TwitVisor
(Web公開用)

＊筑波大学大学院　島田恭平
筑波大学大学院　芹川大地

注意事項

本スライド中のソースコード等は，スライドのレイアウトの都合上，一部省略や簡略化されている部分があります

TwitVisorとは？

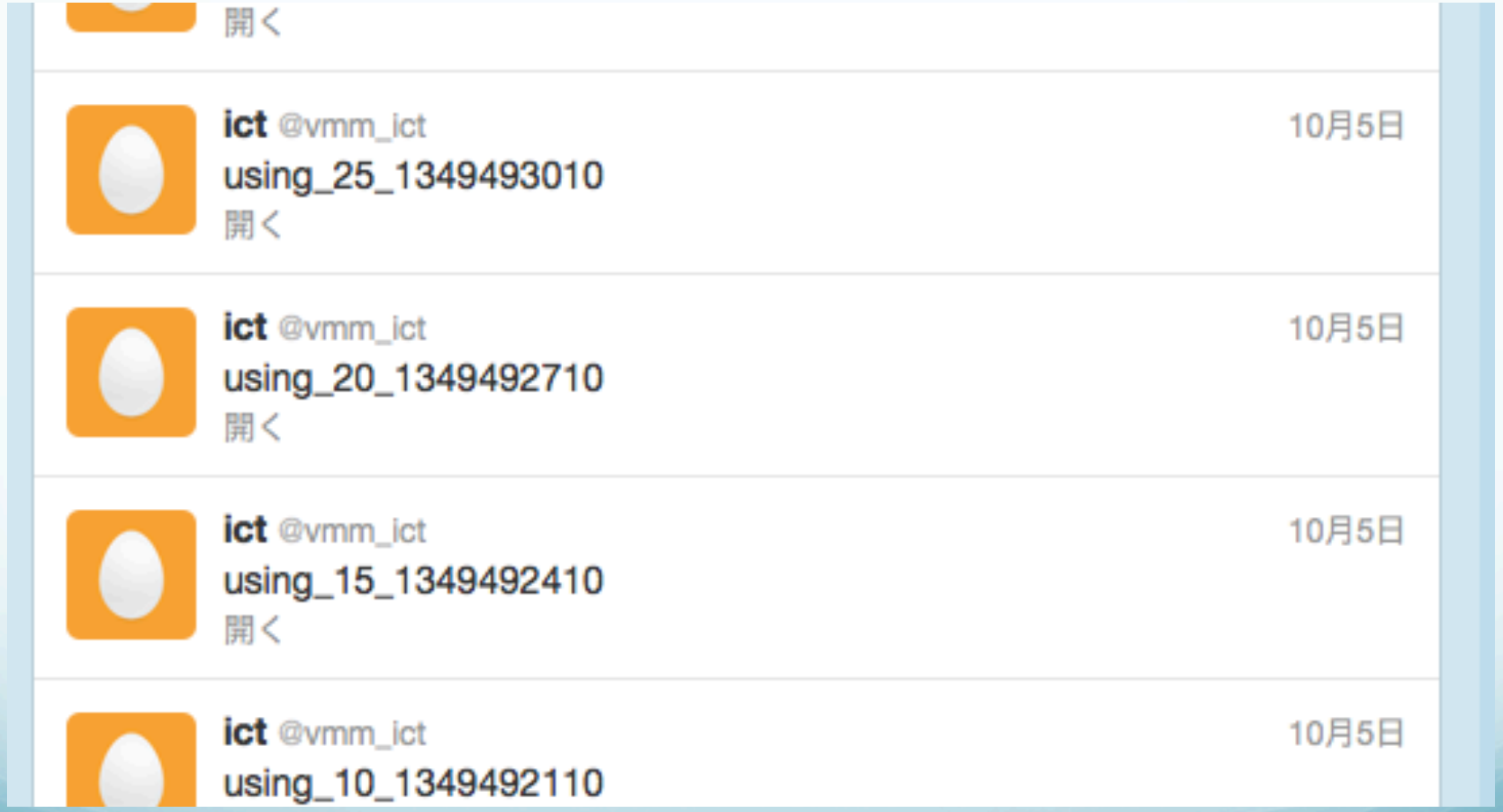
- BitVisor(ver 1.2)を改変し，twitterへのツイート投稿機能を追加
  - ゲストOSを介さずにBitVisorから直接twitterへ投稿する

TwitVisorとは？

- BitVisor(ver 1.2)を改変し，twitterへのツイート投稿機能を追加
  - ゲストOSを介さずにBitVisorから直接twitterへ投稿する

