TTC技術レポート
Technical Report

TR-1093
HTIPのIoTシステム適用に関する実装指針2
- ショートフレームにおけるフレーム形式 -

Implementation guideline 2
for HTIP on IoT system

～Frame format for short frames～

第1.0版

2022年4月5日制定

一般社団法人

情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

TTC Telecommunication Technology Committee

目次

<参考>

1. 国際勧告等との関連

本技術レポートに関する国際勧告は本文中に記載している。

２．改版の履歴

版数	制定日	改版内容
第1.0版	2022年4月5日	制定

３．参照文章

主に、本文内に記載されたドキュメントを参照したが代表的な文書を下記に示す。

参照文献

[LLDP]　IEEE 802.1ab (2005), Station and Media Access Control Connectivity Discovery

[HTIP]　TTC JJ-300.00 ホームNW接続構成特定プロトコル

[HTIP情報リスト]　TTC JJ-300.01 端末区分情報リスト

[TTC TR-1053]　TTC TR-1053 サービスプラットフォームにおけるカスタマサポート機能

[TTC TR-1057]　TTC TR-1057 ホームネットワークにおけるカスタマサポート機能ガイドライン

[TTC TR-1061]　TTC TR-1061 JJ-300.00機能実装ガイドライン　～非イーサネットデータリンク層、複数LLDPDU、障害切り分け情報対応～

[TTC TR-1062]　TTC TR-1062 ホームネットワークサービスにおけるカスタマサポートユースケース

[TTC TR-1073]　TTC TR-1073 JJ-300.00機能実装ガイドライン　～非IP及び非イーサネット通信技術への対応～

[TTC TR-1086]　TTC TR-1086 HTIP評価ツール及び構築ガイドライン

４．技術レポート作成部門

第１．０版　：　IoTエリアネットワーク専門委員会　(WG3600)

５．本技術レポート「非IP及び非イーサネット通信機器のHTIP機能搭載」の制作体制

本技術レポートは、IoT推進コンソーシアム　スマートIoT推進フォーラム(技術開発WG)　技術戦略検討部会　技術・標準化分科会(分科会長: 丹康雄[JAIST/NICT])において原案を作成し、その後TTC IoTエリアネットワーク専門委員会(委員長：西川 嘉樹[NTT])での審議を経てTTC技術レポートとして公開するものである。

スマートIoT推進フォーラムにおける検討においては、エリアネットワークOAMタスクフォース（リーダー:松倉隆一[富士通])にて作業にあたった。

1. はじめに

本技術レポートは、TTC JJ-300.00, JJ-300.01及び、ITU-T G.9973に基づくホームネットワーク運用管理技術(HTIP)をIoTシステムのIoTエリアネットワークに適用するにあたって留意すべき実装指針を与えることを目的に、HTIPをIoT向け伝送技術で用いられる最大フレーム長の短いフレームに乗せる標準方式について述べたものである。

1.1 背景

JJ-300.00では、第１版においてLLDPDU (Link Layer Description Protocol Data Unit) を用いEthernetフレームで機器情報及びネットワーク構成情報を格納しブロードキャストすることにより、Ethernet(Wi-Fiなどのみなしを含む)で構成されたIoTエリアネットワーク上の機器情報や接続情報を取得可能にすることを目的としていた。第２版では、IEEE802.15.4の様なLLDPDUを直接転送できないデータリンク上でHTIPを使用するため、IPパケットにカプセル化するプロトコルとしてGRE(Generic Routing Encapsulation)を用いる方法を定義した。また、TR-1061「JJ-300.00機能実装ガイドライン ～非イーサネットデータリング層、複数LLDPDU,障害切り分け情報対応～」では、GREを利用してLoWPAN上に存在するネットワーク機器情報をイーサネット上に転送する方法が具体的に示されている。TR-1073「JJ-300.00機能実装ガイドライン ～非IP及び非イーサネット通信技術への対応～」では、非IPかつ非Ethernetリンク技術を用いて構成されるネットワークを通常のHTIP/Ethernetに接続するパターンが具体的に示されている。

IoTエリアネットワークではEthernetフレームおよびGREトンネリングが利用できない伝送技術が多く、これらでHTIPを利用するためにはシリアルラインとして扱うか、小さなフレームサイズに乗せるための標準形式が必要となる。

