低コスト IoT 機器の管理運用技術に関する研究

1710214　山本　遥平

主指導教員　　丹　康雄
審査委員主査　リム　勇仁
審査委員　　篠田　陽一
　　知念　賢一

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和元年 5 月

概　要

In recent years, IoT is becoming mainstream of internetworking as development of various protocols which suit for low-cost devices. In recent IoT network, lots of sensor or home appliance, such as temperature, humidity sensor, smart lamp, IoT-enabled Air-Conditioner, connect to network. To connect conventional items that is not intended to network, network module is widely used. Network module is small electrical circuit including physical port like RJ45 and CPU needed to conduct network protocol procedures.

As number of these items get far enormous compared to conventional one, demand to reduce cost of IoT device is increasing. To reduce cost of those items, capacity of IoT device tend to be low compared to conventional items. Because of low capacity of IoT devices, it often has small CPU capacity and restricted communication protocol. In recent IoT network, area network protocol is commonly used because of its low cost and its aptitude for low-influent small-frame length communication.

Those IoT area network protocol often have no Management and operation system. Because of that, most IoT are network is lack of stability and maintainability and reliability.

In this paper, we try to introduce low-cost Management and Operation system to IoT area network. We choose HTIP as Management and Operation system to be implemented.

HTIP is stands for Home-network Topology Identification Protocol, which aim to earn topology information of home network. This protocol has 3 types of HTIP node. HTIP-End Device, HTIP-NW device and HTIP-Manager. HTIP -End Device is correspond to normal user equipment in network, for example, PC, smartphone, smart lamp and IoT-enabled Air-Conditioners. In those HTIP-End devices, program called Agent is running as task. Agent program collect HTIP-end device specific data such as, class of HTIP-end device, name of device and so on. HTIP-NW device is basically layer 2 switches, but it can transfer HTIP frame. HTIP Manager's work is to visualize network topology using information transferred from HTIP-end devices and HTIP-NW devices.

Adapting HTIP as Management and Operation system of IoT area networks is difficult because of those technical problems.

First problem is IoT area network is based on non-Ethernet non-IP network. HTIP is intended for ordinary Ethernet・IP network.

Second problem is that HTIP enabled network device is hard to obtain. HTIP is under-developing protocol and HTIP enabled device is planned to release in few years. But HTIP is still no popular among HTIP devices.

To solve first problem, we try to define HTIP protocol dedicated for Non-Ethernet Non-IP Area network. In this paper, we arrange the architectures

of possible connection between different network protocol. Based on this we defined 2 types of data transmission method, native-frame method and pseudo-serial communication method.

Native-frame method uses frame format which is defined in area network protocol. In data transition, HTIP data is carried as payload of those protocol defined frame. We enact payload format of native-frame method based on PRISM project proposed specification and add some improvement by adding missing but necessary HTIP device data. Frame index is also added for future extension.

To solve second problem, we propose ease method to make ordinary management function enable network device, such as layer 2 switch and modem, HTIP enable by connecting external HTIP agent. This external HTIP agent obtain device data via serial or Ethernet connection and generate and broadcast HTIP frame instead of attached network devices. In this paper, we proposed the attachment case of external HTIP agent in various topology and network devices.

In conclusion, the present study has demonstrated that HTIP is applicable for Management and Operation system of IoT area network. Study also shows the technical method and part of system is implemented as experimental device. For future, full stack implementation of non-Ethernet non-IP HTIP for evaluation and measure behavior of network depends on varying of parameter.

目次

図目次

表目次

第1章 はじめに

この章では、当研究の背景、目的と論文の構成を述べる。

1.1 研究背景

近年、IoT(Internet of Things)が、ネットワークにおける潮流となっている。従来型ネットワークと IoT ネットワークの違いは、その数と性能にある。

従来型ネットワークでは、PC やスマートフォン、ネットワークプリンタなどの情報機器やルーター・スイッチといったネットワーク機器により構成され、それらの構成要素の数は少ない。IoT ネットワークでは、従来型のそれに加え、ネットワーク接続対応家電機器(スマート家電)、屋内・屋外に多数設置される温度・湿度・照度などのセンサ群を構成要素としてもつ。そのため、IoT 機器の構成要素数は、従来型のそれと比べて格段に大きくなる。

また、IoT ネットワークにおける機器は、その数の多さゆえ一つ一つのコストを下げることが求められる。そのため、IoT 対応機器は、その性能・機能の点で従来型ネットワークにつながる機器と比べ性能・機能で劣る場合が多い。例えば、CPU の処理能力が制限される、RAM 容量が少ない、LPWA のような、低速・安価な IoT 向けエリアネットワーク規格が重点的に使われる。

また、IoT ネットワークでは IoT 対応機器が大量に接続される。とくに IoT 向けのエリアネットワーク技術では、管理運用技術が実装されていない規格も多く、ネットワークに存在するノード数の多さもあり、ネットワークの保守性・信頼性が大きく損なわれている。このことから、これら IoT ネットワークの特性に考慮した管理運用技術が必要とされる。

1.2 研究目的

当研究では、IoT エリアネットワークにおける、低コスト管理運用技術の実現を目的とする。これにより、既存の管理運用技術を用いる既存ネットワークとの連携をとりながら、IoT デバイスのような計算能力・通信能力で制限のあるノードで構築された IoT 向けエリアネットワークの管理運用を可能とし、保守性・信

頼性の向上につながる。

1.3 研究手法

当研究では、ホームネットワーク向けトポロジ取得プロトコルである HTIP(Home Network Topology Identifying Protocol)を管理運用技術として採用する。HTIP は、動作が軽量である点から IoT 管理運用技術として適しているが、これを IoT 向けエリアネットワークで運用するには技術的な問題が存在するため、これらを解決するための方法について検討する。

1 つ目は、非 Ethernet・非 IP 向け HTIP（以下、Non-Ether HTIP）の提案である。Non-Ether HTIP は、IoT ネットワークにおいてよく用いられる非 Ethernet・非 IP の無線通信技術に対応させるために設計された HTIP である。Non-Ether HTIP を使うことで、ZigBee、LPWA といった IoT 向け無線通信規格によって接続された IoT 対応機器に対して、HTIP を適用することができる。それに加え、HTIP フレームの内容をバイト列とみなすことで、RS232・RS485 のような有線シリアル通信で接続された IoT 機器に対しても HTIP が適用できるようになる。

2 つ目は、NW 機器の HTIP 化である。L2 スイッチと PLC モデムを NW 機器の代表とみなし、それらの HTIP 対応化について述べる。この方法として、HTIP の通信を担当する外付けの機器とネットワーク機器を接続させ、これら 2 つを 1 つの HTIP 対応機器として扱う方法を述べる。

第2章 関連技術

2.1 LLDP

LLDP(Link Layer Discovery Protocol)は [1]、ネットワーク上のある機器において有線ケーブルもしくは無線通信路によって接続され、隣り合った状態にある機器の情報を取得するプロトコルである。LLDP はレイヤ 2 のプロトコルであり、LLDPDU とよばれる Ethernet フレームによって情報をやりとりする。LLDPDU のフレーム構成を図に示す。LLDPDU では、データは TLV 形式データの連続として記述される。TLV は、データ型(Type)、データ長(Length)、値(Value)の順番でデータを記述する方式である。TLV の構造を表 1 に、LLDPDU のデータを表 2 に示す。

