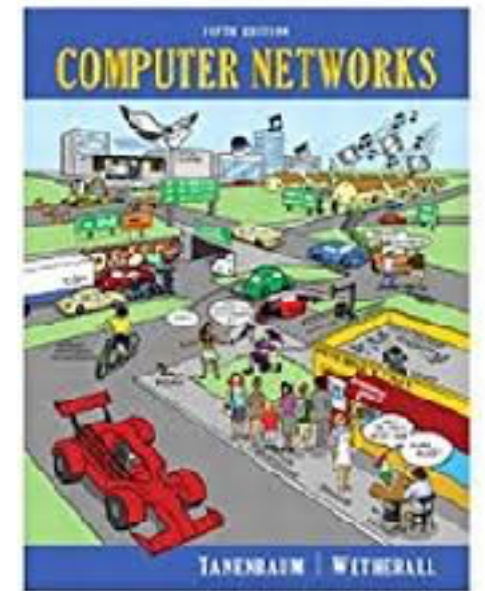
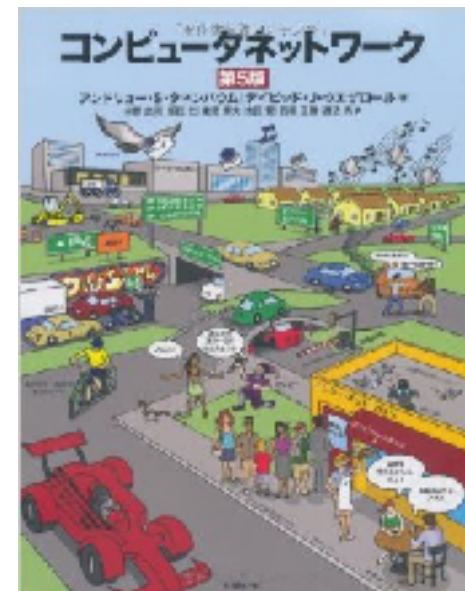


コンピュータネットワーク

開講クォーター: IQ

曜日・時限: 火、金1-2限

講義室: S011

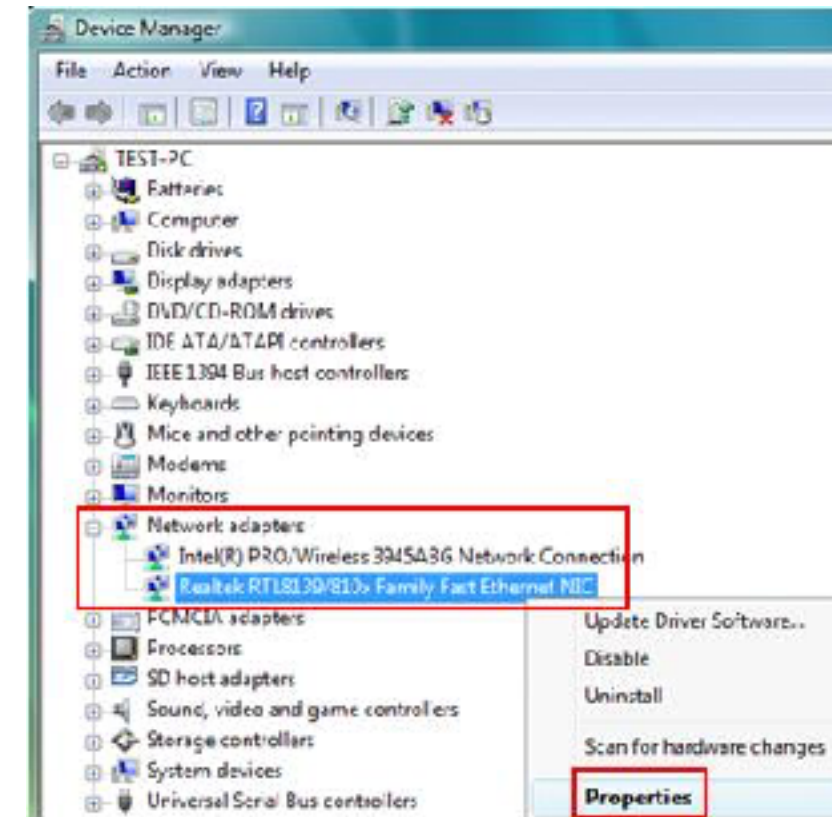
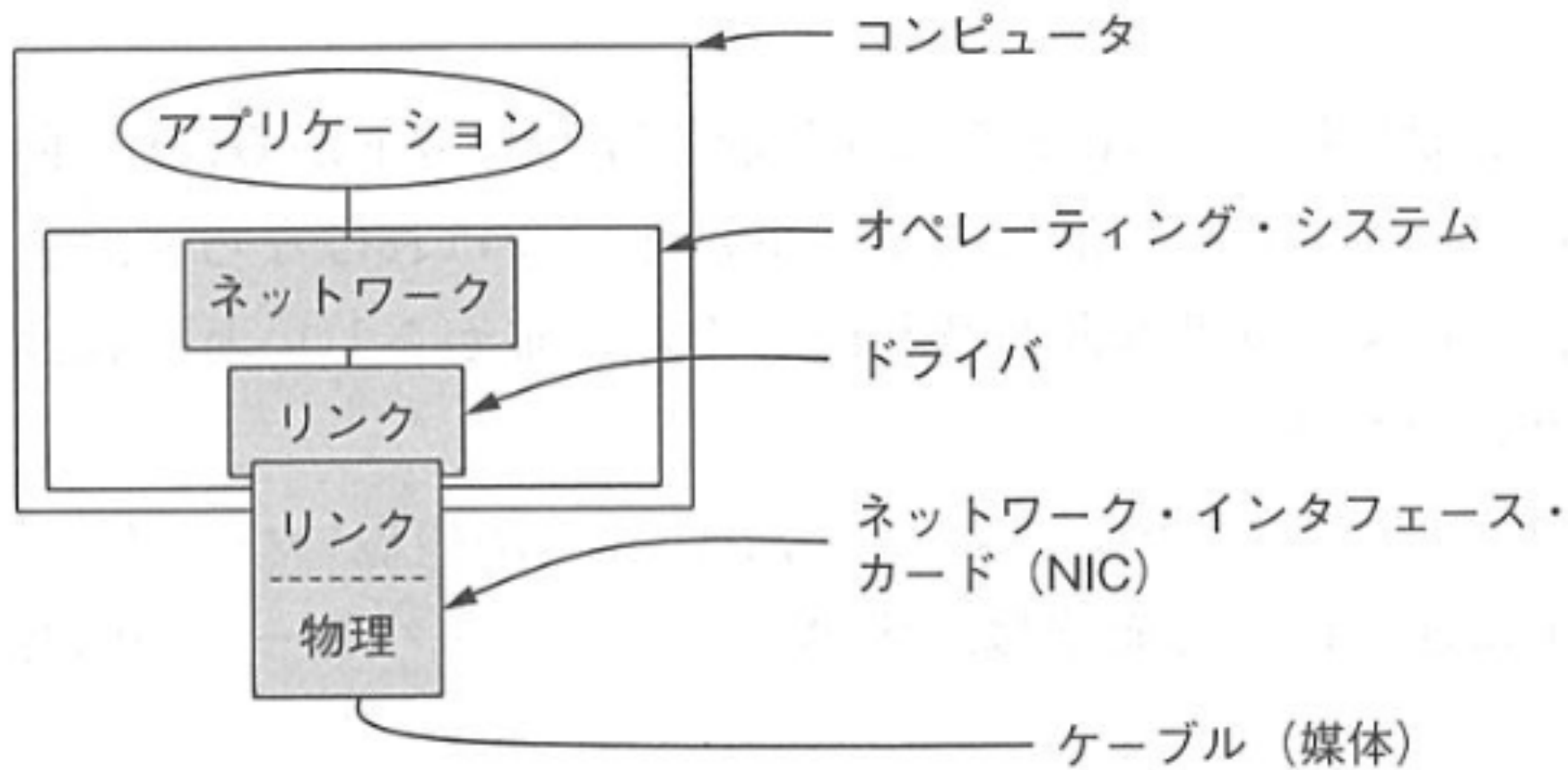