1.2 スコープ

本技術レポートでは、IoT向け伝送技術でしばしば用いられる短いフレーム長でHTIPを利用可能とする方式について述べる。

非Ethernet・非IPネットワークでHTIPを運用する場合、このネットワークは既存のHTIPネットワークと接続する。このとき、その接続するエリアネットワーク技術は様々なものが考えられる。例えば、Sigfoxを利用する場合、プロトコル依存形式のフレームのやり取りによるデータ伝送となる。また、それらのアプリケーション層が提供する他の通信規格のエミュレーションを利用することもできる。例として、Bluetooth ClassicのSPP(Serial Port Profile)を利用し、HTIP側のManagerと非Ethernet・非IPネットワーク側のエージェントとの間において1対1で通信する場合が挙げられる。これらのケースにおいて、Ethernetフレーム以外の通信フレームに乗せる方法をネイティブフレーム方式と呼ぶこととし、以下では、従来のHTIPで通信される情報をネイティブフレームに乗せる方式について述べる。

1.2.1 ネイティブフレーム方式

ネイティブフレーム方式は、HTIPフレームから必要なデータを抽出し、それを伝送するプロトコルのネイティブフレームに乗せて送信する方式である。LPWAなどのフレームベースのやり取りを行うプロトコルに向いている。ネイティブフレーム形式の概要を図1に示す。

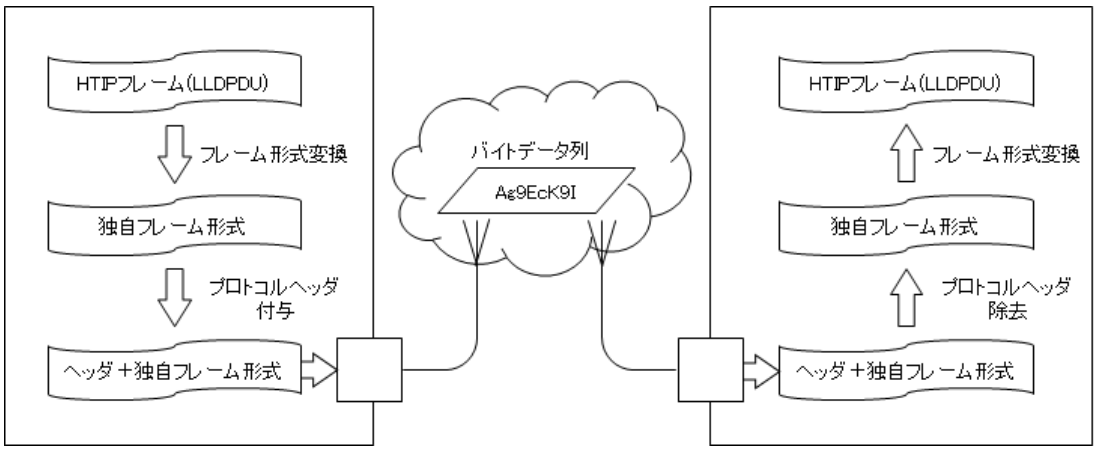

図1. ネイティブフレーム伝送方式

ネイティブフレーム方式では、伝送プロトコルのフレーム長の関係から、HTIPフレームをそのまま送れないことが考えられる。そのため、伝送プロトコルのフレーム長に合わせた短いフレーム形式を定める。詳しくは2章で述べる。

2. ネイティブフレームによる伝送方式

ネイティブフレームはHTIPにおける必要最低限の項目を送信することとし、プロトコルの持つペイロードの長さに応じて24バイトのuplink24と11バイトのuplink11の2種類のフォーマットを定める。

2.1 uplink24

uplink24フォーマットは、Bluetooth SPPの利用を想定した24バイトからなるフォーマットである。

byte	1	2	3	4	5	6	7	8	9	10	11
説明	※1	カウンター値		※2	区分	メーカコード					

byte	12	13	14	15	16	17	18	19	20	21	22	23	24
説明	機器の型番												

図2. uplink24フォーマット

フォーマットの各フィールドについて以下で説明する。

2.1.1 ※1(第1オクテット)

・ 0, 1 ビット　パケットの種類

パケットの種類にかかわる情報を 2bit で格納する。パケットの種類の取りうる値を表に示す。

表1. パケットの種類

パケットの種類	説明
0b00	読み込み可能センサ
0b10	再書き込み可能センサ
0b01	HTIP情報

・ 2 ビット　フレームインデックス

フレームの種類にかかわる情報を 1bit で格納する。フレームインデックスの定義を表に示す。

表2. フレームインデックス

フレームインデックス	説明
0b0	uplink24, 区分・メーカコード(upink11)
0b1	機器の型番(upink11)