BitVisorがつぶやく

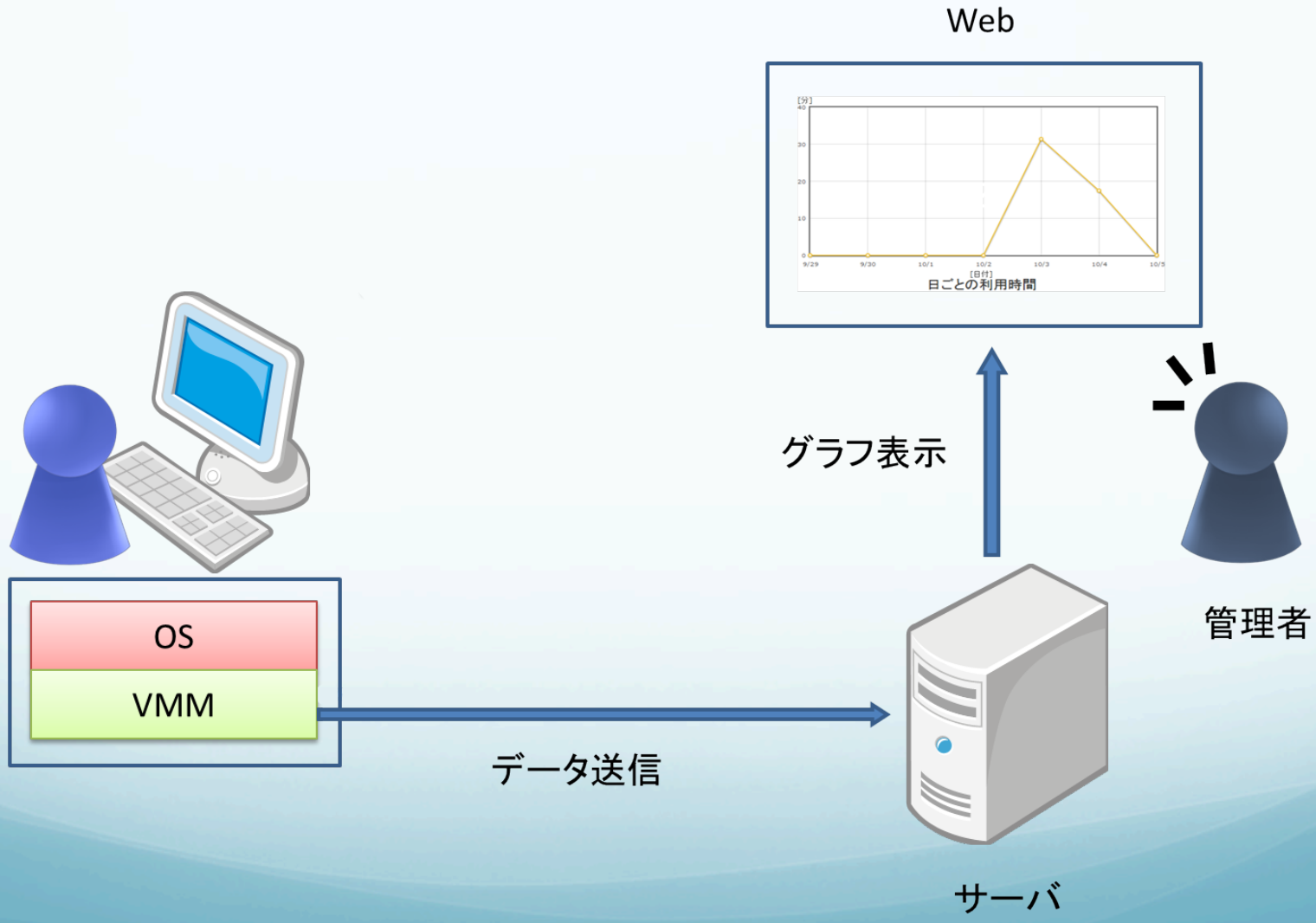
TwitVisor(@vmm_ict)からの投稿



開発経緯

- 大学院のグループワークの授業
  - グループで自由に問題を設定し，問題を解決するシステムの設計，実装を行う
- 「VMMを用いた健全なコンピュータ使用支援システム」の開発をすることに決定
  - 企業で管理者が社内のPC使用状況を監視するシステム
- 利点
  - 使用しているOSやアプリケーションに依存しない
  - 管理者権限でもゲストOSからシステムを停止されない

システム概要図(案)



システム概要

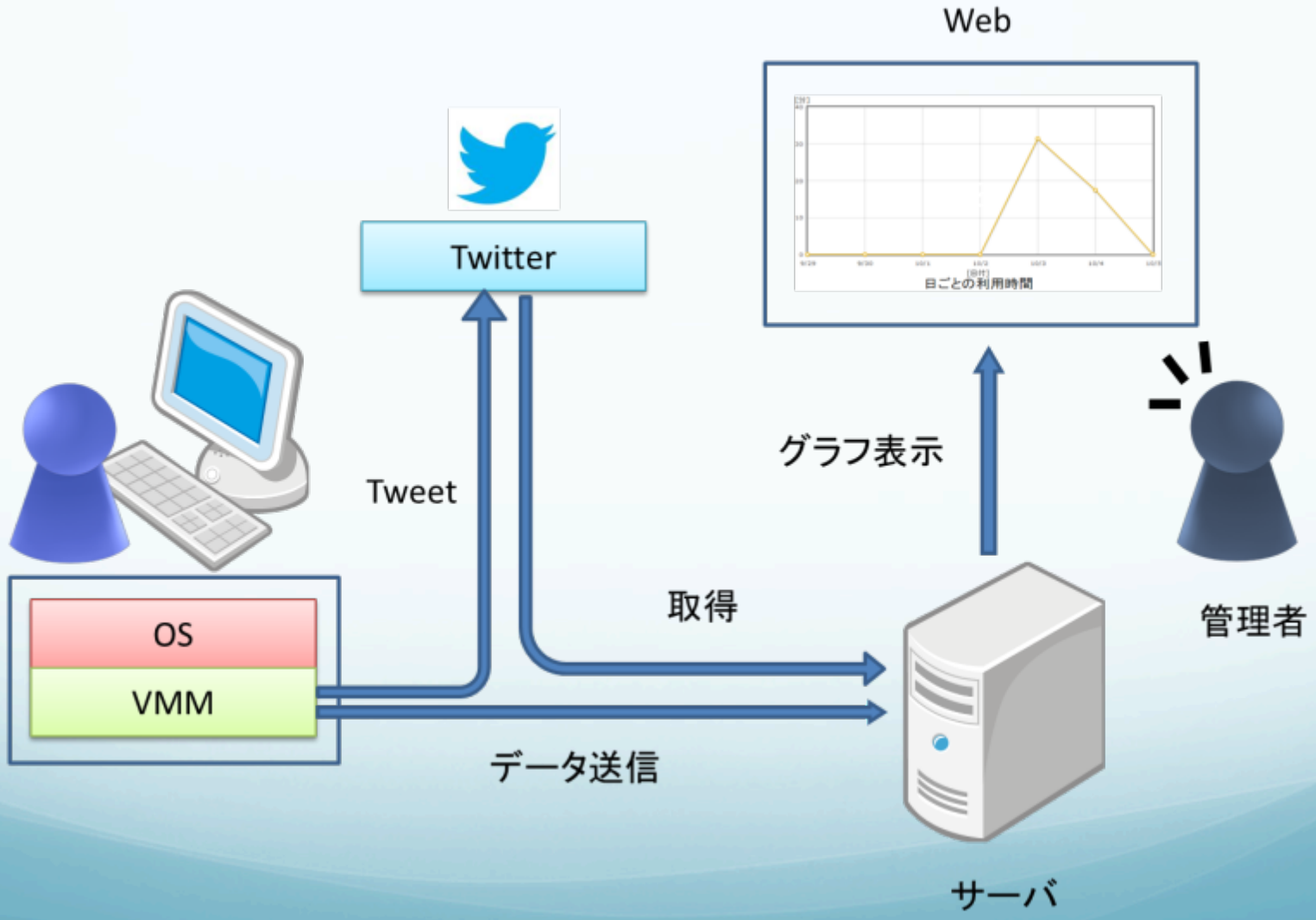
- VMMにBitVisorを使用し，PCの使用状況を監視する
  - PCの使用時間
  - 入力したキー
  - …etc
- 収集したデータをWebサーバ上に送信

システム概要

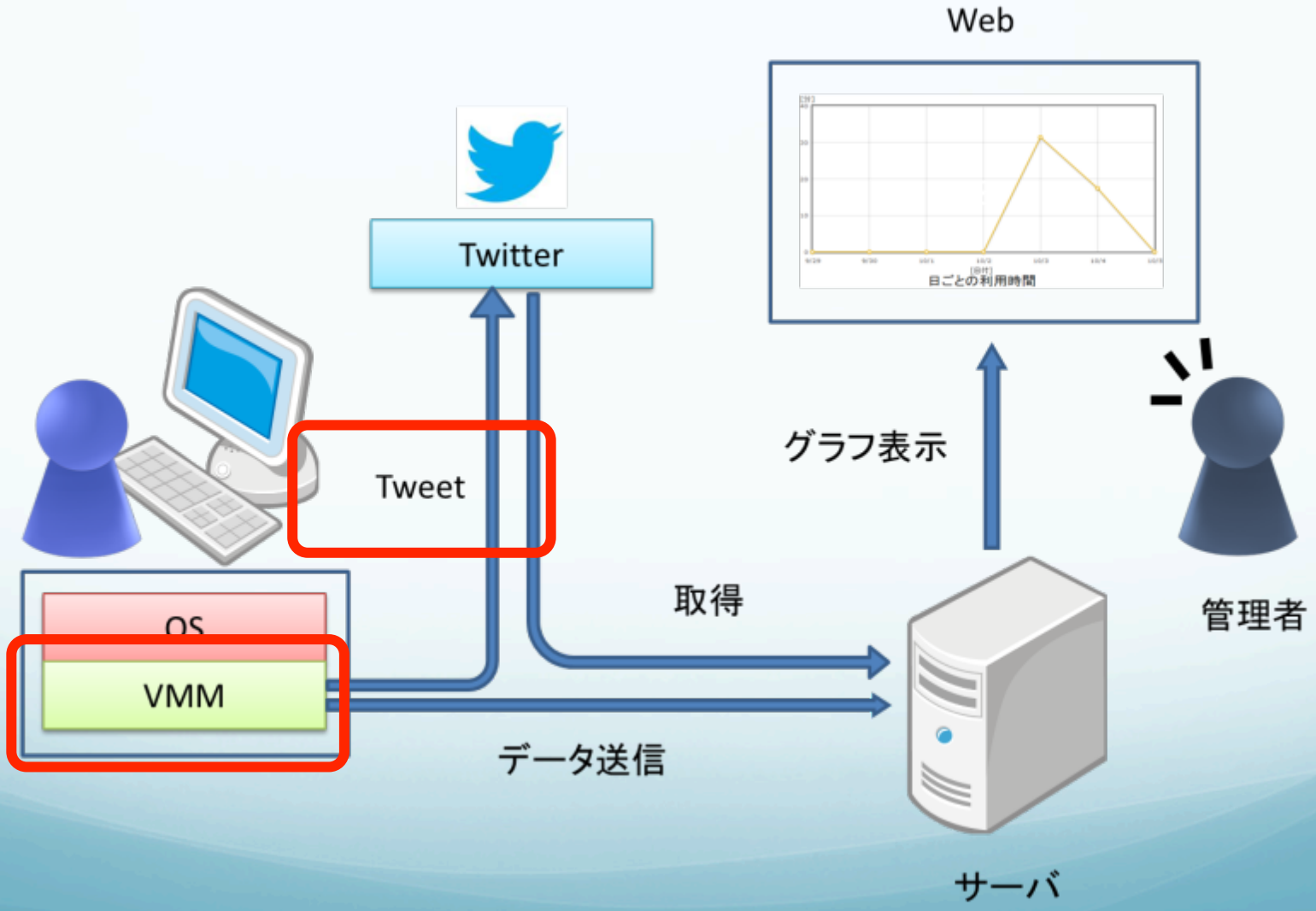
- VMMにBitVisorを使用し，PCの使用状況を監視する
  - PCの使用時間
  - 入力したキー
  - …etc
- 収集