表 1 TLV 書式

データ型	データ長	値
7 bit	9 bit	0-511 octets

表 2 LLDPDU のフレーム構造

TLV type	データ名称	実装必須/オプション
0	End of LLDPDU	必須
1	Chassis ID	必須
2	Port ID	必須
3	Time To Live	必須
4	Port Description	オプション
5	System Name	オプション
6	System Description	オプション
7	System Capabilities	オプション
8	Management Address	オプション
9-126	予約	

127	Custom TLVs	オプション

2.2 UPnP

UPnP(Universal Plug and Play)は、ネットワークに接続した機器の設定の手順を規定したものである。DHCP,HTTP などのプロトコルの集合からなり、UPnP としての手順を定義している。後述の HTIP では、ネットワークに接続されたデバイスの情報を記述する DDD(Device Description Document)を利用して機器情報の伝達を行っている。

2.3 HTIP

HTIP(Home-Network Topology Identify Protocol)は、ホームネットワークを対象としたトポロジ情報取得プロトコルである。 [2]　ホームネットワークにおいて、トラブルシューティング時にトラブル原因切り分けを目的とする。日本国内では TTC(情報通信技術委員会)JJ-300.00, 01 規格として 2010 年に国内標準化され、その後、ITU-T G.9973 として 2011 年には国際標準化も完了している。当初はブロードバンドサービス向けということでパソコンやゲーム機、レコーダーといった計算資源に余裕のある端末向けの規格であったが、その後スマートメーター等省性能機器の利用も想定し、改版を重ね、2017 年には第三版となっている。

HTIP では、以下の 2 種類の情報をもとにネットワークトポロジを生成する。

機器情報は、ネットワーク内のある機器固有の情報を表す。後述する Agent が情報収集と情報送信を行う。機器情報の一覧を表 3 に示す。このうち、区分・メーカーコード・機種名・型番は必須となる。

表 3　HTIP 機器情報詳細

名称	説明
区分	機器の種類を示す. JJ-300.01 で定義される
メーカーコード	IEEE OUI コードで定義される
機種名	ASCII 文字使用可能

型番	ASCII 文字使用可能
チャネル使用状態情報	0~100 で表現 オプション実装
電波強度情報	0~100 で表現 オプション実装
通信エラー率情報	0~100 で実装 オプション実装
ステータス情報	0 は機器の正常動作を表す。 オプション実装
LLDPDU 送信間隔	秒単位 オプション実装

接続構成情報は、ある機器と、それと物理的に接続関係にある別機器との間の接続情報を表す。データとしては、直接接続されている相手機器の MAC アドレスと、それに対応する自分のポート番号の対応表である。後述する HTIP-NW 機器内部の Agent が生成と送信を行う。

HTIP の構成要素として、HTIP-エンド端末、HTIP-NW 機器が定義されている。HTIP の構成例を図 1 に示す。

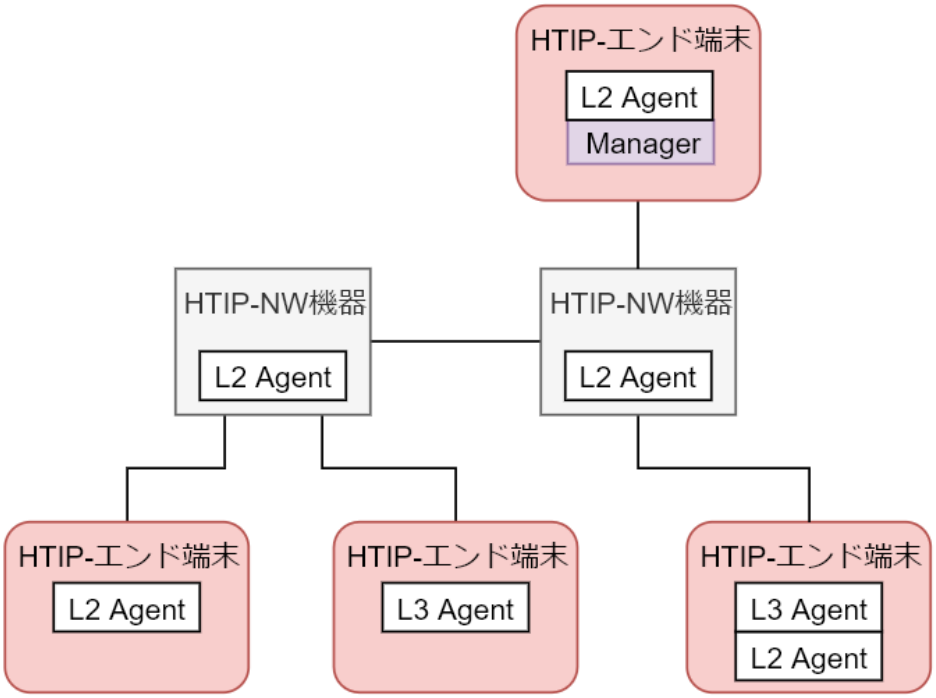


図 1 HTIP 構成要素

HTIP-エンド端末は、ネットワークにおける PC や家電製品などのユーザーが使用する端末にあたる。HTIP-エンド端末は L2-Agent もしくは L3-Agent を実装しなければならない。HTIP-エンド端末は、自らの機器情報を収集し、L2・L3 Agent を介して Manager に送信する。

HTIP-NW 機器は、ポートを 2 つ以上持ち、フレームのフォワーディングを行うことができるネットワーク機器である。おもに L2 スイッチが相当する。NW 機器は、FDB(Forwarding Database)を持ちフォワーディングを行うことと、L2-Agent を実装する必要がある。NW 機器は、自らの機器情報と近接機器情報を収集し、L2-Agent を介して Manager に送信する。接続機器情報は、FDB を基に生成される。

HTIP-Manager は、エンド端末と NW 機器から情報を収集し、トポロジ情報を構成する端末もしくはタスクである。Manager は、エンド機器や NW 機器内部でタスクとして動作する。Manager は、他の機器の Agent から送られてきた機器情報と接続構成情報からトポロジを生成する。ただし、Manager が動作するエンド機器もしくは NW 機器自体の情報は直接取得できるものとする。

Agent は、自分が所属する機器の機器情報と接続構成情報を収集し、Manager に送信する役割をもつタスクである。Agent は Manager に機器情報と接続構成情報を、LLDP による方法と UPnP による方法で伝達することができる。LLDP による方法をとる Agent を L2-Agent、UPnP による方法をとるものを L3-Agent と呼ぶ。すべての Agent は、L2-Agent と L3-Agent の片方もしくは両方を実装しなくてはならない。

L2 Agent は、LLDP を用いて機器情報もしくは接続構成情報を伝達する Agent である。L2 Agent は、LLDP のベンダ拡張フィールドに HTIP 固有の情報を図 3 の形式の TLV で記述し、それを送信する。このベンダ拡張フィールドの TLV type は 127 となる。ベンダ拡張フィールドに収容されるデータを表に示す。イーサネットヘッダの宛先 MAC アドレスは、ブロードキャストアドレスである FF-FF-FF-FF-FF-FF に設定される。LLDPDU のフレーム構成を図 2 に示す。