・ 3-7 ビット　コマンド長

パケット長から1を引いた数をbyte単位で格納する。HTIP情報の場合、後述する機器の型番の長さが可変長となるため、コマンド長は機器の型番に依存し、最大長は 23 となる。例として、機器の型番が 8 文字の長さの場合、コマンド長は 18 (0b010010)となる。

2.1.2 カウンター値(第2, 第3オクテット)

通信時のパケットエラー率をゲートウェイで算出するためのカウンター値を格納する。カウンター値は機器の再起動時には 0 となり、これ以降パケット送信ごとに 1 ずつインクリメントされる。これにより、連続に送信されたパケットの順序や受信漏れを検出することができる。カウンター値が 2byte の最大値の 65535 でカウントオーバーフローした場合はエラーとし、これ以上カウントさせない。

2.1.3 ※2（第4オクテット）

・ 0-3 ビット　バッテリ残量

デバイスの電源電圧もしくは蓄電電圧を、バッテリ残量としてパーセンテージで表現した値を 4bitで格納する。バッテリ残量の取りうる値を表に示す。

表3. バッテリ残量

バッテリ残量値	説明
0x0	バッテリ残量 0-10%
0x1	バッテリ残量 10-20%
0x2	バッテリ残量 20-30%
・・・	・・・
0x9	バッテリ残量 90-100%
0xa	バッテリ残量 100%
0xf	測定範囲外(マイナス値など)

・ 4-7 ビット　Status

デバイスの動作にかかわる情報を 4bitで格納する。Statusの定義を表に示す。

表4. Status

対象ビット	説明
0b000x	0:光発電による動作(照度 2000lx ＞) 1:EDLCによる動作(照度 2000lx ＜)
0b00x0	0:外部 I2C 通信ポート無 1:外部 I2C 通信ポート有
0b0x00	0:外部 UART 通信ポート無 1:外部 UART 通信ポート無

2.1.4 区分（第5オクテット）

デバイスの種類を 1byteで格納する。このとき、端末区分情報リストを参照して該当する区分を記述するが、端末区分情報リストでの端末区分一覧では値が文字列となっており、1byteで入力できるようなコード化がされていない。そのため、実装の際には情報の送り手と受け手の間でコード化のルールを定める必要がある。

2.1.5 メーカコード（第6-第11オクテット）

デバイスのメーカコードを6文字のASCIIで記述する。メーカコードはIEEEに登録されたカンパニーIDのみを記述する。使用可能な文字は以下である。

[a-fA-F0-9]

2.1.6 機種の型番（第12-最大第24オクテット）

任意長でHTIPにおける型番を最大13文字までのASCIIで記述する。使用可能な文字は以下である。

[a-zA-Z0-9]|[-’()+,./:=?;!*¥#@¥$¥_¥%]

2.2 uplink11

uplink11フォーマットは、Sigfoxといった十分なパケット長が取れない通信規約向けのuplinkパケットのフォーマットである。uplink11では2種類のフレーム形式が存在する。uplink11フォーマットの各フィールドに関しては、分割されている点を除きuplink24フォーマットのそれと同一である。ただし、機器の型番については最大8バイトである。

フレーム形式①

byte	1	2	3	4	5	6	7	8	9	10	11
説明	※1	カウンター値		※2	区分	メーカコード					

フレーム形式②

byte	1	2	3	4	5	6	7	8	9	10	11
説明	※1	カウンター値		機器の型番							

図2. uplink11フォーマット

3. BLEでの実装例

BLEとEthernetが混在する環境において、ネイティブフレームを使用した実装例を示す。

3.1 機器構成と働き

BLE端末はBLEのペリフェラル装置かつHTIPのエンド端末として動作する。BLE端末はHTIP情報をネイティブフレームのuplink24のフォーマットで送信する。

中継器はBLEのセントラル装置かつHTIPのNW機器として動作する。中継器はBLE端末がネイティブフレームで発信するHTIP情報を受け取り、フレーム変換してLLDPDUを作成し、Ethernetで送信する。また、中継器は端末アドレスを記憶するテーブルを有し、BLE端末からアドバタイジングPDUを受信した際に、BLE端末の固有アドレス(Advertiser's Address)を前記テーブルに保管する。中継器は定期的に端末アドレスが保存されたテーブルを参照し、接続構成情報を作成して自身の機器情報とともにEthernetで送信する。