<https://join.slack.com/t/titech-comp-network/signup>

講義日程

	授業計画		課題
04/10	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
04/13	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
04/17	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
04/20	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
04/24	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
04/27	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
05/01	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
05/08	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

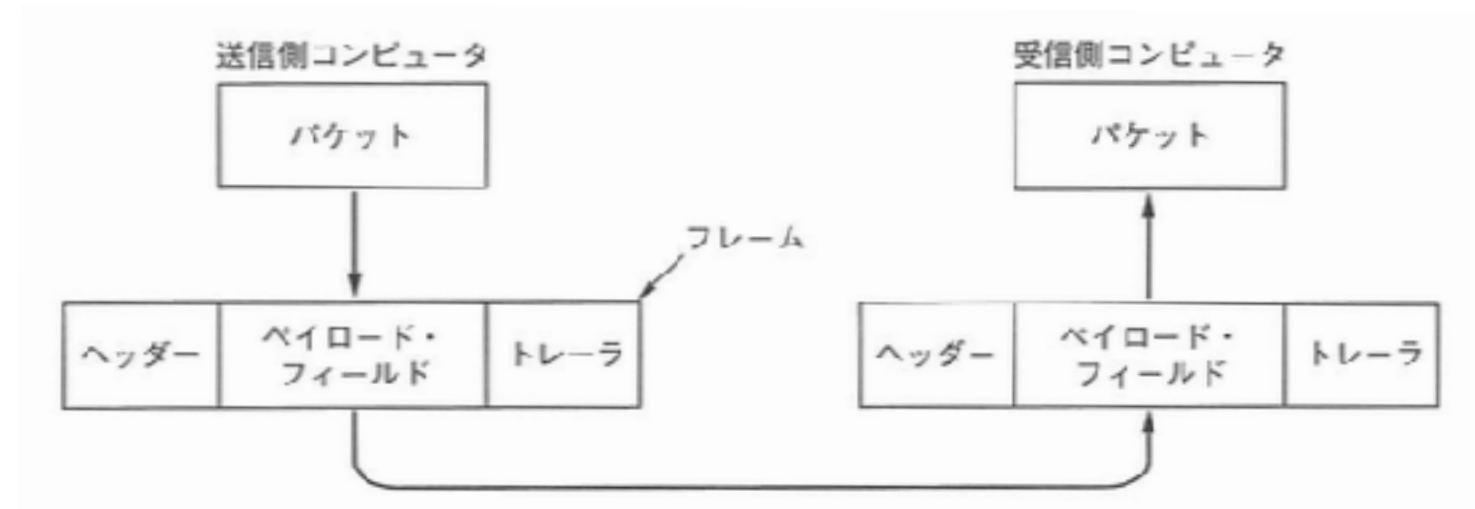
データ・リンク層



ユートピア的単方向プロトコル

仮定

1. 順番通りに届く
2. 送信時間は無視できる
3. 一方向しか送らない
4. 誤りなし
5. 受信バッファ無限



(課題13)

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;

    packet buffer;

    while (true) {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
    }
}
```

```
void receiver1(void) {
    frame r;
    event_type event;

    while (true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
    }
}
```

/* 出力フレームのためのバッファ */

/* 出力パケットのためのバッファ */

/* 送信すべきものを取ってくる */

/* 伝送のためSにコピーする */

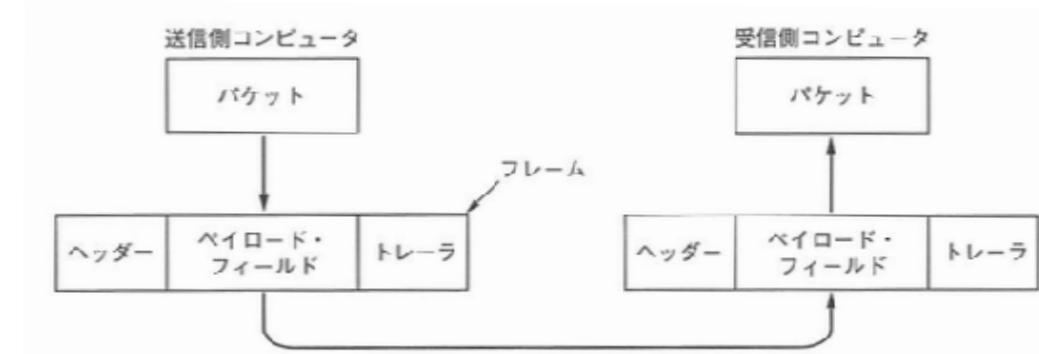
/* 送信する */

/* waitにより設定されるが、ここでは用いない */

/* 唯一の可能性はframe_arrival */

/* 入力したフレームを取ってくる */

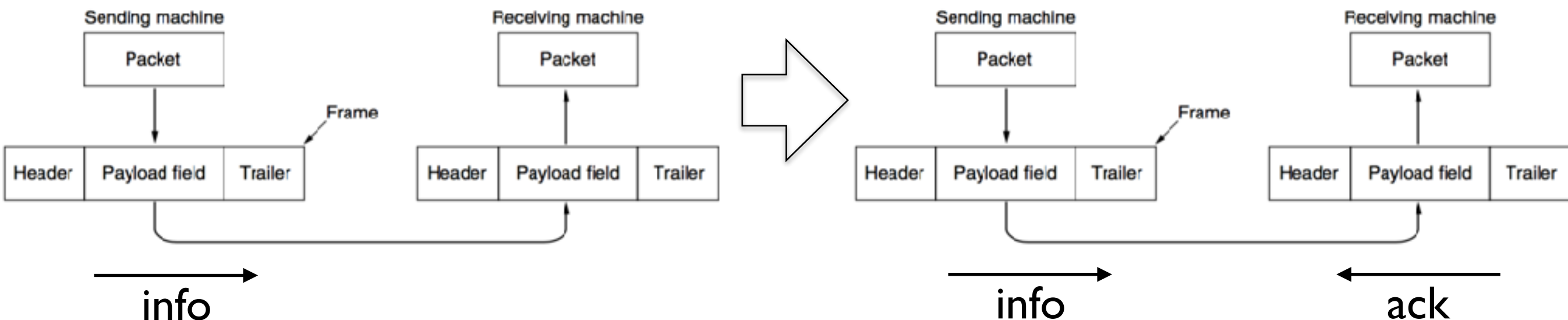
/* ネットワーク層にデータを渡す */



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. 順番通りに届く
2. 送信時間は無視できる
3. 一方向しか送らない
4. 誤りなし
5. 受信バッファ無限



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 */  
    }  
}
```