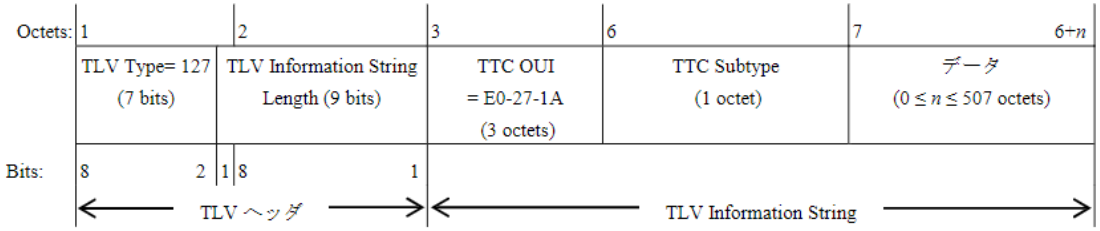


図 2　HTIP 固有情報をベンダ拡張フィールドに埋め込む書式

表 4　ベンダ拡張フィールドのデータ一覧

TCC Subtype	データ内容	実装の必須/推奨/オプション
1	機器情報	必須
2	接続構成情報	NW 機器で必須
3	MAC アドレスリスト	オプション
4	拡張接続構成情報	NW 機器でオプション
5	拡張 MAC アドレスリスト	NW 機器でオプション
0, 6-255	予約領域	

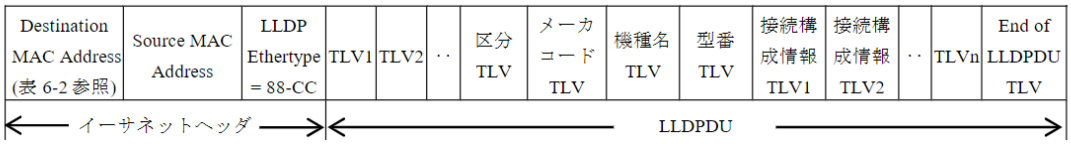


図 3 LLDPDU フレーム構成

L3 Agent は、UPnP を用いて機器情報を伝達する Agent である。L3 Agent は、UPnP Controlled device 機能を用いて機器情報を伝達する。L3 Agent は、NW 機器に存在しないので、接続構成情報を伝達することはない。L3 Agent は、DDD の root device 部の Basic Device Information 部に HTIP 独自データを示すエレメントを追加し、これを Manager に通知する。DDD は L3 Agent に一つだけ存在する。

2.4 RS232/422/485

RS232/422/485 は工業・商用で広く使われるシリアル通信規格である。 [2] 2 線または 4 線配線の通信が可能である。それぞれの差異を表 1 に示す。

表 5 RS232/422/485 の差異

	RS232	RS422	RS485
トポロジ	1 対 1	1 対他(~10 受信機)	多対多(~32 送受信機)
伝送方式	シングルエンド信号	差分信号	差分信号
伝送距離	15m	1200m	1200m

RS232 は、ネットワーク機器・小型コンピューターと管理用 PC との接続に用いられる。RS422/485 は、耐ノイズ性能や複数受信機をもてる性質を生かし、工業・商業ネットワークに用いられる。

第3章 IoT 向けエリアネットワークへの HTIP 適用とその課題

3.1 非 Ethernet・非 IP ネットワークにおける管理運用技術としての HTIP の適用検討

HTIP は、当初ブロードバンドサービス向けとして設計された規格である。そのため、Ethernet・IP ネットワークで用いることを前提とした設計となっている。例として、HTIP は、LLDP プロトコルのフレーム(LLDPDU)の拡張データで伝送する。この LLDP は、Ethernet フレームを用いてフレームのやり取りを行うことが前提の規格となってしまっている。そのため、HTIP は非 Ethernet ネットワークでは使用不可能となっている。

近年、当規格はスマートメーターへの適用を念頭に置き、GRE トンネリングを用いて LLDP フレームを IP/6LoWPAN 対応エリアネットワークで伝送する方式が策定されている。ただし、IoT 向けエリアネットワーク技術の主流は非 IP なので、この方式を管理運用技術として適用することはできない。

当研究では、この Ethernet フレーム・IP パケットの形式に依存しない汎用的手法で、HTIP を利用可能にするための手法を検討する。これにより、IoT 向けエリアネットワークにおいて、HTIP の利点を生かし、安価なノード群で構成されるネットワークにおいて管理運用技術の運用が可能となり、IoT ネットワークの保守性・信頼性を向上させることができる。

3.2 既存ネットワーク機器の HTIP 対応手法の検討

HTIP ネットワークでは、対象となるブロードキャストドメイン内部の全ての機器が HTIP エージェントを実装しなくてはならない。エンド機器の HTIP 対応については、Linux や組み込み PC 向けのオープンソース実装が進められており、スイッチやブリッジなどのネットワーク機器に関しても、HTIP 対応の通信モジュール、Ethernet スイッチや Wi-Fi AP の発売が予定されている。

しかし、HTIP 規格の一般化とそれに伴う対応機器のラインナップには時間がかかると見られている。特にネットワーク機器に関して、HTIP 実装にはファームウェアの変更が必須であり、そのファームウェア更新はネットワーク機器のメーカが更新ファイルを作成するのを待つしかない。そのため、ユーザーが HTIP ネットワーク構築するときの障害となっている。

当研究では、管理運用機能を有する既存ネットワーク機器に対する HTIP 実装の手法を検討する。その方法として、既存ネットワーク機器の通信インターフェースに HTIP を実装し、容易に HTIP を実現する手法について検討を行う。これにより、外付け機器をインターフェースを介してネットワーク機器に接続する作業で HTIP 対応を行うことができ、HTIP 対応のコストや作業時間を短縮することができる。

第4章 非 Ethernet・非 IP ネットワークにおける HTIP の運用

4.1 非 Ethernet・非 IP ネットワークにおける HTIP 運用

非 Ethernet・非 IP ネットワーク、Ethernet・IP ネットワーク同士を連携させたネットワークにおいて、共通管理運用技術として HTIP を用いる場合、二つのネットワークの中継機が必要となる。この中継器は、接続されるネットワークのフレーム形式の差異を吸収するようにフレームに処理を行う必要がある。

非 Ethernet・非 IP ネットワークと Ethernet・IP ネットワークの連携例を図 4 に示す。右側の Ethernet・IP ネットワークと左側の IoT エリアネットワークが AP を介して接続されている。IP・Ethernet ネットワークに存在する HTIP マネージャーは、両方のネットワークのトポロジ・管理運用情報を取得できる必要がある。