機器構成を図に示す。

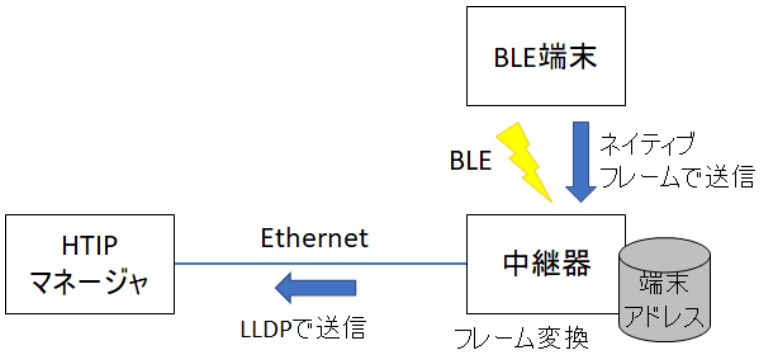

図3. 機器構成

BLEによるHTIP情報の送信方法として、下記の2つの方法が考えられる。

・アドバタイジングPDUを使用してHTIP情報を送信する

・データPDUを使用して、HTIP情報を送信する。

アドバタイジングPDU、データPDUを以下に示す。

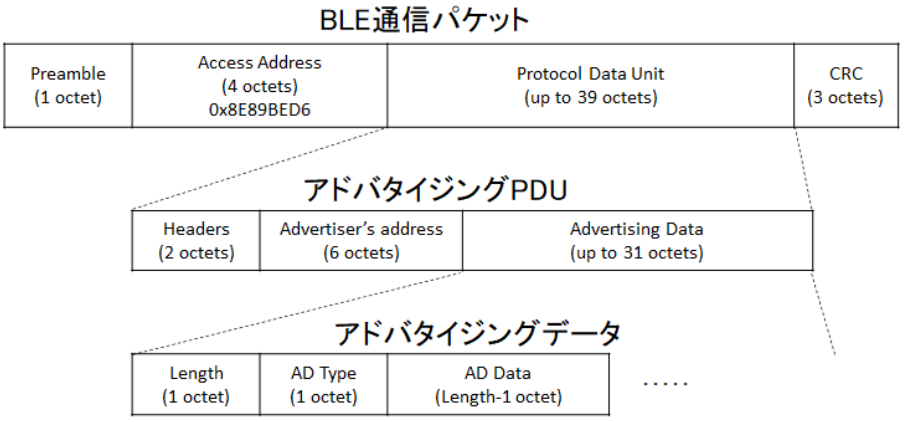

図4. アドバタイジングPDUのフォーマット

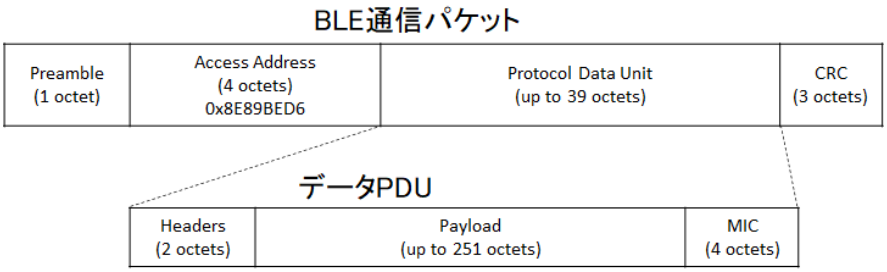

図5. データPDUのフォーマット

3.2 ネイティブフレームで不足する情報

ネイティブフレームでは、エンド端末から限られた情報しか送られてこない。そのため、LLDPで必須であるが、ネイティブフレームのフォーマットに存在しない項目がある。これらの項目について、中継器では以下のように対処する。

・TLV Type=0(End Of LLDPDU)

中継器で0x0000を追加する。

・TLV Type=1(Chassis ID)

BLEからの送信データにある、何かしらの識別情報から中継器で作成し追加する。今回のケースでは、BLEのアドバタイジングPDUに含まれるAdvertiser's addressを利用する。

・TLV Type=2(Port ID)

実装依存であるため、中継器で作成し追加する。

・TLV Type=3(Time To Live)