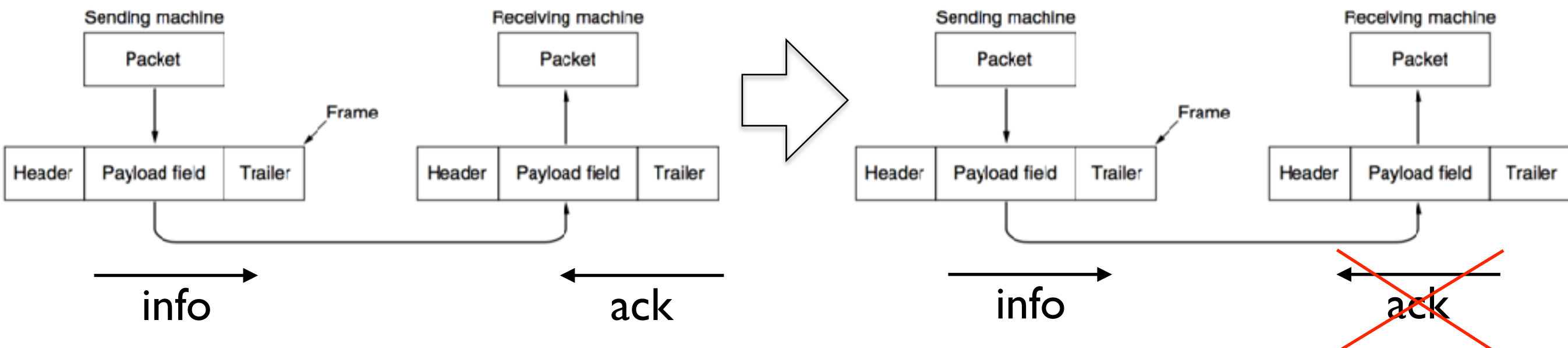
雑音があるチャネル用の単方向逐次確認プロトコル

仮定

(課題14)

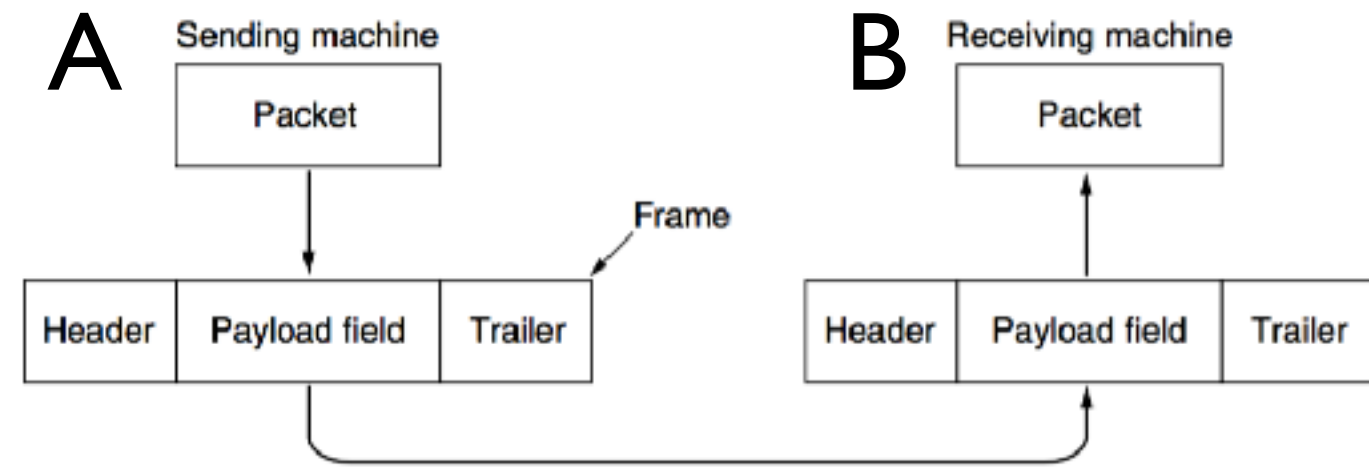
1. 順番通りに届く
2. 送信時間は無視できる
3. 一方向しか送らない
4. ~~誤りなし~~
5. ~~受信バッファ無限~~

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

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

```
#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;                                 /* 出力パッケージのためのバッファ */
    event_type event;

    next_frame_to_send = 0;                       /* 出力順序番号の初期化 */
    from_network_layer(&buffer);                  /* 最初のパッケージを取り出す */

    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);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            s.ack = 1 - frame_expected;
            to_physical_layer(&s);
        }
    }
}

typedef struct {
    frame_kind kind;
    seq_nr seq; 0,1,0,1,0,1,...
    seq_nr ack; 0,1,0,1,0,1,...
    packet info;
} frame;

/* 可能性は : frame_arrival, cksum_err */
/* 有効なフレームが到着した */
/* 新しく届いたフレームを取ってくる */
/* これが待っていたもの */
/* ネットワーク層にデータを渡す */
/* 次回はもう一つの順序番号を期待する */

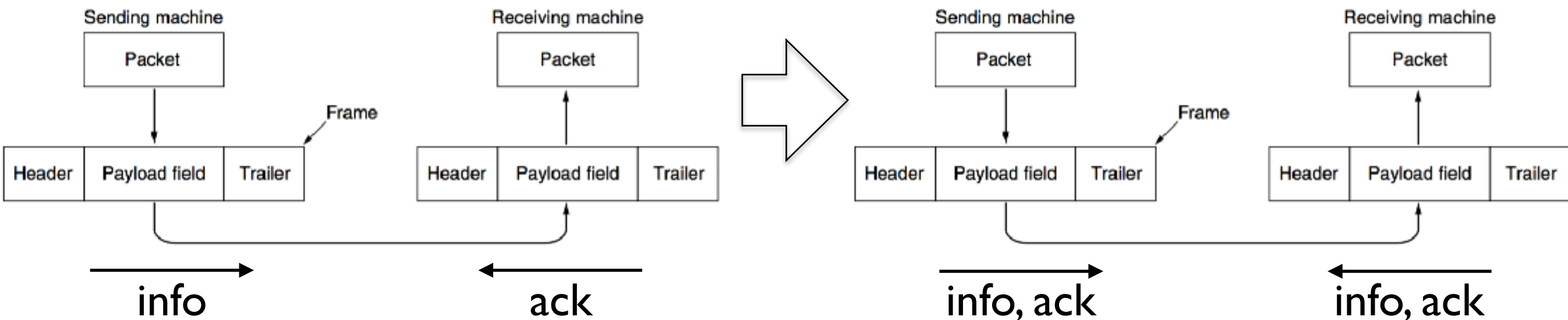
/* どのフレームを確認通知するのか告げる */
/* 確認通知を送る */

/* この層ではフレームが転送される */
/* このフレームの種類は? */
/* 順序番号 */
/* 確認通知番号 */
/* ネットワーク層の packets */
```

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

仮定

1. 順番通りに届く
2. 送信時間は無視できる
3. ~~一方しか送らない~~
4. ~~誤りなし~~
5. ~~受信バッファ無限~~



3 Bit Sliding Window Protocol

```
typedef struct {
```

```
    frame_kind kind;
```

```
    seq_nr seq; 0,1,2,3,4,5,6,7,0,...
```

```
    seq_nr ack; 0,1,2,3,4,5,6,7,0,...
```

```
    packet info;
```

```
} frame;
```