送信先をtwitterにしよう

システム概要図(改)



システム概要図(改)



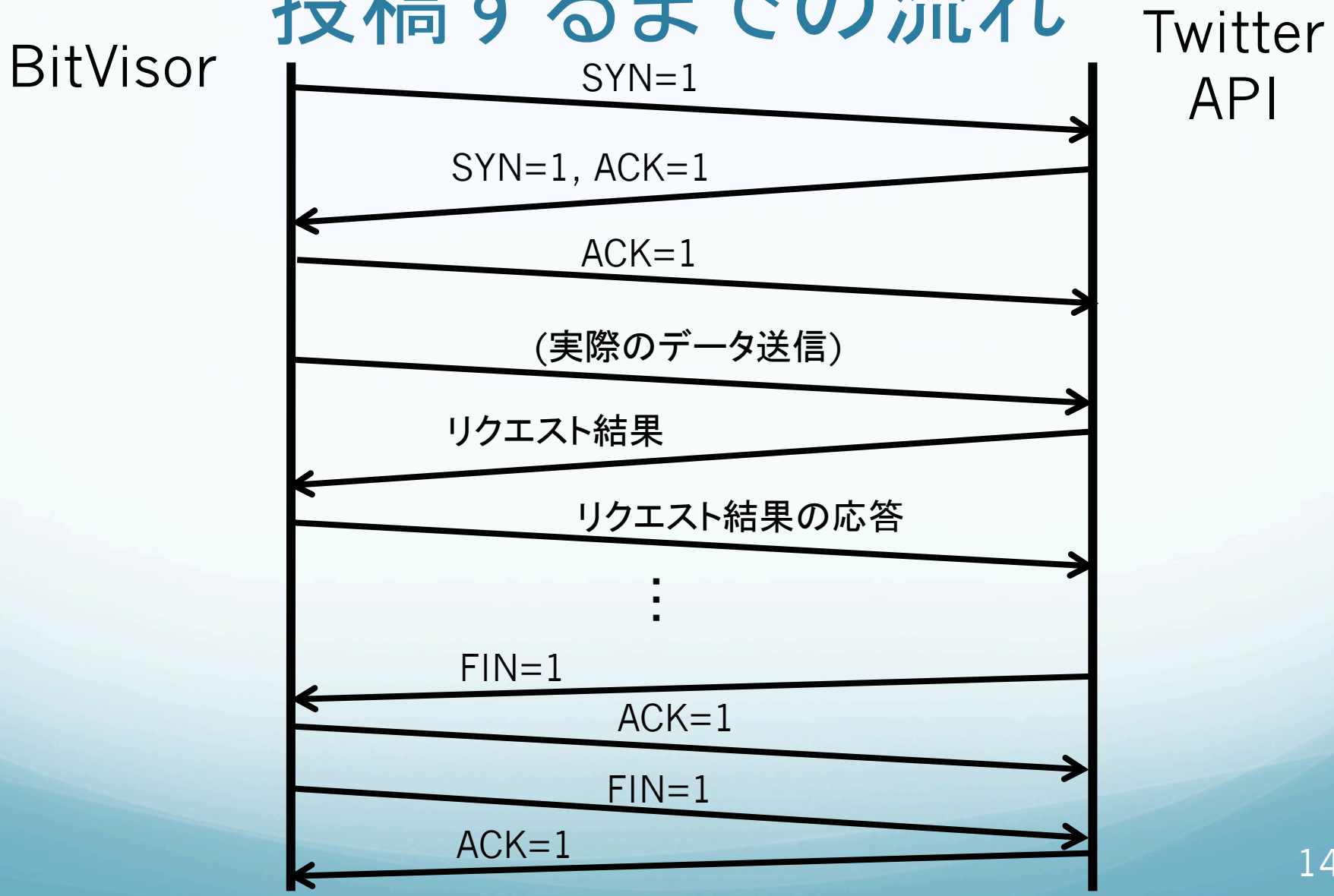
TwitVisorの実装目標

- BitVisorからゲストOSを介さずtwitterに直接投稿する
  - 複雑な実装を極力排除する
  - twitter APIの仕様に沿って，実際にtweetするパケットを作成して送信する