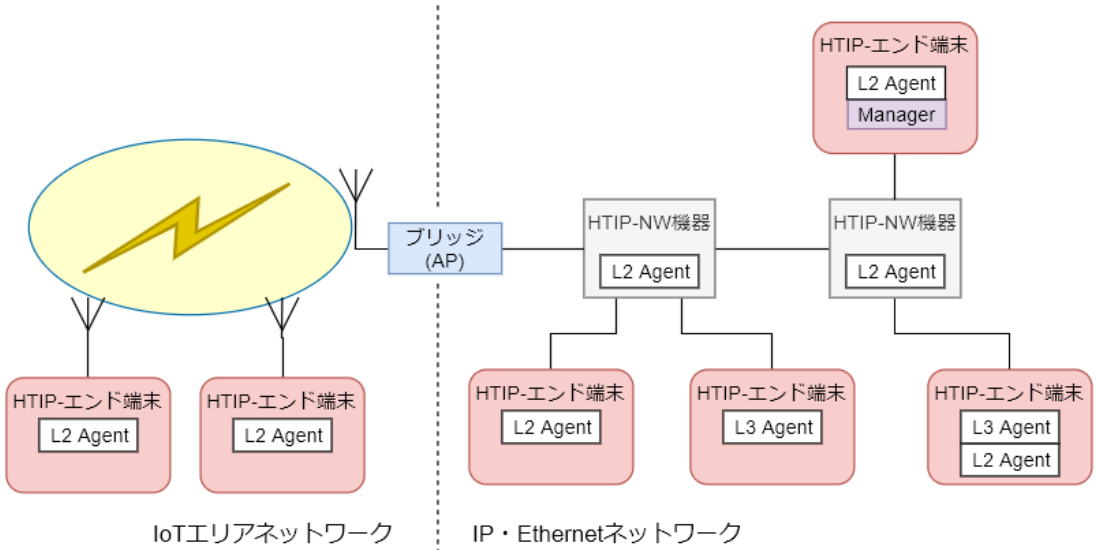


図 4 非 Ethernet・非 IP ネットワークと Ethernet・IP ネットワークの運用例

4.2 ネットワークアーキテクチャの検討

前述したように、Non-Ether HTIP は非 Ethernet・非 IP ネットワークで運用され、このネットワークは別のプロトコル体系をもつ Ethernet・IP ネットワークに接続され、協調動作をしながら運用される。そのため、異なるプロトコルをもつネットワーク同士が接続する場合の接続のアーキテクチャ、運用手法や考えられる障害といったことを考慮する必要がある。

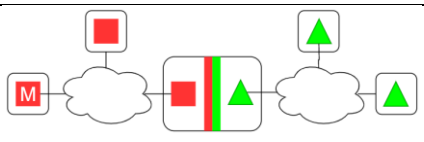
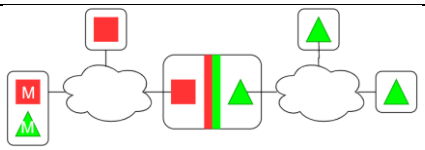
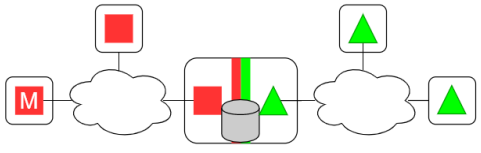
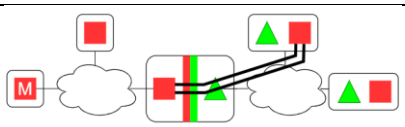
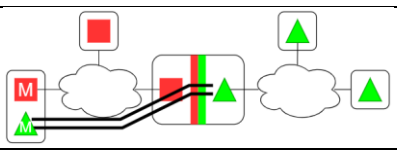
ここでは、異種ネットワークを接続するときのアーキテクチャについて述べる。

表 6 は、二つの異なるプロトコルをもつネット枠を接続する場合の接続形態である。緑・赤のそれぞれ別のプロトコルによって構築されたネットワークが中継機とよばれる機器によって接続されている。それぞれのプロトコルに関係する動作実体は赤四角形または緑三角形で表現される。これらの動作実体は、所属する機器においてタスクとして存在する。この動作実体が機器に存在するということは、そのネットワークが使用するプロトコルのマネジメント機能を通して見ることができることを意味する。これをエージェントと呼ぶ。

マネジメント機能をもつ管理実体は、白抜きの M をつけて表現される。これは、エージェントのマネジメント機能から見えるという機能に加えて、自分自身の持つ機能をもって、ほかのエージェントの情報を参照できるということを表す。これをマネージャーと呼ぶ。

中継器は、そこに到着するパケットもしくはフレームに対して、フレーム変換などの操作を行うことで二つのネットワークの接続を行う。この中継機が行う動作によって 3 種類の接続形態が考えられる。

表 6　中継器を用いた別プロトコルネットワークの接続形態

ステートレスプロトコル変換		
ステートフルプロトコル変換		
トンネリング		

パターン 1 は、中継器がステートレス動作をし、パケット(フレーム)の変換を行うパターンである。中継機はパケットの変換を行う時、フレームに対して編集を行った後、直ちにパケットを送信する。

パターン 1 において、考えられる接続形態が 3 つ挙げられる。

パターン 1-A は、中継機がパケットの単純変換を行う接続形態である。2 つのプロトコルのパケット書式について、持っているデータ内容が同じであり、パケット上でその並びが異なるといった程度の違いしかない場合、このパターンとなる。パケット中継機は、受け取ったパケットを変換先パケットの書式に合わせてデータの組み換えを行えばよい。赤プロトコルのマネージャーは赤・緑プロトコル両方で得られたマネジメント情報を一括して受け取る。

パターン 1-B は、中継器がアプリケーショントンネリングを行う接続形態である。2 つのプロトコルがもつパケット書式について、持っているデータの内容や書式に違いがあり、パターン 1-A のような変換ができない場合のパターンとなる。接続形態上の違いとして、一つの機器に赤・緑プロトコルのマネージャーが共存していることがあげられる。これらのマネージャーは、赤ネットワークからマネジメント情報を直接得ることはできるが、緑ネットワークのマネジメント情報を得るには中継器のアプリケーショントンネリングを用いらなくてはならい。中継器は、緑ネットワークからのパケットを受信すると、これのデータ構造を変えずに赤プロトコルのパケットに乗せて右側ネットワークに送る。この方法として、緑プロトコルのパケットをそのまま赤プロトコルパケットのペイロードとして乗せるといったものがある。右側プロトコルのマネージャーは、到着したパケットから右側プロトコルの情報を取り出し、自分にそれを記録する。この方法を、GRE などの汎用的なトンネリング技術を使わず、プロトコル独自の方法を用いることからアプリケーショントンネリングと呼ぶ。

パターン 1 の通信の流れを図 5 に示す。エージェントとマネージャーが交換機を経由して直接通信するのが特徴である。

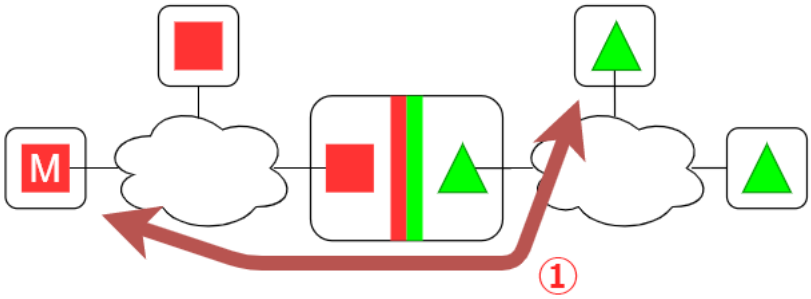


図 5　パターン 1 の通信手順

パターン 2 は、中継器がステートフル動作をし、パケット(フレーム)の変換を行うパターンである。パターン 1 との違いは、パターン 1 では受け取ったパケットは直ちに変換し送出されるのに対し、パターン 2 では受け取ったパケットを一旦メモリに保存し、通信の手順が終了するのを待ってから、保存されたパケットをまとめて変換・送出する。