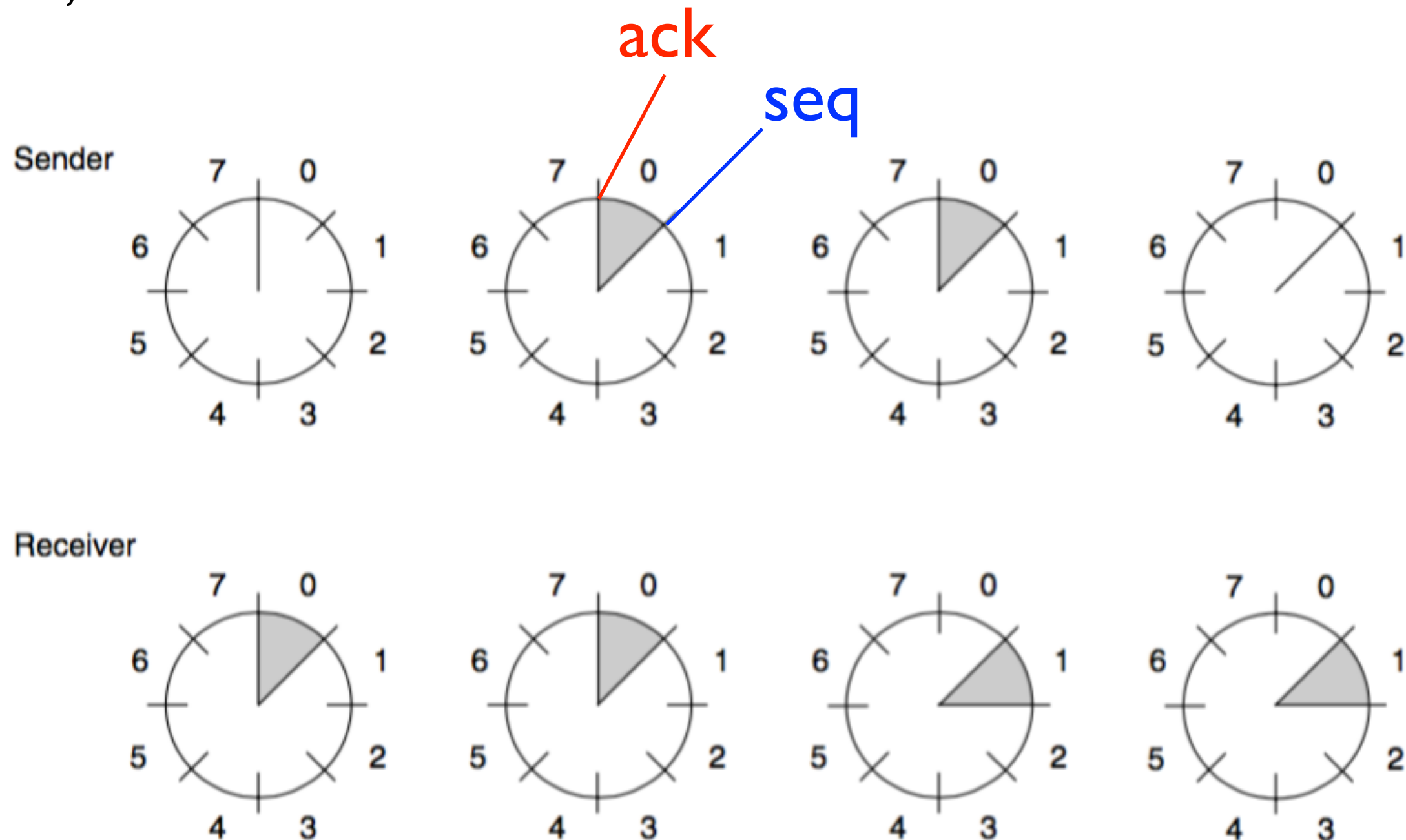
/* この層ではフレームが転送される */

/* このフレームの種類は? */

/* 順序番号 */

/* 確認通知番号 */

/* ネットワーク層のパケット */



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;
    seq_nr frame_expected;
    frame r, s;
    packet buffer;
    event_type event;
    next_frame_to_send = 0;
    frame_expected = 0;
    from_network_layer(&buffer);
    s.info = buffer;
    s.seq = next_frame_to_send;
    s.ack = 1 - frame_expected;
    to_physical_layer(&s);
    start_timer(s.seq);
    while (true) {
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            if (r.ack == next_frame_to_send) {
                stop_timer(r.ack);
                from_network_layer(&buffer);
                inc(next_frame_to_send);
            }
        }
        s.info = buffer;
        s.seq = next_frame_to_send;
        s.ack = 1 - frame_expected;
        to_physical_layer(&s);
        start_timer(s.seq);
    }
}
```

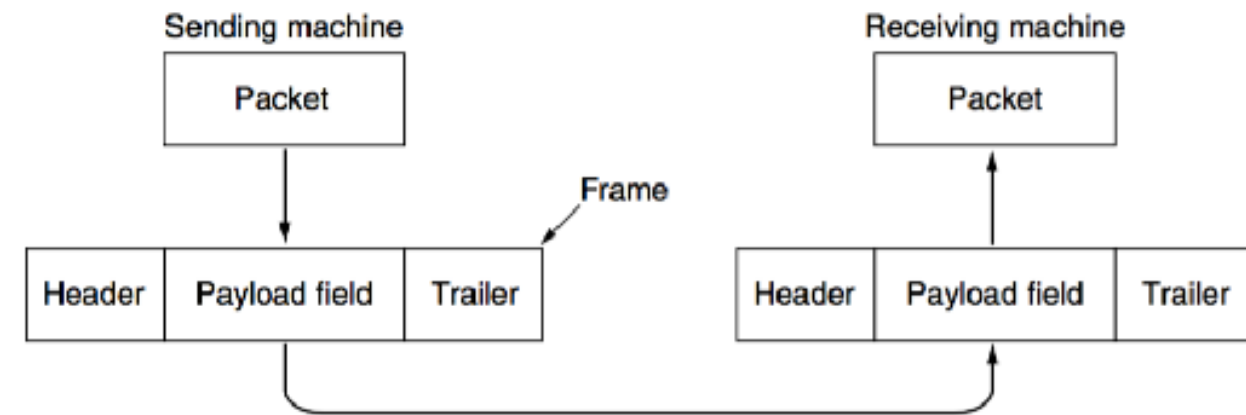
```
/* 0または1だけ */
/* 0または1だけ */
/* 作業変数 */
/* 現在送信中のパケット */

/* 出力ストリーム上の次のフレーム */
/* 次に届く予定のフレーム */
/* ネットワーク層からのパケットを取り出す */
/* 最初のフレームの送信準備 */
/* フレームに順序番号を挿入する */
/* ピギーバックされた確認通知 */
/* フレームを送信する */
/* タイマーの動作を開始 */