実装依存であるため、中継器で作成し追加する。

3.3 アドバタイジング PDU を利用した HTIP の送信

アドバタイジングPDUを利用してHTIP情報を送信する場合、アドバタイジングPDUのManufacture Specificデータにネイティブフレームを埋め込む方法によって実現する。Manufacture Specificデータは、製造元やユーザが自由に実装できるエリアである。

アドバタイジングPDUのデータフォーマットは、AD Structureという構造により構成されている。AD Structureは、Length、AD Type、AD Dataの3要素により定義される。LengthにはAD TypeとAD Dataの合計のバイト数が1バイトで格納される。AD Typeには、Manufacture SpecificではOxFFが格納される。AD Dataには、先頭に2バイトのCompany IDが入り、その後ろにカスタムデータが格納される。今回の実装では、このカスタムデータの部分にHTIP情報をネイティブフレームフォーマット形式で格納する。Company IDとはBluetooth SIGによってSIGメンバー企業に割り当てられる固有の番号で、SIGメンバー企業が必要に応じて申請することでCompany IDを割り当ててもらうことができる。Company IDの一覧はBluetooth SIGにより公開されている。Manufacture Specificデータの構成を表5に示す。

表5. Manufacture Specificデータの構成

Length(1バイト)	AD Type(1バイト)	AD Data(2バイト + カスタムデータの長さ)
AD Type の長さ+AD Data の長さ	0xFF	Company ID +カスタムデータ

3.4 データ PDU を利用した HTIP の送信

データPDUを利用してHTIP情報を送信する場合、最初にアドバタイズによってBLE端末と中継器との間でコネクションを作成する。次に、BLE端末がHTIP情報をデータPDUに含めて送信する。

BLEでは、データPDU内の情報の識別にUUIDを利用している。BLEにおけるUUIDは、BluetoothのグローバルコミュニティであるBluetooth SIGが定義している定義済みUUIDと、開発者が独自に割り当てるオリジナルUUIDがある。HTIP情報の送信にはオリジナルUUIDを用いる。

3.5 具体的な使用例(トポロジ表示)

BLE端末とEthernetの混在環境で構成されたネットワークのトポロジ表示を行う場合について説明する。

3.5.1 機器構成

エンド端末に見立てたBLE端末を１つないし複数台設置する。BLE端末からのHTIP情報を受信する中継器を設置する。中継器にはエンド端末であるBLE端末が発信するHTIP情報をフレーム変換してEthernetに送信する機能だけでなく、BLE端末固有の端末アドレスを保存し、その保存した端末アドレスから接続構成情報を組み立て、中継器自身の機器情報に接続構成情報を追加してEthernetに送信する、NW機器としての役割が求められる。中継器とEthernetで接続され、HTIP情報を収集しトポロジ表示を行うHTIPマネージャを設置する。構成を図6に示す。

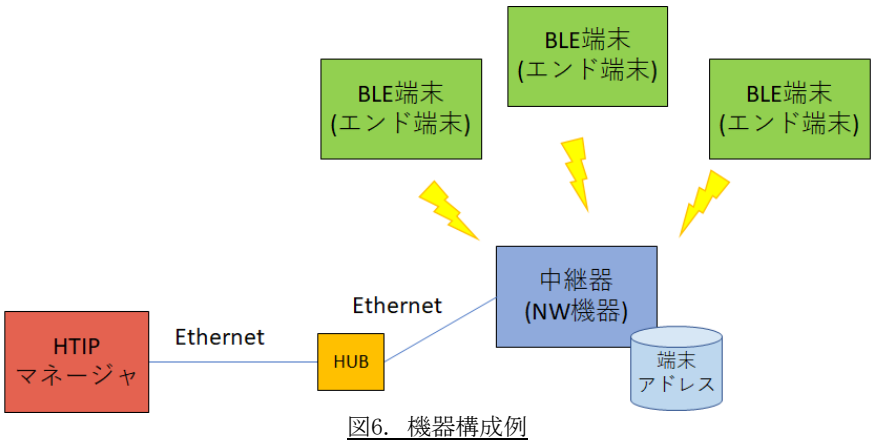

図6. 機器構成例

3.5.2 トポロジ表示における注意点

BLEから受信したアドバタイジングPDUに含まれるAdvertiser's addressをBLE端末の端末アドレスとして利用し、LLDPのchassis IDや接続構成情報でも利用することは前にも述べたとおりである。これらに加えて、Ethernetヘッダの送信元アドレスにも、この端末アドレスを設定する必要がある。Ethernetヘッダの送信元アドレスを中継器のMACアドレスとした場合、HTIPマネージャのトポロジ表示において、BLE端末がunknownとなり、HTIP Agentとして表示されなくなってしまう。