パターン 2 の通信の流れを図 6 に示す。中継器を境界として通信が 2 段階に分かれるのが特徴である。

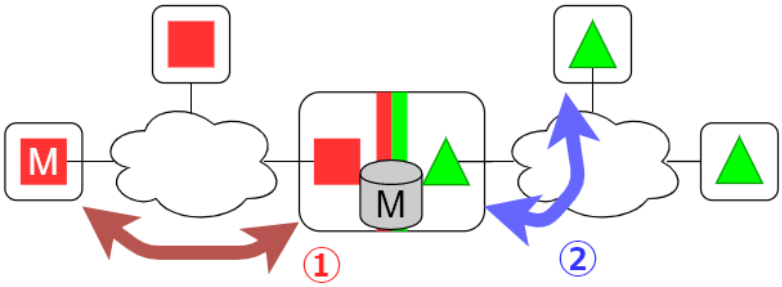


図 6　パターン 2 の通信手順

パターン 3 は、中継器がトンネリング動作を行うパターンである。中継器は、到着したパケットに対してトンネリングプロトコルを適用して送る。 パターン 1-B との違いは、1-B はプロトコルが用意したアプリケーショントンネリングを用いるのに対して、パターン 3 は、GRE などの汎用的なトンネリングプロトコルを用いることである。赤プロトコルネットワークと緑プロトコルネットワークのどちらをトンネリングするかによって、接続形態 A、B に分類される。

3-A は、緑プロトコルネットワーク内部にトンネリングを通す接続形態である。マネージャーは赤プロトコルだけで全ネットワークの管理が可能であるが、トンネリング先の緑ネットワークの機器に赤ネットワークのエージェントが必要となる。トンネリング先のネットワークの機器の数が少ない場合に有効であ

る接続形態である。

3-B は、赤プロトコルネットワークに対してトンネリングを行う接続形態である。マネージャー機能を担う機器が、赤・緑プロトコル両方のマネージャーをもつのが特徴である。このことから、トンネリング先に機器が多い場合に有効となる接続形態である。

パターン 3 の通信の流れを図 7 に示す。トンネリングを通してエージェントとマネージャーが直接通信するのが特徴である。

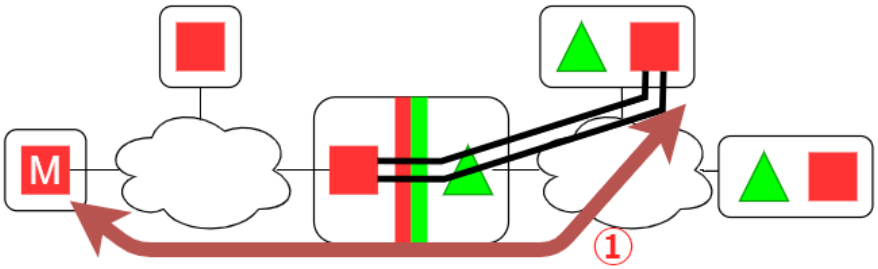


図 7　パターン 3 の通信手順

4.3 フレーム伝送方式の検討

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と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 ネットワーク側エージェントの間において 1 対 1 で通信する場合が挙げられる。

これらの例から、非 Ethernet・非 IP ネットワークにおける HTIP のデータ電押す方式として、次の 2 方式が考えられる。

- ネイティブフレーム方式
- みなしシリアル方式

それぞれの方式について説明する。

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。図 8 図 8 にネイティブフレーム形式の概要を示す。

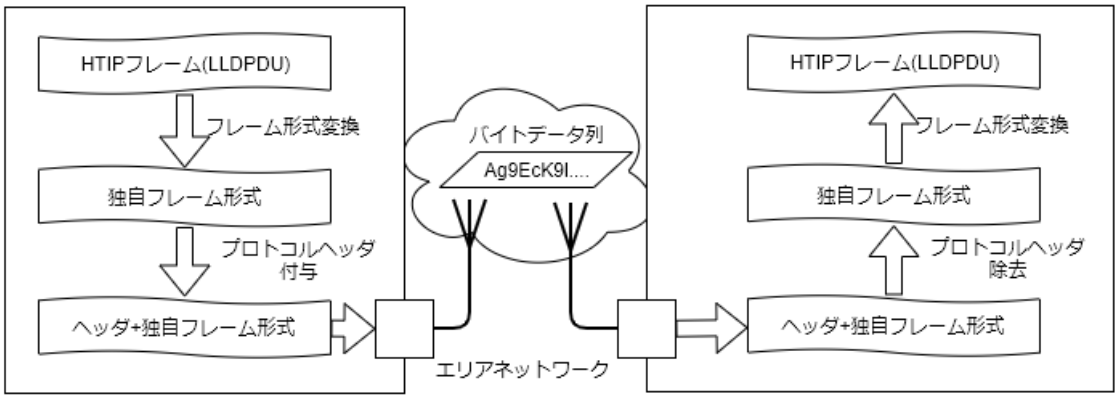


図 8　ネイティブフレーム伝送方式の概要

みなしシリアル方式は、エリアネットワーク技術の上位層で提供されるシリアル通信のシミュレート機能を用いる、もしくは RS232 や RS485 のようなレガシーなシリアル通信を用いた通信方式である。みなしシリアル方式の概要を図 9 に、レガシーシリアル規格を用いる場合の概要を図 10 に示す。