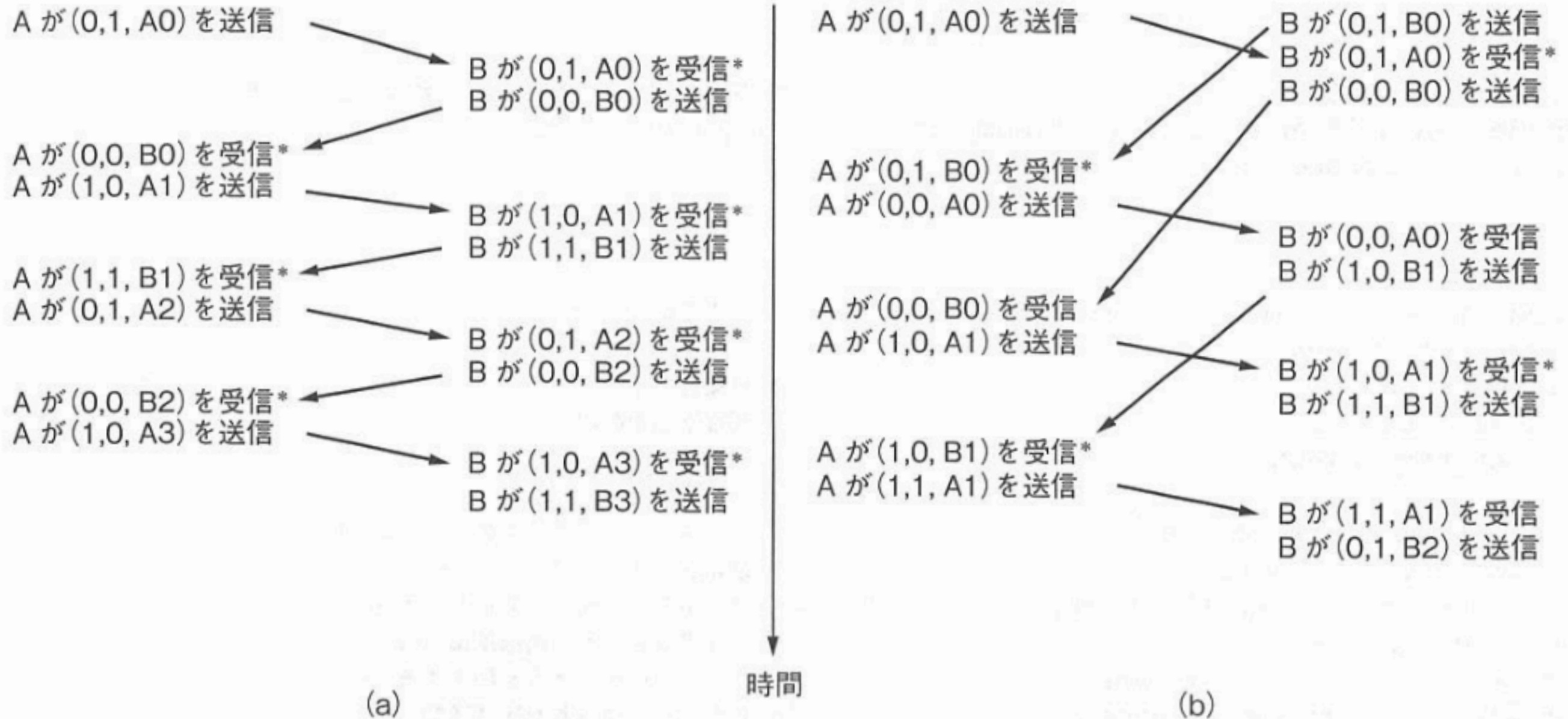
/* frame arrival, cksum err, or timeout */
/* フレームが損傷なしに到着した */
/* それを取ってくる */
/* 入力フレーム・ストリームを処理する */
/* ネットワーク層にパケットを渡す */
/* 次に期待する順序番号を反転する */

/* 出力フレーム・ストリームを処理する */
/* タイマーを止める */
/* ネットワーク層から新しいパケットを取り出す */
/* 送信者の順序番号を反転する */

/* 出力フレームを作成する */
/* 順序番号を挿入する */
/* 最後に受信したフレームの順序番号 */
/* フレームを送信する */
/* タイマーの動作を開始する */
```



One Bit Sliding Window Protocol

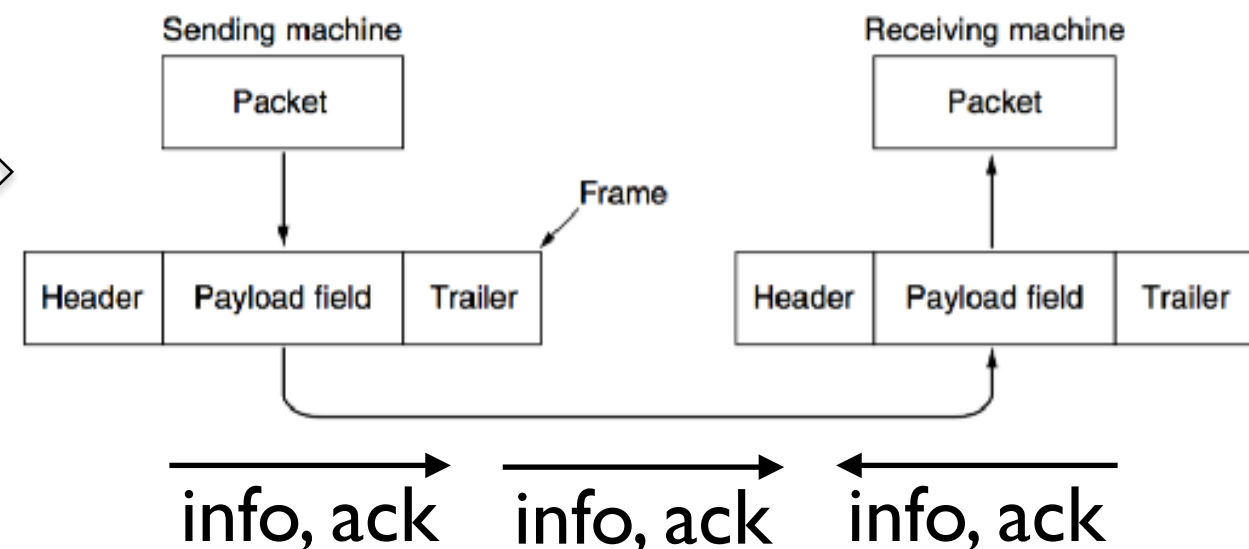
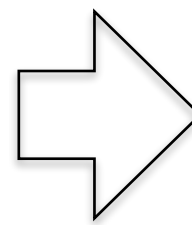
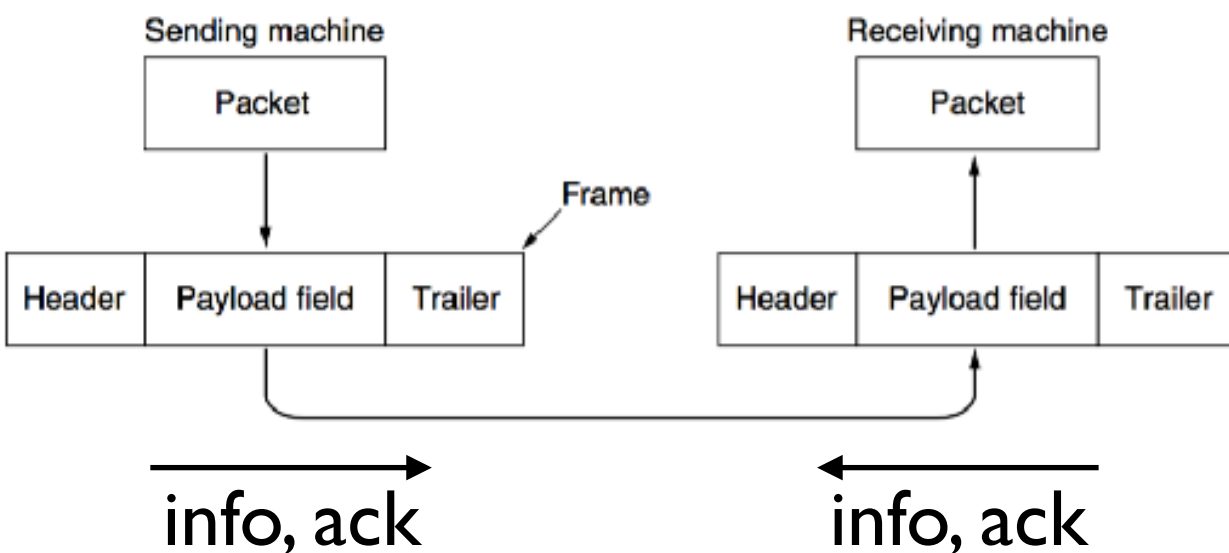
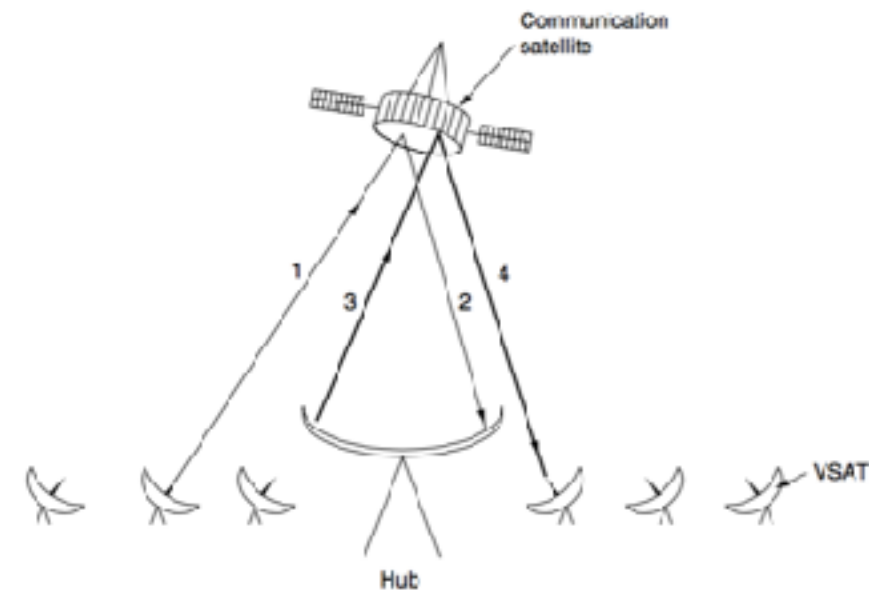


同じフレームが何度も送信されることがある

一括再送(Go-back-N)プロトコル

仮定

1. 順番通りに届く
2. ~~送信時間は無視できる~~
3. ~~一方向しか送らない~~
4. ~~誤りなし~~
5. ~~受信バッファ無限~~



3 Bit Sliding Window Protocol