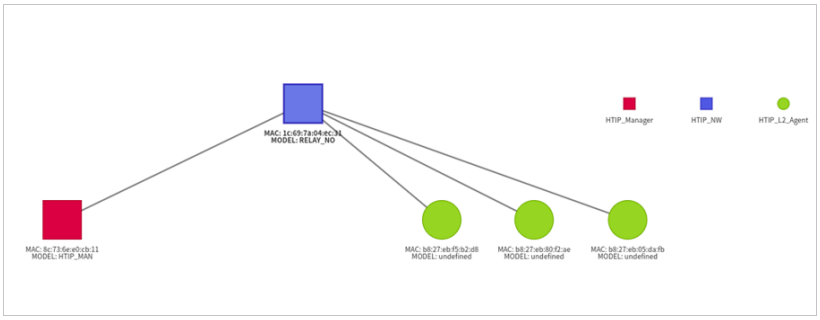

図7. トポロジ表示における注意点

4. まとめ

本技術レポートでは、TTC JJ-300.00, JJ-300.01及び、ITU-T G.9973に基づくIoTエリアネットワーク運用管理技術(HTIP)を非IP及び非Ethernetの通信リンクで利用できるように、標準方式を定め実装例について記載した。これにより、非IP及び非Ethernetの通信リンクが混在する環境下においても、HTIPの利用が可能となる。

付録A. 実装

1. アドバタイジングPDUを利用した実装例

1.1 BLE端末

フリーのBluetoothスタックであるBluez(http://www.bluez.org/)のVer5.61をもとに作成した。Bluezで提供されているPythonで実装されたアドバタイジングのサンプルプログラムに手を加え、Manufacturer Specificのカスタムデータ部分にHTIP情報をネイティブフレームのフォーマットで埋め込んだ。なお、Company IDは0xFFFFとした。

1.2 中継器

アドバタイズのスキャンを行うサンプルコードをもとに、Pythonで作成した。周辺のBLE機器が発信するアドバタイジングPDUを受信し、その中にManufacturer Specificデータがあり、なおかつ内容がHTIPのネイティブフレームであればLLDPDUに変換し、Ethernetで送信するプログラムとした。

2. データPDUを利用した実装例

2.1 BLE端末

アドバタイジング時と同様にBluezをもとに行った。データのやり取りを行うPythonのサンプルプログラムに手を加え、オリジナルUUIDを用いてネイティブフレームのフォーマットでHTIP情報を埋め込んだ。

2.2 中継器

アドバタイズのスキャンを行うサンプルコードをもとに、Pythonで作成した。周辺のBLE機器が発信するアドバタイズを受信後、接続処理をし、オリジナルUUIDと同一のUUIDがあればそのデータをLLDPDUに変換しEthernetで送信するプログラムとした。

付録B. ソースコード

リスト1.a 【BLE端末】アドバタイジングPDUを利用してHTIPを送信するプログラム(抜粋) part1

```
#!/usr/bin/python
# SPDX-License-Identifier: LGPL-2.1-or-later

from __future__ import print_function

import argparse
import dbus
import dbus.exceptions
import dbus.mainloop.glib
import dbus.service
import time
import threading
from gi.repository import GLib
#try:
#    from gi.repository import GObject  # python3
#except ImportError:
#    import gobject as GObject  # python2

mainloop = None
```

リスト1.b 【BLE端末】アドバタイジングPDUを利用してHTIPを送信するプログラム(抜粋) part2

```
class TestAdvertisement(Advertisement):

    def __init__(self, bus, index):
        Advertisement.__init__(self, bus, index, 'peripheral')
        self.add_manufacturer_data(0xffff, [0x4e, 0x00, 0x01, 0x90, 0xff, 0x4a, 0x41, 0x49,
0x53, 0x54, 0x00, 0x56, 0x54, 0x56, 0x30])
        self.add_local_name('htiptest')
```