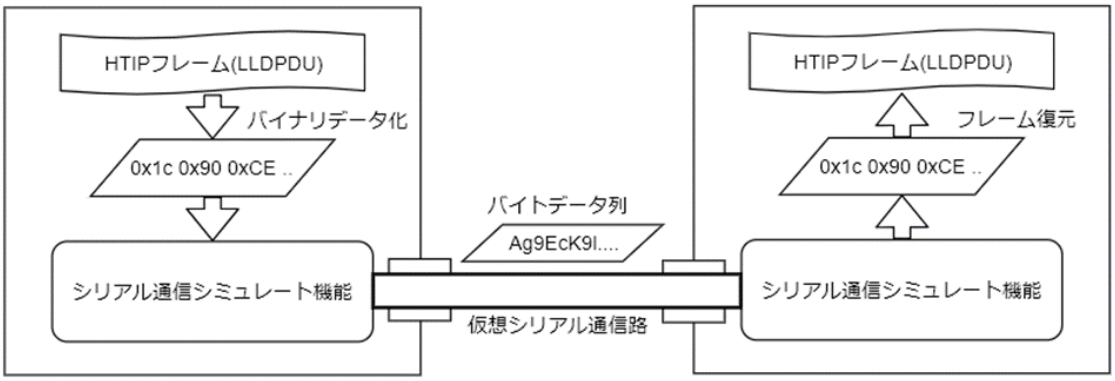


図 9　仮想シリアルを通信路とするみなしシリアル伝送方式の概要

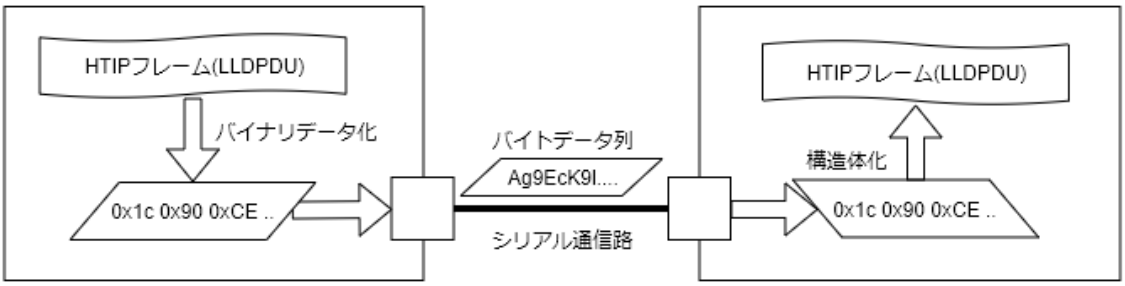


図 10　レガシーシリアル規格を通信路とするみなしシリアル伝送方式

4.4 ネイティブフレームによる伝送方式の検討

4.4.1 参考規格の概要

ネイティブフレーム方式伝送における、フレームフォーマットの制定を行った。有力とみなされるフレームフォーマットを考慮し、汎用的なフォーマットの定義を試みた。当研究では、内閣府による PRISM（民間研究開発投資拡大プログラム）で採用された IoT 共通基盤の確立・実証の研究開発課題の一部であり、北陸先端科学技術大学院大学、富士通株式会社、SMK 株式会社によって提案された非 Ethernet・非 IP 向けネットワーク向け HTIP フレームフォーマット(以下、PRISM 規格)をベースとした。

PRISM 規格では、uplink パケットと downlink パケットが定義されている。

Uplink パケットは、デバイスから GW に対して送信されるパケットであり、センサ・HTIP の情報を格納する。Uplink パケットは、ペイロード長に制限がある無線規格に対応するため、uplink11 と uplink24 の 2 つのフォーマットが定義されている。Uplink11 は、ペイロード長が 11byte、uplink24 はペイロード長が 24byte となっている。

Downlink パケットは、GW からデバイスに対して送信されるパケットであり、制御情報を格納する。Downlink パケットは、ペイロード長が 24byte のフォーマットのみが定義される。

4.4.2 Uplink24 フォーマットの概要

Uplink24 は、BLE・LoRA・LoRaWAN など十分なペイロード長をもつ通信規格を対象とした uplink パケットのフォーマットである。Uplink24 フォーマ

ットの例として、BLE・LoRA・LoRAWAN におけるパケットフォーマットを図 11、図 12、図 13 に示す。PRISM 独自部分に、当規格で定義するセンサ・HTIP 情報が格納される。PRISM 独自部分を除くヘッダについては、それぞれのプロトコル規格に従う。

■ BLE

図 11 BLE における uplink24 フォーマット

■ LoRa

図 12 LoRa における uplink24 フォーマット

■ LoRaWAN

図 13 LoRaWAN における Uplink24 フォーマット

4.4.3 Uplink24 の PRISM 独自ペイロードフォーマット

Uplink24 パケットの PRISM 独自ペイロードには、センサ情報と HTIP 情報のうち 1 つを格納できる。それぞれのフォーマットについて述べる

● センサ情報

センサ情報は、デバイスに存在するセンサから取得した情報(気温・気圧・圧力・ポテンショメータ出力等)を格納する。フォーマットを図 14 に示す。

センサ	1		2	3	4	5	6	7	8	9	10
byte	xx	yyyyyy	2		1	0.5	2		1	0.5	2
(子)センサ	00：センサ(Read only) 10:センサ(Rewritable) ※1	コマンド長(※2)	カウンター値（※3）(通信エラー率に使用)		Type	Inst.	Data1		Type	Inst.	Data2

センサ	11	12	13	14	15	16	17	18	19	20	21	22	23	24
byte	1	0.5	2		1	0.5	2		1	0.5	2		1	0.5	2	
(子)センサ	Type	Inst.	Data3		Type	Inst.	Data4		Type	Inst.	Data5		Type	Inst.	Data6	

図 14 Uplink24 におけるセンサ情報フォーマット

フォーマットの各フィールドについて説明する。

- パケットの種類（第 1 オクテット先:0,1bit）

パケットの種類を表す。対象のセンサが読み込み専用である場合は、0b00 が格納される。対象のセンサが再書き込み可能であれば 0b10 が格納される。

- コマンド長(第 1 オクテット 2-7bit)

パケット長から 1 を引いた数を byte 単位で格納する。例として、センサ情報が 6 つの uplink24 パケットならば 23 が格納される。

- カウンタ値(第 2・3 オクテット)

通信時のパケットエラー率を GW で算出するためのカウンタ値を格納する。カウンタ値は、機器再起動時には 0 となり、これ以降、パケット送信ごと 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知できる。カウンタ値が 2byte の最大値である 65535 でカウントオーバーフローした場合、エラーとしこれ以上カウントさせない。

- センサデータ(第 4 オクテット-第 24 オクテット)

デバイスのセンサデータを格納する。1つのセンサデータは、Type(8bit)、Instance(4bit)と Data(2bytes)から構成され、全体の長さは 2.5bytes となる。ひとつの uplink24 パケットには、最大 6 つのセンサ情報を格納できる。センサが奇数個の場合、末尾の 0.5byte は 0 埋めされる。

センサ情報のそれぞれの要素について説明する。

- Type(8bit)

対象としたセンサの種類を 8bit の値で格納する。センサの種類と Type に格納される値の対応表を表に示す。

表 7 センサ種類のタイプ

センサ種類	対応するセンサ
00000000	センサなし
00000001	温度
00000010	湿度
00000011	照度(日射)
00000100	気圧
00000101	振動
00000110	Co2
00000111	土壌水分
00001000	風速
00001001	UV
00001010	IR
00001011	Red 波長
00001100	Green 波長
00001110	Blue 波長
00001111>	未定義

- Instance

Instance(インスタンス番号)は、ひとつのデバイス上に同一種類のセンサが複数存在するときに、それらの区別をするための番号である。4bit の 0b000~0b1111 で格納する。

- Data
  センサの出力値そのものを格納する。2bytes で格納する。

- HTIP 情報

デバイスもしくは NW 機器内部に存在する HTIP Agent が送り出す情報を格納する。HTIP の機器情報の一覧を表 8 に、それに対応するパケットフォーマットの詳細を図 15 に示す。

表 8 対応する HTIP 情報の一覧

機器情報			パケット構成対応
(a)	区分	端末区分情報リスト	5byte 目
(b)	メーカーコード	OUI コード	6-11byte 目
(c)	機種名	機器のブランド名やシリーズ名	ー
(d)	型番	機器の型番	12-24byte 目
(e)	チャネル使用情報	端末では取得困難	ー
(f)	電波強度	GW で RSSI を取得する	ー
(g)	通信エラー率	端末から送信パケットに No.を付与し、インクリメントされた数値と受信可の数を GW で取得し算出してもらう	2-3byte 目
(h)	ステータス情報	障害検知の補助になるような内部の充電やモードを番号でテーブル化し、ステータスとして入れる	4byte 目
(j)	応答時間	受信から送信までの時間 ：アドバタイズ送信時は無し	ー
(k)	関連デバイス数	GW が判断	ー
(l)	アクティブノー	GW が判断	ー