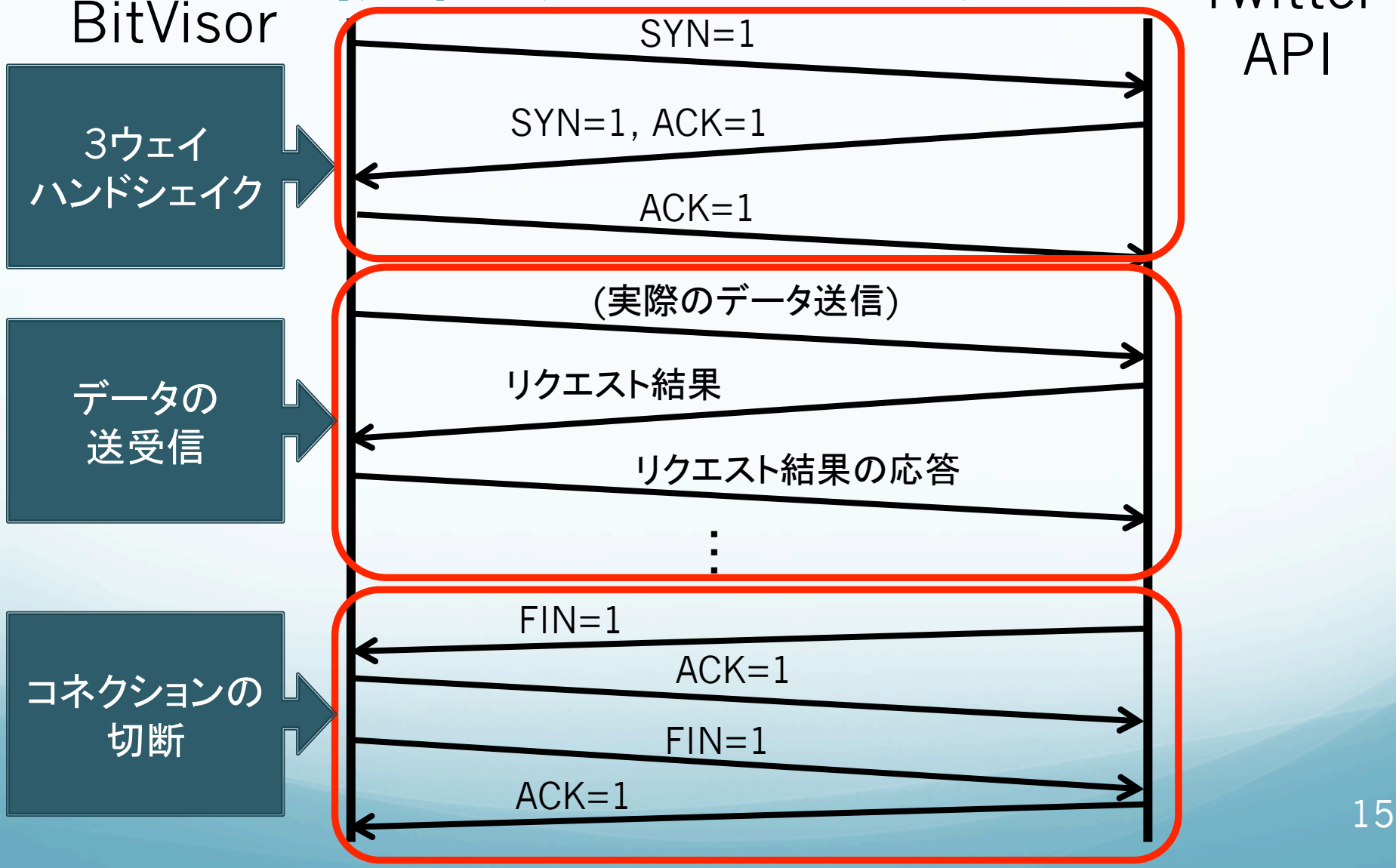
実装した点

- Twitterに投稿するための必要最低限のTCP実装
  - 3ウェイハンドシェイク
  - データの送受信
  - コネクション切断
- ゲストOSから独立したtwitterへの送受信機能
- OAuthに必要なデータを取得，計算する機能