```
typedef struct {  
    frame_kind kind;  
    seq_nr seq; 0,1,2,3,4,5,6,7,0,...  
    seq_nr ack; 0,1,2,3,4,5,6,7,0,...  
    packet info;  
} frame;
```

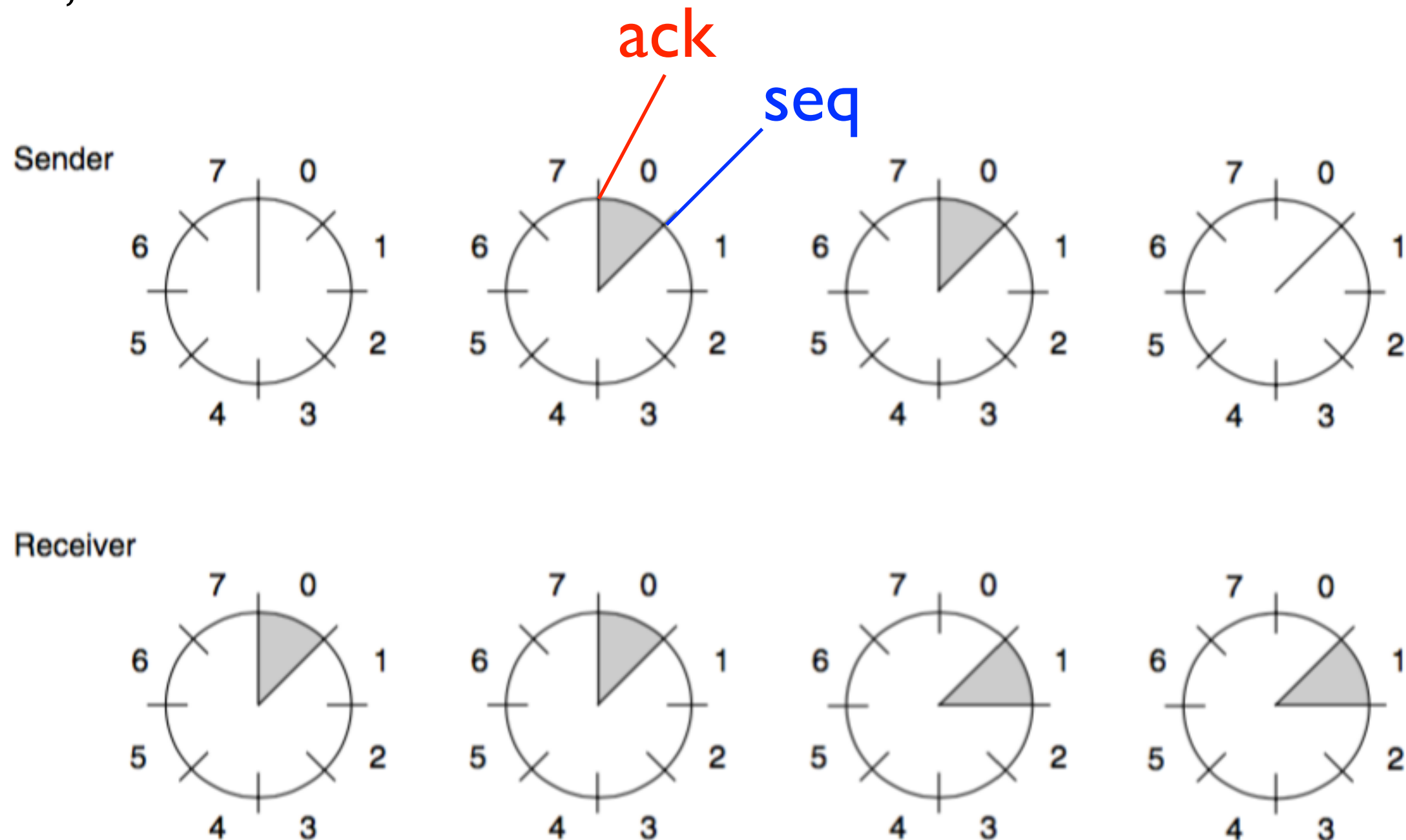
/* この層ではフレームが転送される */

/* このフレームの種類は? */

/* 順序番号 */

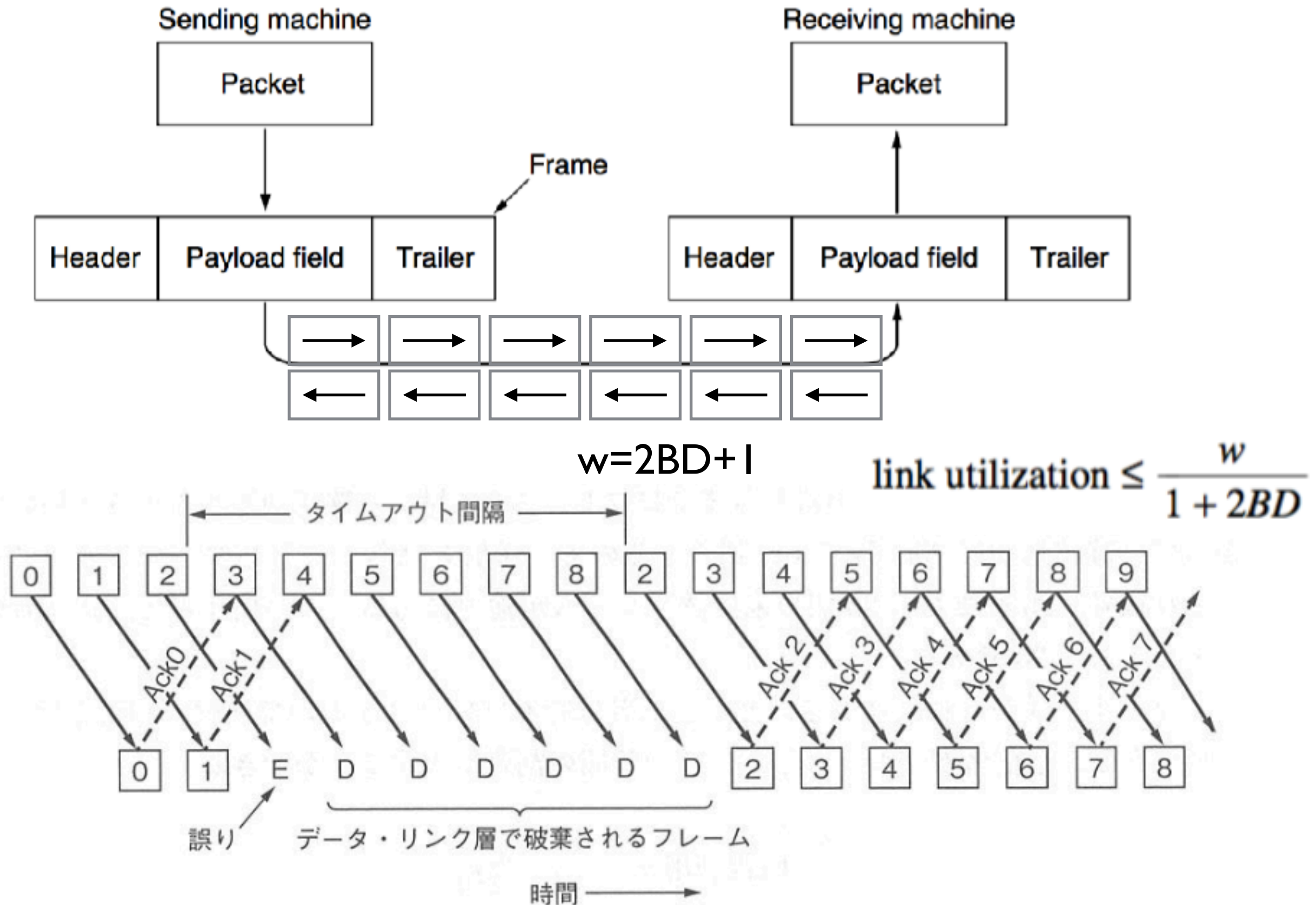
/* 確認通知番号 */

/* ネットワーク層のパケット */



一括再送(Go-back-N)プロトコル

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



一括再送(Go-back-N)プロトコル