	ド数		
(m)	無線品質	端末では取得困難	ー
(n)	再送数	アドバタイズ送信時は無し	ー
(o)	CPU 使用率	省電力デバイスでは殆どゼロ	ー
(p)	メモリ使用率	メモリ使用率は一定(変動無し)	ー
(q)	HDD 使用率	HDD なし	ー
(r)	バッテリ残量	蓄電素子の電位から残量に数値変換	4byte 目
(s), (t)	通信品質関連情報	サンプリング間隔、送信間隔	2,3byte 目

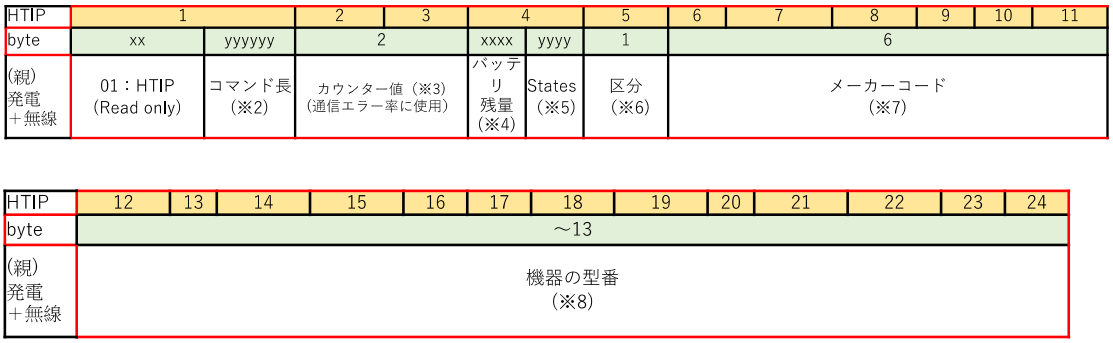


図 15 uplink24 における HTIP 情報フォーマット

フォーマットの各フィールドについて説明する。

- パケットの種類

パケットの種類を表す。HTIP 情報では、0b01 を格納する。

- 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、最大値は 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 となる。

- カウンタ値(第 2・3 オクテット)

通信時のパケットエラー率を GW で算出するためのカウンタ値を格納する。カウンタ値は、機器再起動時には 0 となり、これ以降、パケット送信ごと 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知できる。カウンタ値が 2byte の最大値の 65535 でカウントオーバーフローした場合、エラーとしこれ以上カウントさせない。

- バッテリ残量(第 4 オクテット 0-3bit)

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。バッテリ残量のとり得る値を表 9 に示す。

表 9 バッテリ残量の値

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)

- States(第 4 オクテット 4-7bit)

デバイスの動作に関わる情報を 4bit で格納する。Status の定義を表 10 に示す。

表 10 States の値

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 通信ポート有

- 区分(第 5 オクテット)

デバイスの種類を 1byte で格納する。デバイスの種類は、HTIP の機器情報における区分に対応する 1byte の値が格納される。区分のとり得る値の表を示す。

- メーカーコード（第 6 オクテット-第 11 オクテット）

デバイスのメーカーコードを 6 文字の ASCII で記述する。メーカーコードは、IEEE の OUI を使用する。使用できる文字は以下である。

[a-fA-F0-9]

- 機種の型番(第 12 オクテット-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。利用できる文字は以下である。

#x20 | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]

4.4.4 Uplink11 フォーマットの概要

Uplink11 フォーマットは、Sigfox や特定条件下の LoRaWAN といった、十分なパケット長が取れない通信規格向けの Uplink パケットのフォーマットである。フォーマット B の例として、LoRAWAN におけるフォーマットを図 16 に示す。PRISM 独自部分が当規格で定める部分となり、それ以外のヘッダについてはそれぞれのプロトコル規格に従う。

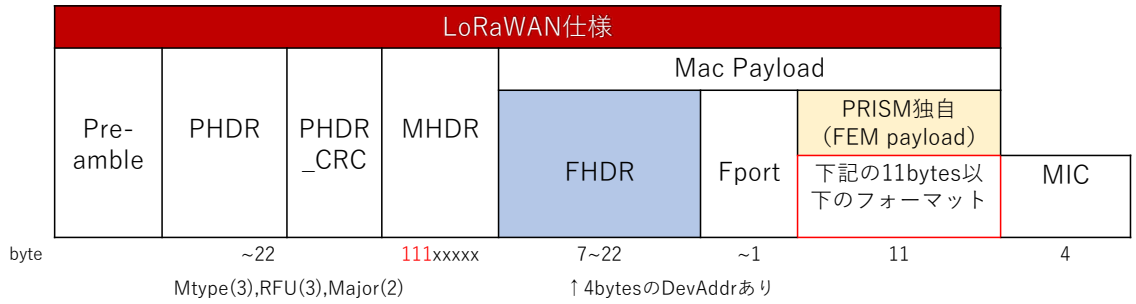


図 16 LoRaWAN における Uplink11 フォーマット

4.4.5 Uplink11 フォーマットにおけるペイロードフォーマット

PRISM 独自ペイロードには、センサ情報と HTIP 情報のうち 1 つを格納できる。Uplink11 マットと比べてペイロード長が短いため、ペイロードは 2 つに分

割されてそれぞれ別のパケットで送信される。フォーマットを図 17 に示す。それぞれのフォーマットについて述べる。

- センサ情報

センサ情報は、デバイスに存在するセンサから取得した情報(気温・気圧・圧力・ポテンショメータ出力等)を格納する。フォーマットを図に示す。Uplink11 フォーマットにおいては、ペイロードは分割され、2 つ以上のパケットで送信される。よって、格納するセンサデータ数を Uplink24 より多い 6 以上にできる。その場合、コマンド長もそれに従った値になる。それ以外の点は Uplink22 と同一である。

センサ1	1		2	3	4	5	6	7	8	9	10
byte	xx	yyyyyy	2		1	0.5	2		1	0.5	2
(子)センサ	00：センサ(Read only) 10:センサ(Rewritable) ※1	コマンド長(※2)	カウンター値（※3）(通信エラー率に使用)		Type	Inst.	Data1		Type	Inst.	Data2

センサ2	1		2	3	4	5	6	7	8	9	10
byte	xx	yyyyyy	2		1	0.5	2		1	0.5	2
(子)センサ	00：センサ(Read only) 10:センサ(Rewritable) ※1	コマンド長(※2)	カウンター値（※3）(通信エラー率に使用)		Type	Inst.	Data3		Type	Inst.	Data4

図 17 Uplink11 におけるセンサ情報フォーマット

- HTIP 情報

デバイスもしくは NW 機器内部に存在する HTIP Agent が送り出す情報を格納する。Uplink11 フォーマットにおける HTIP 情報のフォーマットを図 18 に示す。Uplink11 における HTIP 情報フォーマットは、分割されている点を除き Uplink22 のそれと同一である。