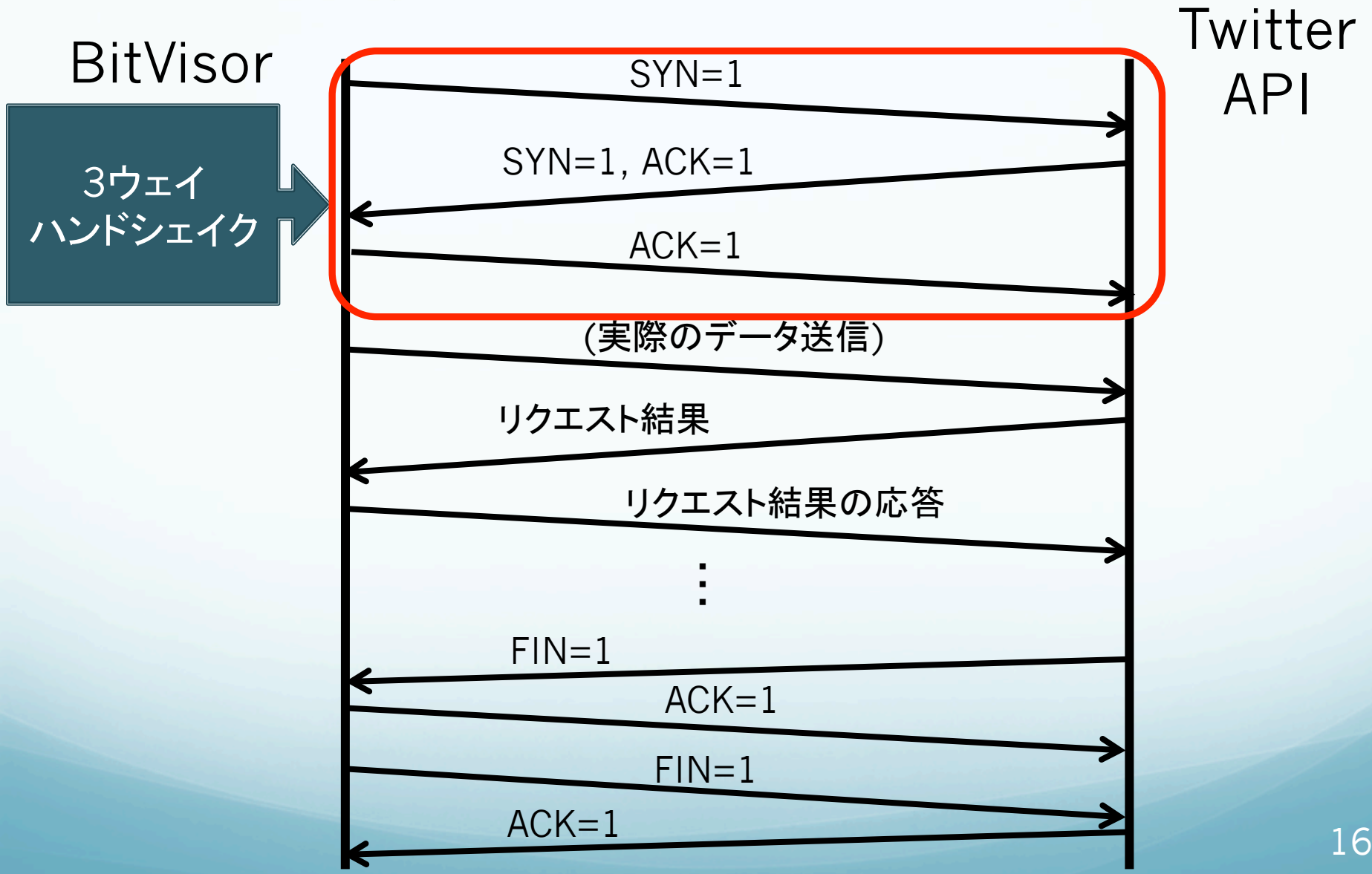
twitterに投稿するまでの流れ



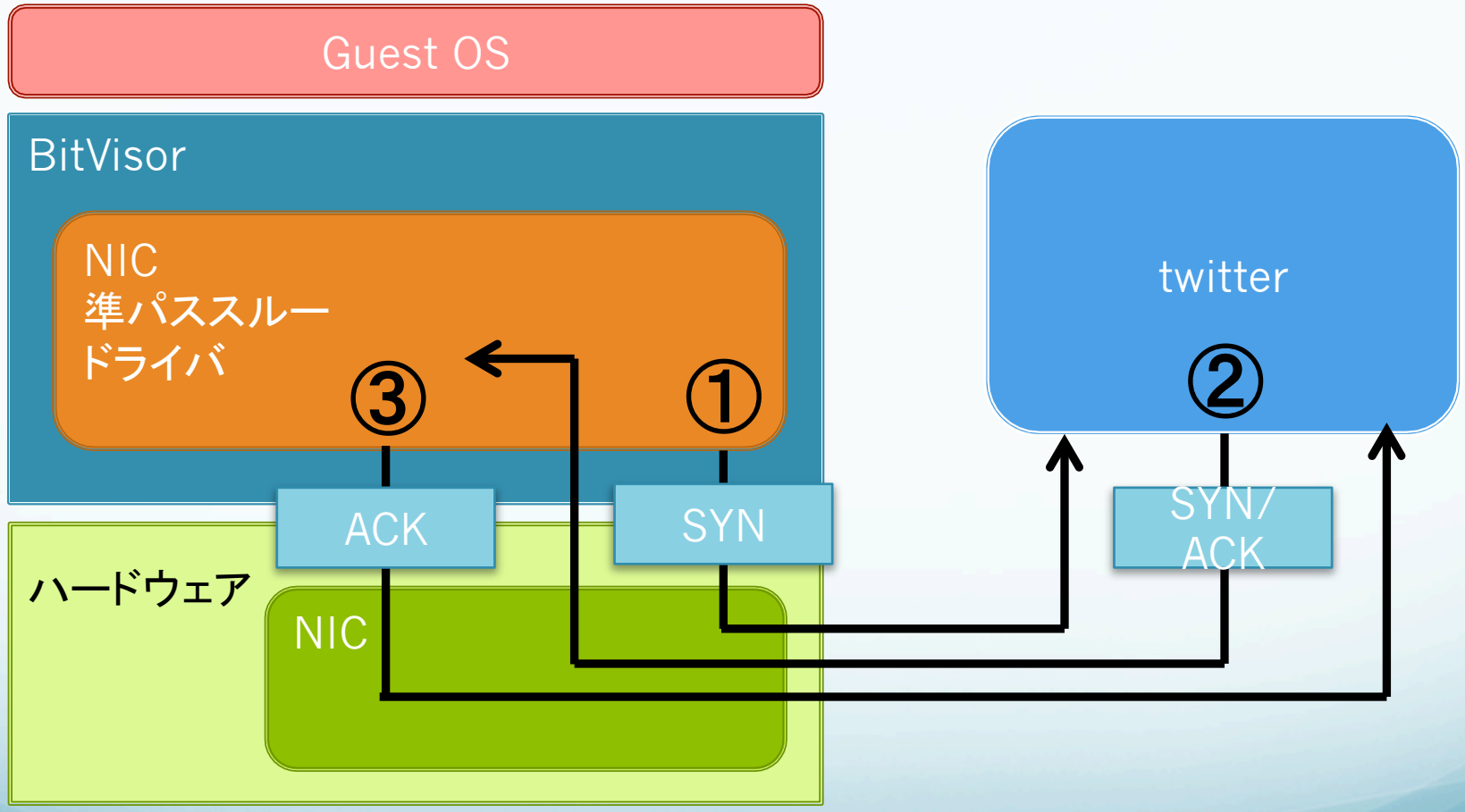
twitterに投稿するまでの流れ



3ウェイハンドシェーク



3ウェイハンドシェイク　実装概要



3ウェイハンドシェイク/1手目(1/2)

1. SYNパケットを作る
   - core/tty.cのmkudp(UDPヘッダ作成)を参考に実装
   - UDPヘッダの変わりにTCPヘッダ,TCPパケットを作成する関数を作成
     - mktcpHeader関数
     - mktcp関数

```
void   oneHand(u8 *sip, u16 sport, u8 *dip, u16 dport)
{
    <略>
    code_bit = mkcodeBit(off, off, off, off, on, off);
    th = mktcpHeader(sport, dport, (u32)12345,
                     (u32)0, code_bit, (u16)1500, (u16)0);

    memcpy(pkt+12, "\x08\x00", 2); //フレームタイプ
    pktsiz = mktcp(pkt+14, sip, dip, th, NULL, 0) + 14;

    tty_tcp_putchar((char *)pkt, pktsiz);
    <略>
}
```

SYNフラグ(0x02)

TCPヘッダ構造体の作成

パケット(pkt)にIPヘッダとTCPヘッダを結合(NULLはTCPペイロード)

パケット送信(次スライド)

3ウェイハンドシェイク/1手目(2/2)

2. パケットを送信する
   - core/tty.cのtty_udp_putcharを参考にtty_tcp_putcharを実装

```
void tty_tcp_putchar (char *pkt, unsigned int pktsiz)
{
        <略>
        LIST1_FOREACH (tty_udp_list, p) {
                p->tty_send (p->handle, pkt, pktsiz);
        }
}
```

任意のパケットとパケットサイズを受け取るように

3ウェイハンドシェイク/2手目,3手目

- TwitterAPIからのパケットをフックする
  - vpn/Se/SeVpn.cのSeVpnRecvEtherPacketにTCPフック用の関数を追加

```
void *SeVpnRecvEtherPacket(SE_VPN *v, SE_ETH *e)
{
    <略>
    // 次のパケットを取得
    packet = SeEthGetNextRecvPacket(e);
    if (packet == NULL){return NULL;}

    if( hookTcpPacket(packet, 56789) ){
        return NULL;
    }
    <略>
}
```

全てのTCPパケットをフックし，
Twitter APIのパケットならtrue
(詳細は次ページ)

NULLを返して
ゲストOSにパケットが届かないようにする

hookTcpPacket

```
bool hookTcpPacket(void *_packet, u16 selected_port){
    <略>
    if( !is_tcp(_packet) ) return false;
    if( !is_twitter(_packet, selected_port) ) return false;

    if( is_syn_ack_packet(_packet) ){
      threeHand(selected_port, seq_num, ack_num);

      tweet();

    }else{
        //3ウェイハンドシェークとは関係のないパケット
    }
    return true;
}
```

twitter APIからの
パケットであることを確認

3ウェイハンドシェークの
3手目を送信
(1手目と同様に送信)

実際にツイートする
データを送信(後述)

hookTcpPacket

```
bool hookTcpPacket(void *_packet, u16 selected_port){
    <略>
    if( !is_tcp(_packet) ) return false;
    if( !is_twitter(_packet, selected_port) ) return false;

    if( is_syn_ack_packet(_packet) ){
      threeHand(selected_port, seq_num, ack_num);

      tweet();

    }else{
        /
    }
    retu
}
```

