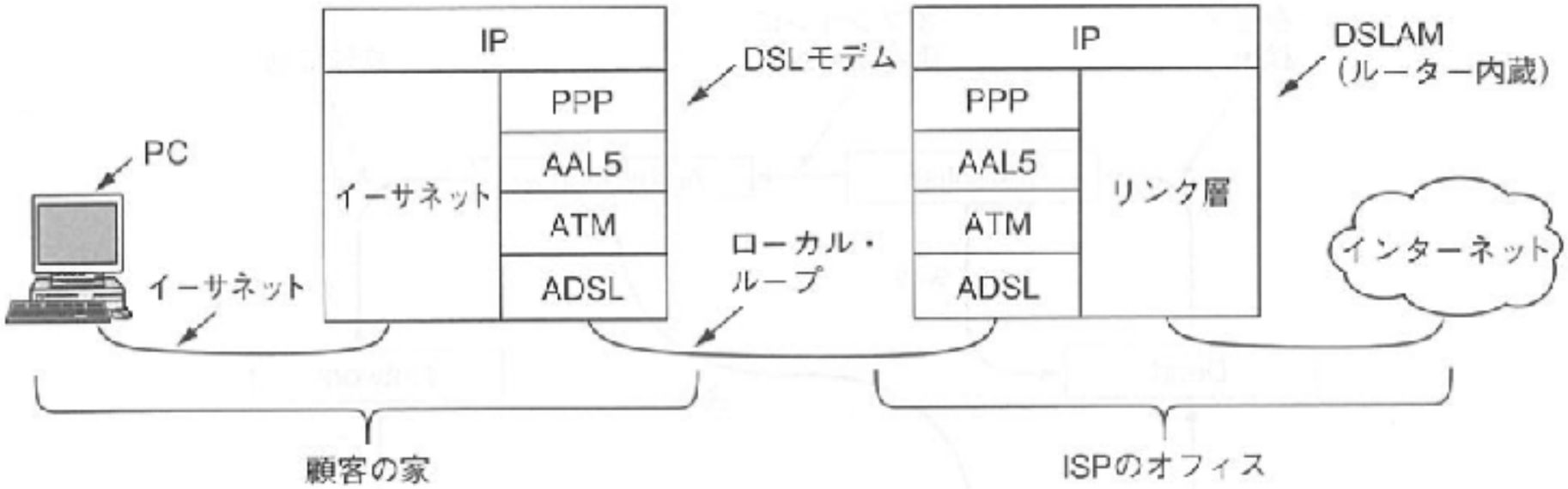
1. あるフレームの終わりと次のフレームの始まりを曖昧性なく示すフレーム方式。フレーム・フォーマットは誤り検出も扱っている。
2. 回線を立ち上げ、検査し、オプションをネゴシエーションし、回線が不要となったときには行儀よく切断するためのリンク制御プロトコル。このプロトコルはLCP(Link Control Protocol:リンク制御プロトコル)と呼ばれている。これは同期・非同期の回線及びバイト指向・ビット指向の符号化をサポートしている。
3. 用いるネットワーク層プロトコルからは独立な方法で、ネットワーク層オプションをネゴシエーションするための方法。サポートするネットワーク層ごとに異なるNCP(Network Control Protocol:ネットワーク制御プロトコル)を持つ方式が採用された。