```
#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);
    switch(event) {
        case network_layer_ready:
            from_network_layer(&buffer[next_frame_to_send]);
            nbuffered = nbuffered + 1;
            send_data(next_frame_to_send, frame_expected, buffer);
            inc(next_frame_to_send);
            break;
        case frame_arrival:
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            while (between(ack_expected, r.ack, next_frame_to_send)) {
                nbuffered = nbuffered - 1;
                stop_timer(ack_expected);
                inc(ack_expected);
            }
            break;
        case cksum_err:
            break;
        case timeout:
            next_frame_to_send = ack_expected;
            for (i = 1; i <= nbuffered; i++) {
                send_data(next_frame_to_send, frame_expected, buffer);
                inc(next_frame_to_send);
            }
    }
    if (nbuffered < MAX_SEQ)
        enable_network_layer();
    else
        disable_network_layer();
}
}
```

/* 四つの可能性がある：上記のevent_typeを参照 */

/* ネットワーク層に送信すべきパケットがある */

/* 新しいパケットを取り出す */

/* 送信者のウィンドウを広げる */

/* フレームを送信する */

/* 送信者のウィンドウの上端を進める */

/* フレームまたは制御フレームが到着した */

/* 物理層から入力フレームを取ってくる */

/* フレームが期待したものなら受理する */

/* ネットワーク層にパケットを渡す */

/* 受信者のウィンドウの下端を進める */

/* ピギーバックされた確認通知を処理する */

/* バッファされたフレームが一つ減る */

/* フレームが損傷なく到着。タイマーを停止 */

/* 送信者のウィンドウを縮める */

/* 正しくないフレームは無視する */

/* 問題発生。全ての未確認フレームを再送する */

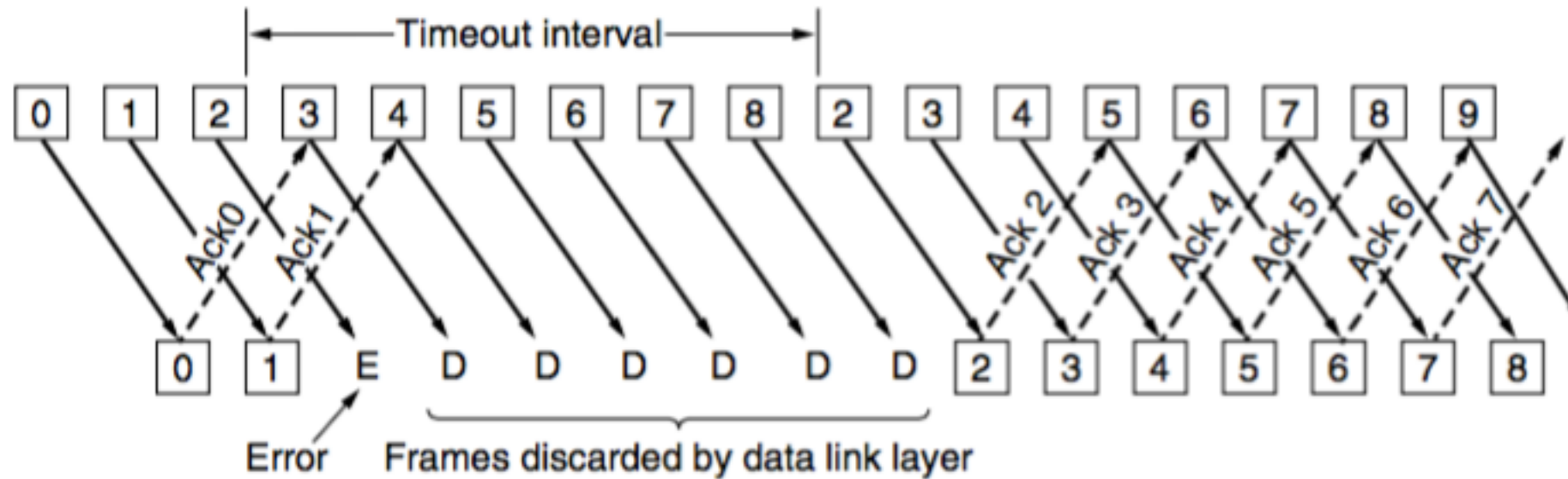
/* ここから再送開始 */

/* フレームを再送 */

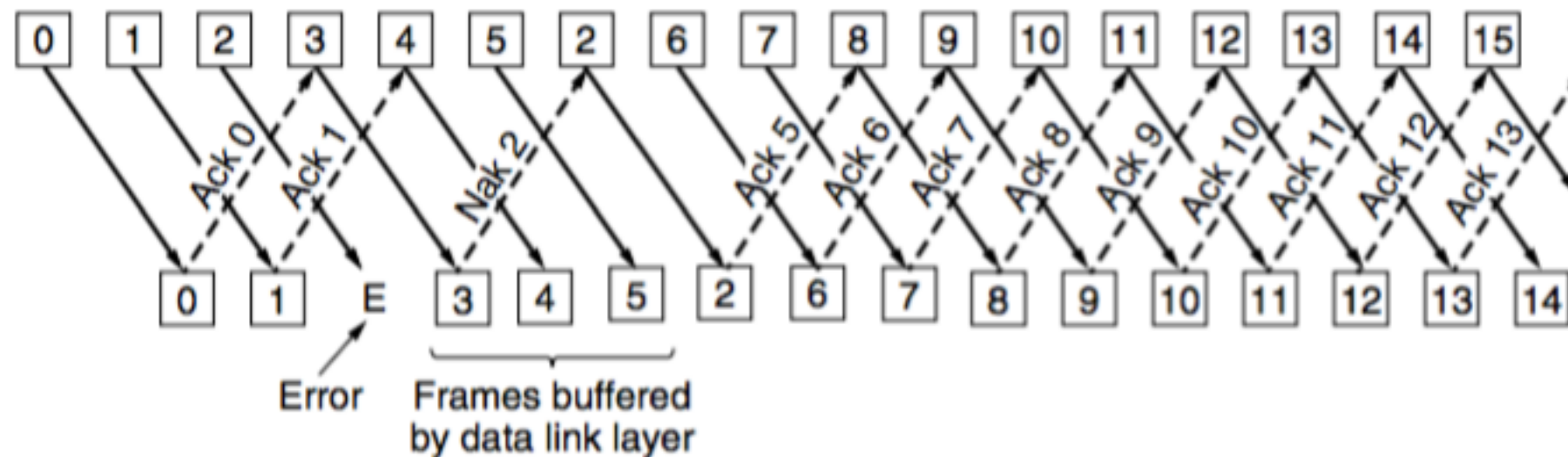
/* 次のものを送る準備をする */

選択的再送プロトコル

(課題15)



一括再送プロトコル



選択的再送プロトコル