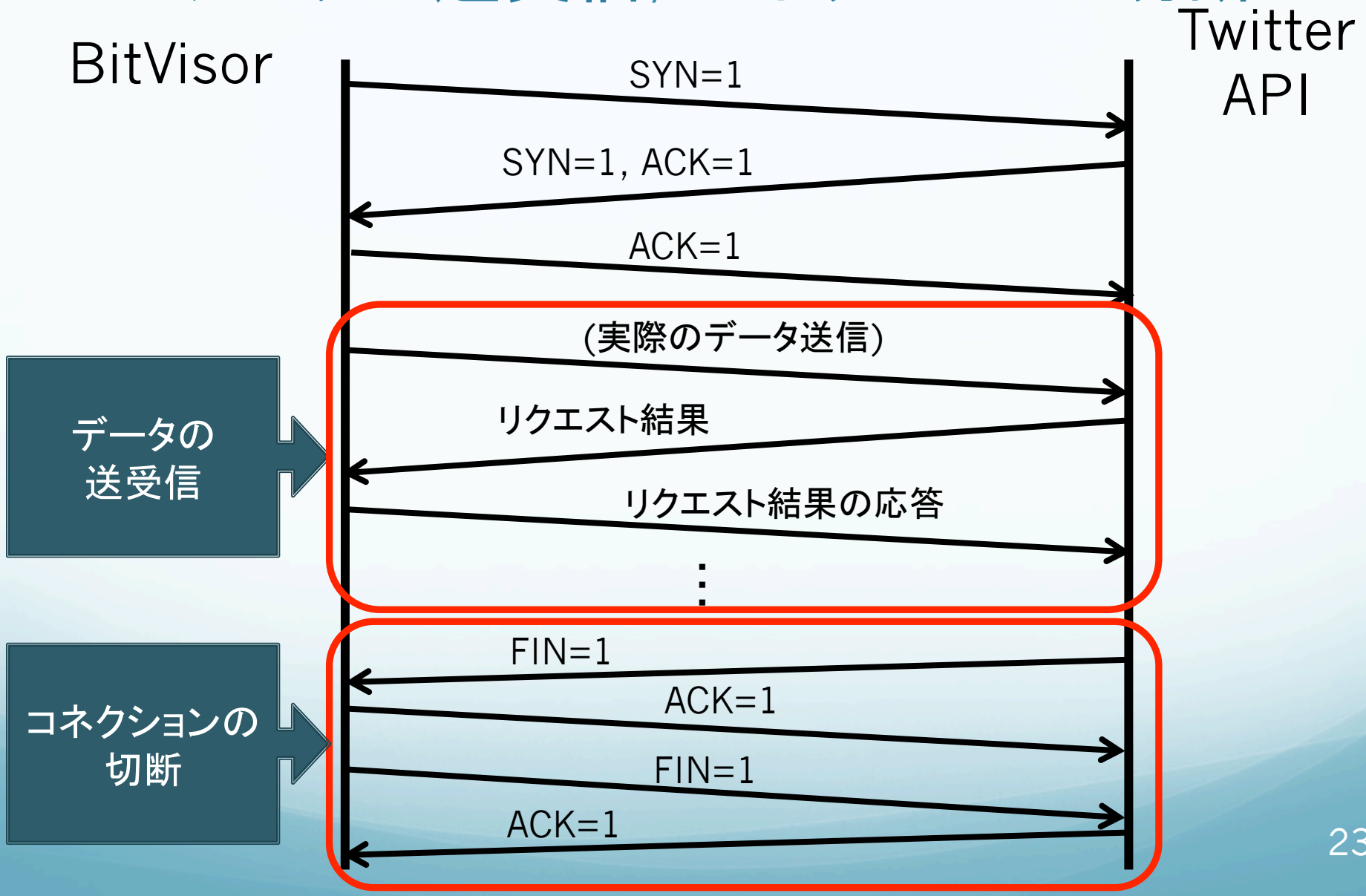
twitter APIからの
パケットであることを確認

3ウェイハンドシェークの
3手目を送信
(1手目と同様に送信)

実際にツイートする
データを送信(後述)

最低限の3ウェイハンドシェークを実装

データの送受信，コネクションの切断



データの送受信，コネクションの切断

- 前述したhookTcpPakect関数に処理を追加

```
bool hookTcpPacket(void *_packet, u16 selected_port){
    <略>
    if( is_syn_ack_packet(_packet) ){
        //前述した3ウェイハンドシェークの実装
        tweet();
    }else{
        if( is_fin_packet(_packet) ){
            send_tcp_packet(/*FINに対するACKを送信*/)
            send_tcp_packet(/*BitVisorからもFINを送信*/)

        }else if( is_fin_ack_packet(_packet) ){
            //切断完了の処理
        }else{

            //シーケンシャル番号，ACK番号を更新しACKを送信する
            send_tcp_packet(/*略*/);
        }
    }
    return true;
}
```

①．実際にツイートするデータを送信(後述)

データの送受信，コネクションの切断

- 前述したhookTcpPakect関数に処理を追加

```
bool hookTcpPacket(void *_packet, u16 selected_port){
    <略>
    if( is_syn_ack_packet(_packet) ){
        //前述した3ウェイハンドシェークの実装
        tweet();
    }else{
        if( is_fin_packet(_packet) ){
            send_tcp_packet(/*FINに対するACKを送信*/)
            send_tcp_packet(/*BitVisorからもFINを送信*/)

        }else if( is_fin_ack_packet(_packet) ){
            //切断完了の処理
        }else{

            //シーケンシャル番号，ACK番号を更新しACKを送信する
            send_tcp_packet(/*略*/);
        }
    }
    return true;
}
```

①．実際にツイートするデータを送信(後述)

②．①で送信したリクエスト結果に対してACKを返送

データの送受信，コネクションの切断

- 前述したhookTcpPakect関数に処理を追加

```
bool hookTcpPacket(void *_packet, u16 selected_port){
    <略>
    if( is_syn_ack_packet(_packet) ){
        //前述した3ウェイハンドシェークの実装
        tweet();
    }else{
        if( is_fin_packet(_packet) ){
            send_tcp_packet(/*FINに対するACKを送信*/)
            send_tcp_packet(/*BitVisorからもFINを送信*/)

        }else if( is_fin_ack_packet(_packet) ){
            //切断完了の処理
        }else{

            //シーケンシャル番号，ACK番号を更新しACKを送信する
            send_tcp_packet(/*略*/);
        }
    }
    return true;
}
```

①．実際にツイートするデータを送信(後述)

③．FINを受信するとACKを返送し，こちらからもFINパケットを送信

②．①で送信したリクエスト結果に対してACKを返送

データの送受信，コネクションの切断

- 前述したhookTcpPakect関数に処理を追加

```
bool hookTcpPacket(void *_packet, u16 selected_port){
    <略>
    if( is_syn_ack_packet(_packet) ){
        //前述した3ウェイハンドシェークの実装
        tweet();
    }else{
        if( is_fin_packet(_packet) ){
            send_tcp_packet(/*FINに対するACKを送信*/)
            send_tcp_packet(/*BitVisorからもFINを送信*/)

        }else if( is_fin_ack_packet(_packet) ){
            //切断完了の処理
        }else{

            //シーケンシャル番号，ACK番号を更新しACKを送信する
            send_tcp_packet(/*略*/);
        }
    }
    return true;
}
```

①. 実際にツイートするデータを送信(後述)