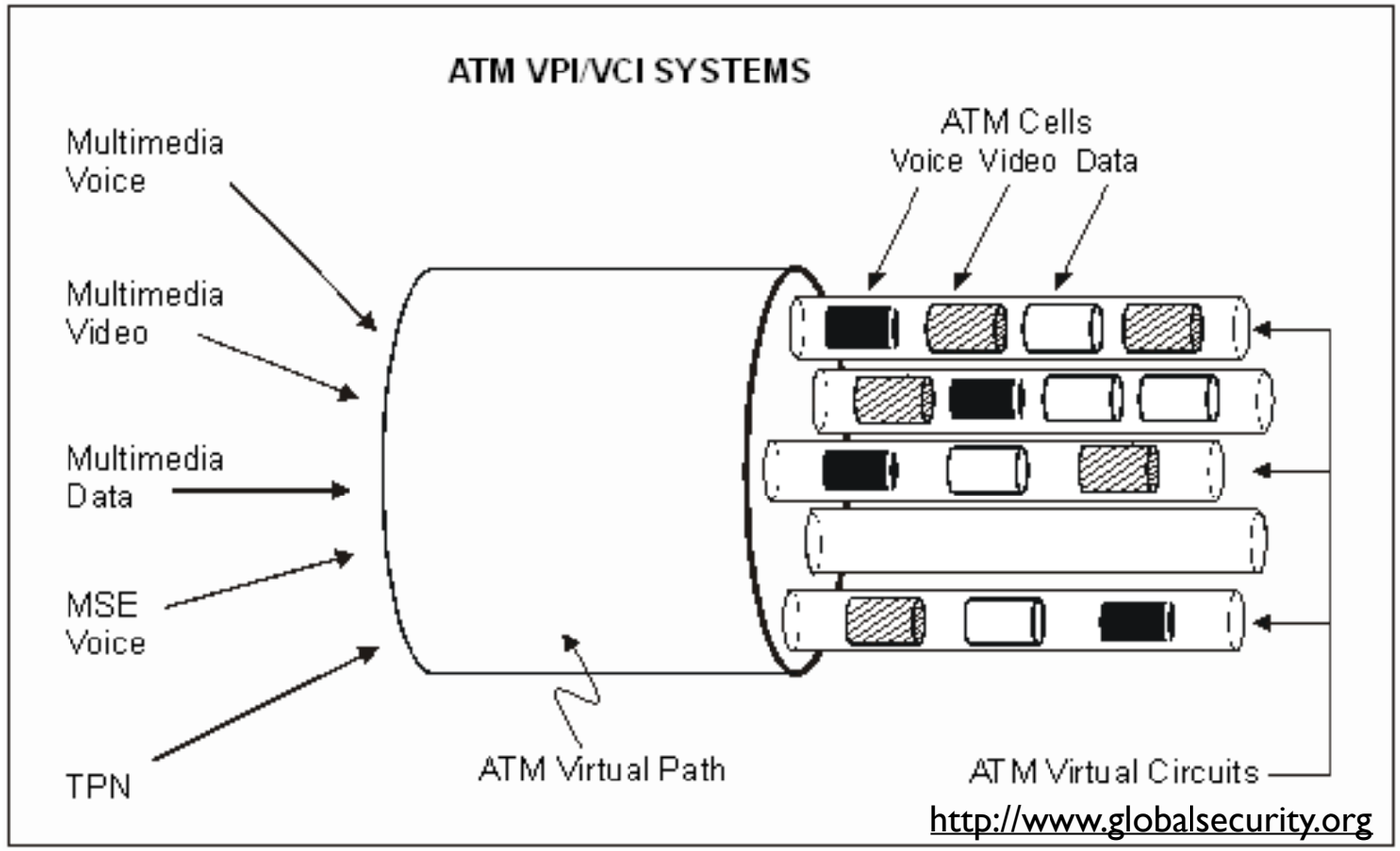
PPP over SONET

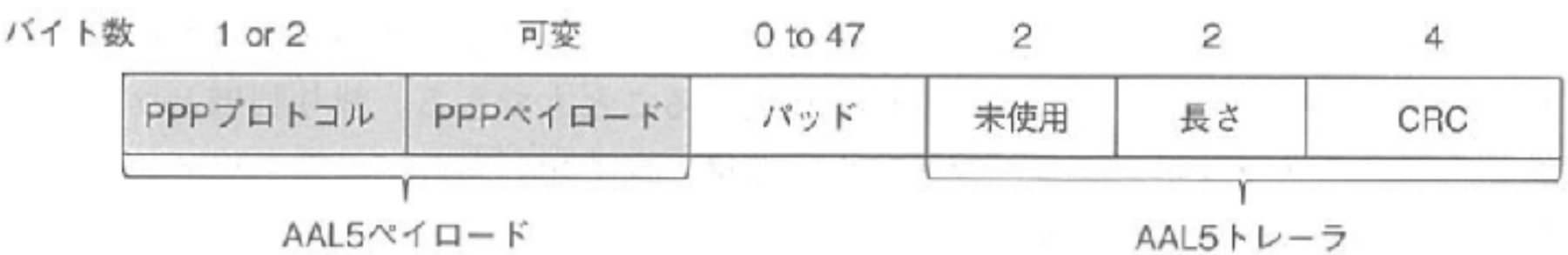


PPP over ADSL



Asynchronous Transfer Mode (ATM)
AAL5 (ATM Adaptation Layer 5)





講義日程

		授業計画		課題
04/11	第１回	計算機ネットワークの基本概念 ハードウェア・ソフトウェア，参照モデル	１章	ネットワークの種類と参照モデルを理解し プロトコル階層と各層の設計課題
04/14	第２回	物理層１ 有線伝送と無線伝送	２章	物理チャネルの特性を理解し データ通信の理論的基礎を理解
04/18	第３回	物理層２ デジタル変調と多重化	２章	ベースバンド伝送と通過帯域伝送， 電話網，携帯電話システムを説明できる
04/21	第４回	データリンク層１ 誤りの検出・訂正	３章	誤りの検出・訂正のしくみを理解し 検出・訂正符号の計算ができる
04/25	第５回	データリンク層２ データリンク・プロトコル	３章	データリンク・プロトコルの種類， 各プロトコルを定量的に評価できる
04/28	第６回	メディア・アクセス副層１ ブロードキャスト・チャネル	４章	多重アクセス・プロトコルを理解し データ・レートを計算できる
05/02	第７回	メディア・アクセス副層２ 無線 LAN，Bluetooth, RFID	４章	個別のプロトコル・スタックを理解し データリンク層スイッチングを理解
05/09	第８回	理解度確認総合演習 (中間試験) 第１回から第７回までの内容の演習形式による確認		第１回から第７回までの理解度確認と 到達度自己評価