リスト1.c 【BLE端末】アドバタイジングPDUを利用してHTIPを送信するプログラム(抜粋) part3

```
def main(timeout=0):
    global mainloop

    dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)

    bus = dbus.SystemBus()

    adapter = find_adapter(bus)
    if not adapter:
        print('LEAdvertisingManager1 interface not found')
        return

    adapter_props = dbus.Interface(bus.get_object(BLUEZ_SERVICE_NAME, adapter),
                                   "org.freedesktop.DBus.Properties")

    adapter_props.Set("org.bluez.Adapter1", "Powered", dbus.Boolean(1))

    ad_manager = dbus.Interface(bus.get_object(BLUEZ_SERVICE_NAME, adapter),
                                LE_ADVERTISING_MANAGER_IFACE)

    test_advertisement = TestAdvertisement(bus, 0)

#    mainloop = GObject.MainLoop()
    mainloop = GLib.MainLoop()

    ad_manager.RegisterAdvertisement(test_advertisement.get_path(), {},
                                     reply_handler=register_ad_cb,
                                     error_handler=register_ad_error_cb)

    if timeout > 0:
        threading.Thread(target=shutdown, args=(timeout,)).start()
    else:
        print('Advertising forever...')

    mainloop.run()  # blocks until mainloop.quit() is called

    ad_manager.UnregisterAdvertisement(test_advertisement)
    print('Advertisement unregistered')
    dbus.service.Object.remove_from_connection(test_advertisement)
```

リスト2. 【中継器】アドバタイジングPDUに含まれるHTIPを受信するプログラム（全文）

```
from bluepy.btle import Scanner, DefaultDelegate
import socket
import netifaces

class ScanDelegate(DefaultDelegate):
    def __init__(self):
        DefaultDelegate.__init__(self)

scanner = Scanner().withDelegate(ScanDelegate())
devices = scanner.scan(5.0)

tardev:str = ""
manudata:str = ""
for dev in devices:
    if dev.addr == "b8:27:eb:05:da:fb":
        print("Hit!")
        tardev = dev
        for (adtype, desc, value) in dev.getScanData():
            if desc == "Manufacturer":
                manudata = value
        break

if tardev != "":
    if_data = netifaces.ifaddresses('eno1')
    mac = if_data[netifaces.AF_LINK][0]['addr']
    machex = mac[0:2] + mac[3:5] + mac[6:8] + mac[9:11] + mac[12:14] + mac[15:17]
    tarmac = tardev.addr
    tarhex = tarmac[0:2] + tarmac[3:5] + tarmac[6:8] + tarmac[9:11] + tarmac[12:14] +
tarmac[15:17]

    ETH_P_ALL = 3
    interface = 'eno1'
    dst = b'¥xff¥xff¥xff¥xff¥xff¥xff'   # destination MAC address
    src = bytes.fromhex(machex)
    htip = b'¥x88¥xcc'                  # Ethernet frame type
    tlv0 = b'¥x00¥x00'
    tlv1 = b'¥x02¥x07¥04' + bytes.fromhex(tarhex)
    tlv2 = b'¥x04¥x07¥03' + bytes.fromhex(tarhex)
    tlv3 = b'¥x06¥x02¥x00¥x3c'
    ttc1 = b'¥xfe¥x0a¥xe0¥x27¥x1a¥x01¥x01¥x04' + 'MISC'.encode()
    ttc4 = b'¥xfe¥x0a¥xe0¥x27¥x1a¥x01¥x04¥x04' + bytes.fromhex(manudata[26:34])
    senddata = dst + src + htip + tlv1 + tlv2 + tlv3 + ttc1 + ttc4 + tlv0
    print(senddata)

    s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(ETH_P_ALL))
    s.bind((interface, 0))
    s.sendall(senddata)

    print('Sent!')
    s.close()

else:
    print("Not Found.")
```

リスト3.a 【BLE端末】データPDUを利用してHTIPを送信するプログラム(抜粋) part1

```
class Application(dbus.service.Object):
    """
    org.bluez.GattApplication1 interface implementation
    """
    def __init__(self, bus):
        self.path = '/'
        self.services = []
        dbus.service.Object.__init__(self, bus, self.path)
        self.add_service(TestService(bus, 2))

    def get_path(self):
        return dbus.ObjectPath(self.path)

    def add_service(self, service):
        self.services.append(service)

    @dbus.service.method(DBUS_OM_IFACE, out_signature='a{oa{sa{sv}}}')
    def GetManagedObjects(self):
        response = {}
        print('GetManagedObjects')

        for service in self.services:
            response[service.get_path()] = service.get_properties()
            chrcs = service.get_characteristics()
            for chrc in chrcs:
                response[chrc.get_path()] = chrc.get_properties()
                descs = chrc.get_descriptors()
                for desc in descs:
                    response[desc.get_path()] = desc.get_properties()

        return response
```