③. FINを受信するとACKを返送し，こちらからもFINパケットを送信

④. ③で送信したFINに対するACKを受信すると切断

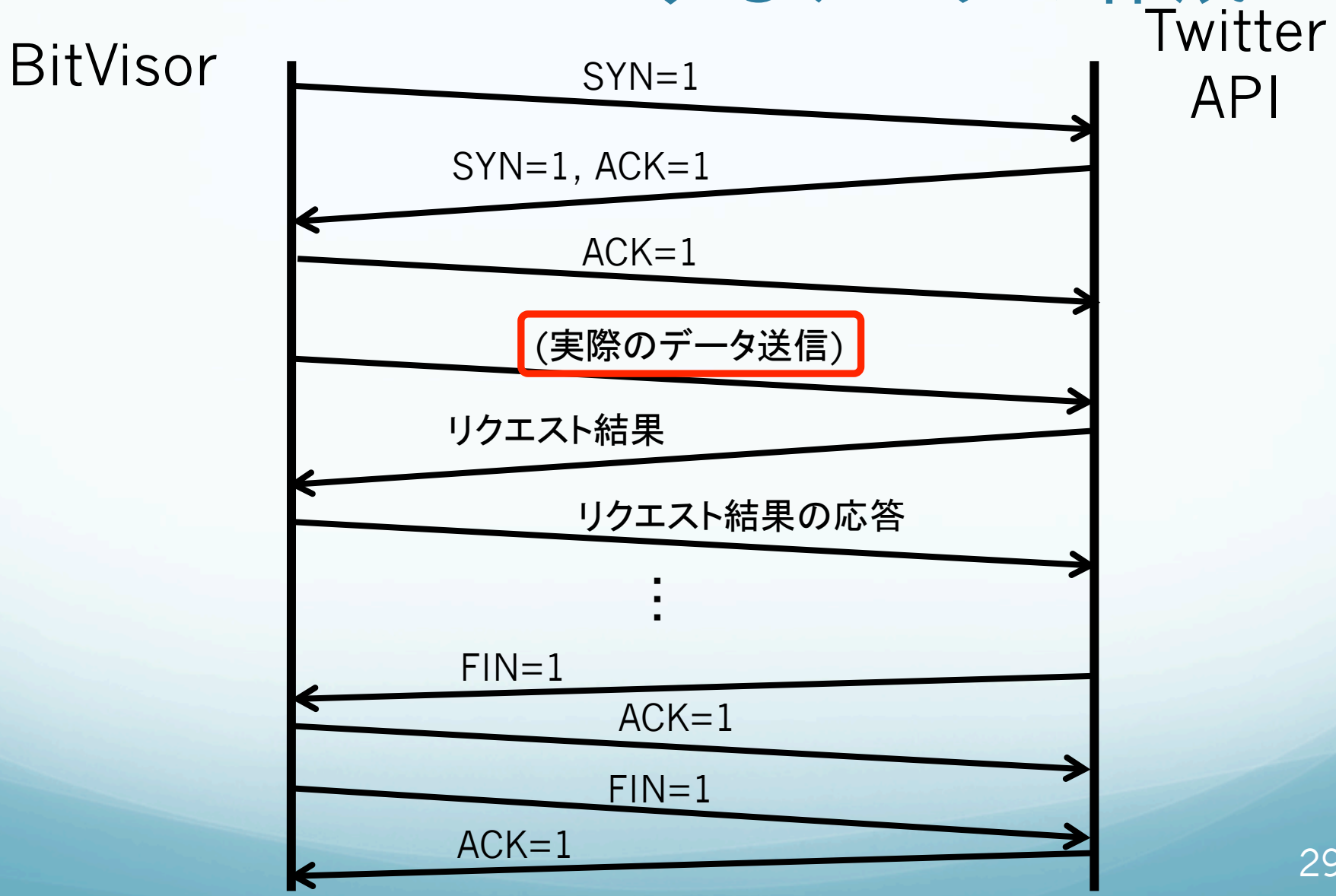
②. ①で送信したリクエスト結果に対してACKを返送

TwitVisorのTCP通信の問題点

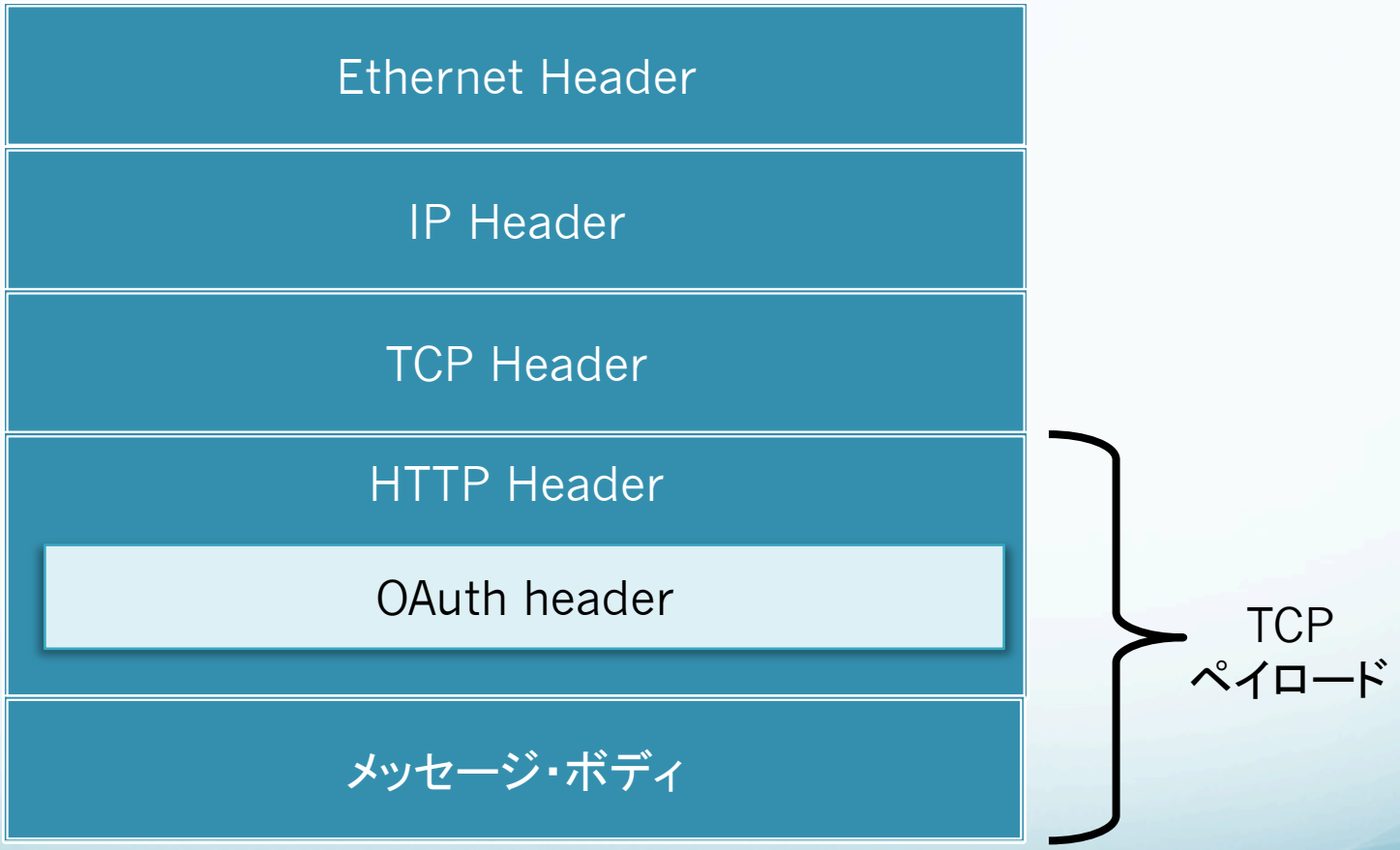
- エラー等に対する処理が全くない
  - 送受信パケットが欠損すると正しく通信できない
  - 再送機能ない

- 送信パケットにフラグメントが起きることを考慮していない
  - 1ツイートは最大でも約1100バイト(日本語で140字の場合)
  - 1文字のツイートで約600バイト
  - 意図的に長い文字列を送信しない限りはツイートのPOSTでフラグメンテーションを気にする必要はない

TwitterにPOSTするデータの作成



Twitterに送信するデータ



TwitterにPOSTするデータの作成

- 前述のhookTcpPacket関数内のtweet()関数でデータを作成する
  - 送信方法は3ウェイハンドシェークと同様
  - TCPヘッダに加えてさらに各ヘッダ，データを追加
    - HTTPヘッダ
    - Oauthヘッダ
    - メッセージ・ボディ
- パケットの送信は3ウェイハンドシェークと同様

HTTPヘッダ &
メッセージ・ボディの作成(1)

- 接続先のURLとリクエストの種類を指定する
- Twitterへの投稿は特定のURLへポストすればよい
  - POST http://api.twitter.com/1.1/statuses/update.json
  - メッセージ・ボディ
    - status=【tweetしたい文字列】
- その他の値はHTTPの規定どおりにヘッダを作成する

OAuthヘッダ

- 必要な情報
  - Consumer Key(アプリ登録時に既知)
  - Consumer Key Secret(同上)
  - Access Token(同上)
  - Access Token Secret(同上)
  - Timestamp
  - Nonce
  - Signature
  - Signature Method(HMAC-SHA1固定)
  - Version(1.0で固定)

BitVisor内で値の取得や計算が必要

Timestampの取得(1)