```

#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;
for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
while (true) {
    wait_for_event(&event);
    switch(event) {
        case network_layer_ready:
            nbuffered = nbuffered + 1;
            from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]);
            send_frame(data, next_frame_to_send, frame_expected, out_buf);
            inc(next_frame_to_send);
            break;
        case frame_arrival:
            from_physical_layer(&r);
            if (r.kind == data) {
                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;
                    in_buf[r.seq % NR_BUFS] = r.info;
                    while (arrived[frame_expected % NR_BUFS]) {
                        to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                        no_nak = true;
                        arrived[frame_expected % NR_BUFS] = false;
                        inc(frame_expected);
                        inc(too_far);
                        start_ack_timer();
                    }
                }
            }
            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;
                stop_timer(ack_expected % NR_BUFS);
                inc(ack_expected);
            }
            break;
    }
}

```

/* 最初はバッファされているパケットはない */

/* 五つの可能性がある。上記のevent_typeを参照 */

/* 受理、保存、新しいフレームを送信する */

/* ウィンドウを広げる */

/* 新しいパケットを取り出す */

/* フレームを送信する */

/* ウィンドウの上端を進める */

/* フレームまたは制御フレームが到着した */

/* 物理層から入力フレームを取り出す */

/* 損傷のないフレームが到着した */

/* バッファに満の印をつける */

/* バッファにデータを挿入する */

/* フレームを渡してウィンドウを進める */

/* 受信者のウィンドウの下端を進める */

/* 受信者のウィンドウの上端を進める */

/* 個別の確認通知が必要かどうか調べるため */

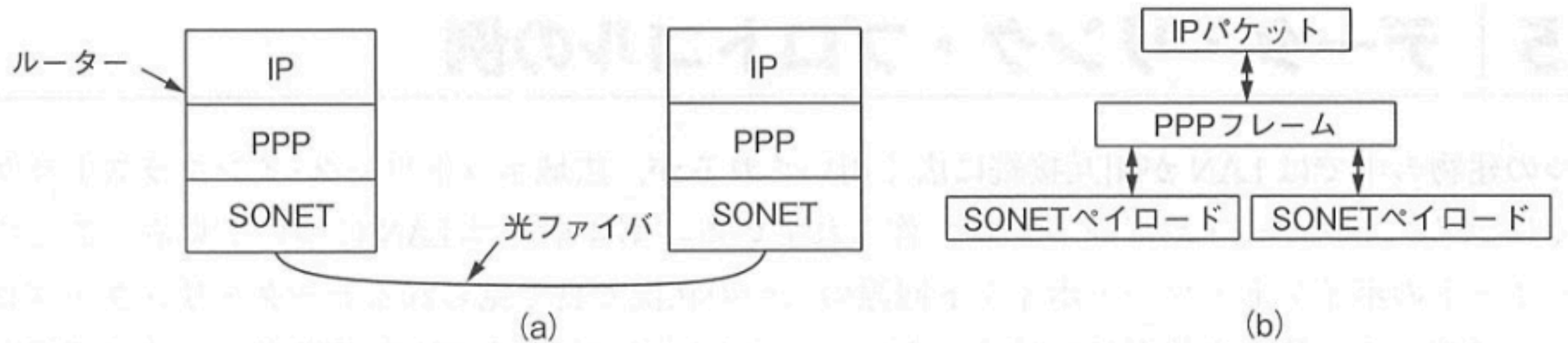
/* ピギーバックされた確認通知を処理する */

/* フレームはそのまま到着した */

/* 送信者のウィンドウの下端を進める */

```
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。

Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110

講義日程

	授業計画		課題
04/10	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
04/13	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
04/17	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
04/20	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
04/24	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
04/27	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
05/01	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
05/08	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