リスト3.b 【BLE端末】データPDUを利用してHTIPを送信するプログラム(抜粋) part2

```
class TestService(Service):
    """
    Dummy test service that provides characteristics and descriptors that
    exercise various API functionality.

    """
    TEST_SVC_UUID = '12345678-1234-5678-1234-56789abcdef0'

    def __init__(self, bus, index):
        Service.__init__(self, bus, index, self.TEST_SVC_UUID, True)
        self.add_characteristic(TestCharacteristic(bus, 0, self))
        #self.add_characteristic(TestEncryptCharacteristic(bus, 1, self))
        #self.add_characteristic(TestSecureCharacteristic(bus, 2, self))

class TestCharacteristic(Characteristic):
    """
    Dummy test characteristic. Allows writing arbitrary bytes to its value, and
    contains "extended properties", as well as a test descriptor.

    """
    TEST_CHRC_UUID = '12345678-1234-5678-1234-56789abcdef1'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.TEST_CHRC_UUID,
                #['read', 'write', 'writable-auxiliaries'],
                ['read'],
                service)
        self.value = [0x4e, 0x00, 0x01, 0x90, 0xff, 0x4a, 0x41, 0x49, 0x53, 0x54, 0x00, 0x56,
0x54, 0x56, 0x30]
        self.add_descriptor(TestDescriptor(bus, 0, self))
        self.add_descriptor(
                CharacteristicUserDescriptionDescriptor(bus, 1, self))

    def ReadValue(self, options):
        print('TestCharacteristic Read: ' + repr(self.value))
        return self.value

    def WriteValue(self, value, options):
        print('TestCharacteristic Write: ' + repr(value))
        self.value = value
```

リスト4. 【中継器】データPDUに含まれるHTIPを受信するプログラム（全文）

```
from bluepy import btle
import socket
import netifaces

MAC_ADDRESS = 'b8:27:eb:05:da:fb'
SERVICE_UUID = '12345678-1234-5678-1234-56789abcdef0'
READ_UUID = '12345678-1234-5678-1234-56789abcdef1'

scanner = btle.Scanner(0)
devices = scanner.scan(5.0)

tardev:str = ""
manudata:str = ""
for dev in devices:
    if dev.addr == MAC_ADDRESS:
        tardev = dev
        print(f'MACアドレス:{dev.addr}')

if tardev != "":
    peripheral = btle.Peripheral()
    peripheral.connect(MAC_ADDRESS)

    service = peripheral.getServiceByUUID(SERVICE_UUID)
    characteristic = peripheral.getCharacteristics(uuid=READ_UUID)

    manudata = characteristic[0].read()
    peripheral.disconnect()

    if_data = netifaces.ifaddresses('eno1')
    mac = if_data[netifaces.AF_LINK][0]['addr']
    machex = mac[0:2] + mac[3:5] + mac[6:8] + mac[9:11] + mac[12:14] + mac[15:17]
    tarmac = tardev.addr
    tarhex = tarmac[0:2] + tarmac[3:5] + tarmac[6:8] + tarmac[9:11] + tarmac[12:14] +
tarmac[15:17]

    ETH_P_ALL = 3
    interface = 'eno1'
    dst = b'¥xff¥xff¥xff¥xff¥xff¥xff'   # destination MAC address
    src = bytes.fromhex(machex)
    htip = b'¥x88¥xcc'                  # Ethernet frame type
    tlv0 = b'¥x00¥x00'
    tlv1 = b'¥x02¥x07¥04' + bytes.fromhex(tarhex)
    tlv2 = b'¥x04¥x07¥03' + bytes.fromhex(tarhex)
    tlv3 = b'¥x06¥x02¥x00¥x3c'
    ttc1 = b'¥xfe¥x0a¥xe0¥x27¥x1a¥x01¥x01¥x04' + 'MISC'.encode()
    ttc4 = b'¥xfe¥x0a¥xe0¥x27¥x1a¥x01¥x04¥x04' + manudata[11:]
    senddata = dst + src + htip + tlv1 + tlv2 + tlv3 + ttc1 + ttc4 + tlv0
    print(senddata)

    s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(ETH_P_ALL))
    s.bind((interface, 0))
    s.sendall(senddata)

    print('Sent!')
    s.close()
else:
    print("Not Found.")
```