- TimestampにはUnixtimeを指定する
  - BIOS時刻から取ってくる
  - RTCレジスタの値を取得するget_rtc_register関数を作成

```
#define CMOS_ADDRESS 0x70
#define CMOS_DATA    0x71

unsigned char get_rtc_register(int reg){
        u8 data;
        asm_outb(CMOS_ADDRESS, reg);
        asm_inb(CMOS_DATA, &data);
       return data;
}
```

Timestampの取得(2)

- regの値を変えて年，月，日，時間，分，秒をそれぞれ取得

```
second = get_rtc_register(0x00);
minute = get_rtc_register(0x02);
hour = get_rtc_register(0x04);
day = get_rtc_register(0x07);
month = get_rtc_register(0x08);
year = get_rtc_register(0x09);
```

- 取得した時刻をUnixタイムに変換

Nonce(1)

- 毎リクエストごとにランダムかつ一意な文字列が入る
  - Timestamp値をMD5でハッシュ化
- MD5ハッシュ化はBitVisorに内蔵されているOpenSSLの機能を使用
  - #include <openssl/md5.h>をインクルード

Nonce(2)

- Nonceを取得するget_oauth_nonce関数を作成

```
#include <openssl/md5.h>

void get_oauth_nonce(char *timestamp, char *oauth_nonce){
    u32 i, index = 0;
    char tmp[17];
    const char TO_HEX[] = “0123456789abcdef”;

    MD5(timestamp, strlen(timestamp), tmp);

    for(i = 0; i < MD5_DIGEST_LENGTH; i++){
        oauth_nonce[index] = TO_HEX[tmp[i] / 16];
        oauth_nonce[index + 1] = TO_HEX[tmp[i] % 16];
        index += 2;
    }
}
```

TimestampをMD5でハッシュ化してtmpに格納

数値から文字列に変換してoauth_nonceに格納

Signatureの作成

1. OAuthの仕様にしたがってSignature Base Stringを作成する
   - OAuthのパラメータ等を”&”で連結し文字列を作成する
   - 値は全て既知なのでそのまま連結
   - 連結した文字列をURLエンコーディング
2. Signature Base StringをHMAC-SHA1でハッシュ化
   - キーはConsumer SecretとAccess Token Secretを”&”で連結したもの(いずれも既知)
3. ハッシュ化した文字列をBase64エンコーディングする

URL/Base64エンコーディング

- それぞれurl_encode関数，b64_encode関数を実装

```
void url_encode(const char *src,
        u32 src_len, char *dest, u32 *dest_len);

void b64_encode(const char *src,
        u32 src_len, char *dest, u32 *dest_len);
```

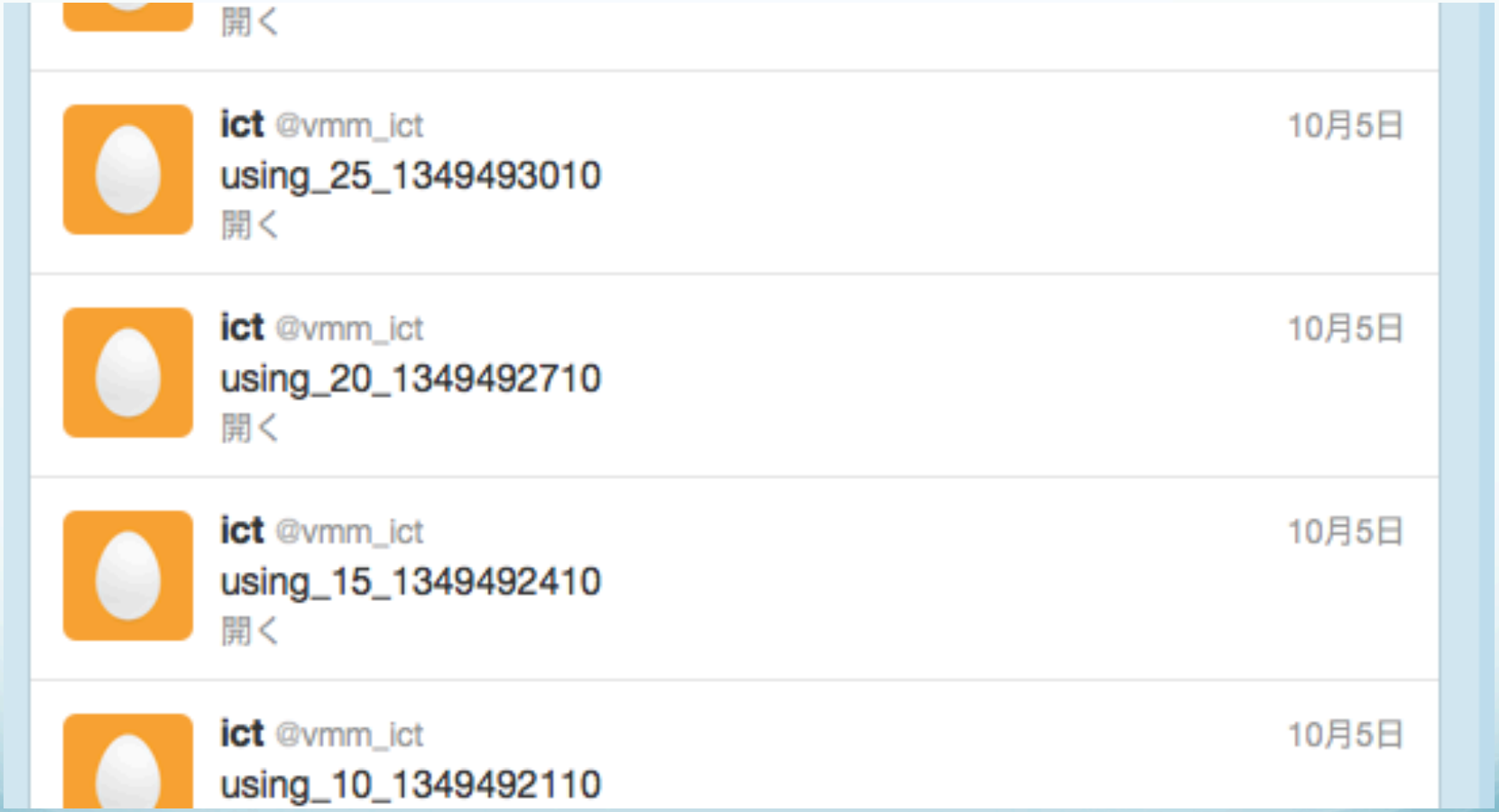
- 各関数内でエンコード規則にしたがってsrcをエンコードしdestに格納する

HMAC-SHA1

- HMAC-SHA1はBitVisorに内蔵されている OpenSSLの機能を使用
  - #include <openssl/hmac.h>をインクルード
  - HMACマクロが使用できる

```
HMAC(
    const EVP_MD EVP_sha1(), //ハッシュ化方法を選択【今回はEVP_sha1()固定】
    const void *key,         //前述のキー
    int key_len,             //キーの長さ
    const u8 *base_str,      //前述のSignature Base String
    size_t base_str_len,     //Signature Base Stringの長さ
    u8 *dest,                //ハッシュ化した値を格納する配列
    u32 *dest_len            //destの長さが格納される
);
```

twitterへの投稿成功



まとめ

- BitVisorからTwitterに投稿する機能を実装
  - BitVisorにTwitterに投稿するための最低限のTCP実装をした
  - OAuthヘッダを作成し，Signature等の各種値の計算をBitVIsor内で行った

＊汎用的なTCP実装とはほど遠い

おわりに

- 本システムは「筑波大学大学院　ICTソリューション・アーキテクト育成プログラム　ソリューション型研究開発プロジェクト」の支援のもと作成しました