HTIP1	1		2	3	4		5	6	7	8	9	10	11
byte	xx	001101	2		xxxx	yyyy	1	6					
(親)発電＋無線	01：HTIP(Read only)	コマンド長(※2)	カウンター値（※3）(通信エラー率に使用)		バッテリ残量(※4)	States(※5)	区分(※6)	メーカーコード(※7)					

HTIP2	1		2	3	4	5	6	7	8	9	10	11
byte	xx	yyyyyy	2		8							
(親)発電＋無線	01：HTIP(Read only)	コマンド長(※2)	カウンター値（※3）(通信エラー率に使用)		機器の型番(※8)							

図 18 uplink11 における HTIP 情報フォーマット

4.4.6 Downlink フォーマットの概要

Downlink フォーマットは、Uplink11 と Uplink24 の両方が対象とする通信規格に対応している。Downlink フォーマットの例として、BLE・LoRA・LoRAWAN におけるパケットフォーマットを図 19、図 20、図 21 に示す。PRISM 独自部分に、当規格で定義するセンサ・HTIP 情報が格納される。PRISM 独自部分を除くヘッダについては、それぞれのプロトコル規格に従う。

■ BLE

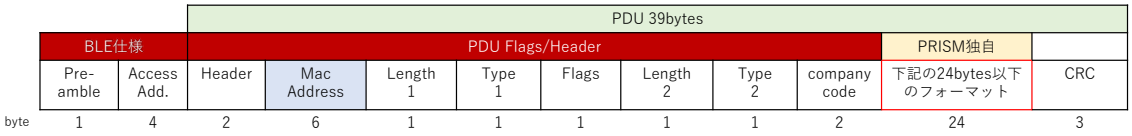


図 19 BLE における Downlink フォーマット

■ LoRa

図 20 LoRa における Downlink フォーマット

■ LoRaWAN

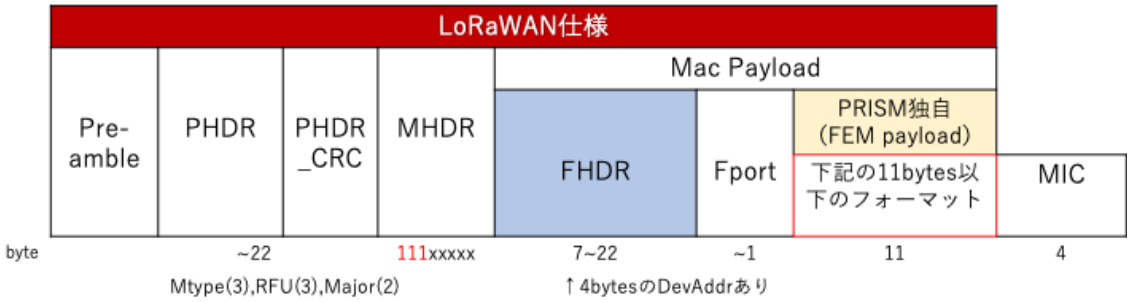


図 21 LoRa における Downlink フォーマット

- Downlink の PRISM 独自ペイロードフォーマット

Downlink パケットの PRISM 独自ペイロードには、制御情報のうち 1 つを格納できる。制御情報のフォーマットを図 22 に示す。データの詳細を述べる。

- 制御情報

制御情報は、GW からデバイスに対して、設定したいパラメータ値を格納する。ペイロード長が 12byte 固定となっており、12byte 以上のペイロードを扱えるプロトコルを使用していても、12byte 以降は使用できない。

制御	1		2	3	4	5	6	7	8	9	10	11
byte	xx	yyyyyy	2		2		2		2		2	
	11：Control	コマンド長 (※1)	無線 送信間隔 (※2)		無線 送信パワー (※3)		無線 送信チャネル (※4)		HTIP定義の 端末品質 サンプリング間隔 (※5)		HTIP定義の 端末品質 送信間隔 (※6)	

図 22 Downlink における制御情報フォーマット

フォーマットの各フィールドについて説明する。

- パケットの種類(第 1 オクテット 0,1bit)

パケットの種類を 2bit で格納する。制御情報では、0b11 が格納される。

- コマンド長(第 1 オクテット 2-7bit)

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、最大値は 23 となる。例として、制御情報では、コマンド長は 10 となる。

- 無線送信間隔(第 2、3 オクテット)

デバイスのパケット送信間隔の設定値を 2bytes で格納する。単位はミリ秒である。

- 無線送信パワー(第 4、5 オクテット)

デバイスの無線送信電力の設定値を 2bytes で格納する。単位は dBm である。

- 無線送信チャンネル(第 6 、 7 オクテット)

デバイスの無線送信チャンネルの番号の設定値を 2bytes で格納する。

- HTIP 定義の端末品質サンプリング間隔(第 8、9 オクテット)

デバイスが、HTIP の機器情報のうち、(h)ステータス情報と(r)バッテリ残量の再サンプリングを行う間隔の設定値を格納する。単位はミリ秒である。

- HTIP 定義の端末品質送信間隔(第 10、11 オクテット)

デバイスが、HTIP の機器情報のうち、(h)ステータス情報と(r)バッテリを GW に向けて送信する間隔の設定値を格納する。単位はミリ秒である。

4.5 提案する非 Ethernet・非 IP 向け HTIP 規格

前章の非 Ethernet・非 IP 向け規格は、IoT 向けエリアネットワーク向けた規格として制定されているが、現在の規格では以下のような問題がある。

1. HTIP 情報”機器名”の未定義
2. 規格拡張時のフレーム識別問題

これらの問題について説明する。

PRSIM 規格では、機器情報の機種名がフレームフォーマットで定義されていない。JJ-300.00 においては、機種名は実装必須となっている。

PRSIM 規格の拡張性の問題が存在する。規格のフレームフォーマットを拡張する場合、エリアネットワークプロトコルの許容するペイロード長の制限が問題となる。そのため、Uplink24 であればフレームの分割が必要となり、Uplink11 であれば分割フレーム数が増える。このとき、パケットのフラグメンテーションを起こさないために、HTIP 側のフォーマットフレームとして分割する。ここで、それぞれの分割されたフレームが持つデータの素性を識別できない問題が発生する。これは、分割されたフレームのヘッダ部分が共通しており、データ種類を表す情報を持たないため発生する。

これらの問題を解決した規格（以下、提案規格）を制定した。提案規格の改良点を説明する。

1. 機器情報である機器名をフレームに追加した。Uplink24 フォーマットでは、フォーマットの後端に機器名を追加し、これを Uplink37 フォーマットとした。Uplink11 では、フレーム数を増やし、新しいフレームに機器名を追加した。

2. フレーム分割時にデータの区別をつけるため、uplink11 フォーマットに 2bit のフレームインデックスを追加した。PRSIM 規格ではコマンド長に 6bit を予約していたが、uplink11 フレームでは最大でも 10 進数で 11 となり、二進数表現では上位 2bit が使われることがない。提案規格では、この 2bit をフレームインデックス番号と定め、0b00 からフレームの順番を割り振る。これにより、追加されたフレームの区別が可能となった。

提案規格のフレームフォーマットを図 23、図 24 に示す。

図 23　提案規格の Uplink37 フォーマット

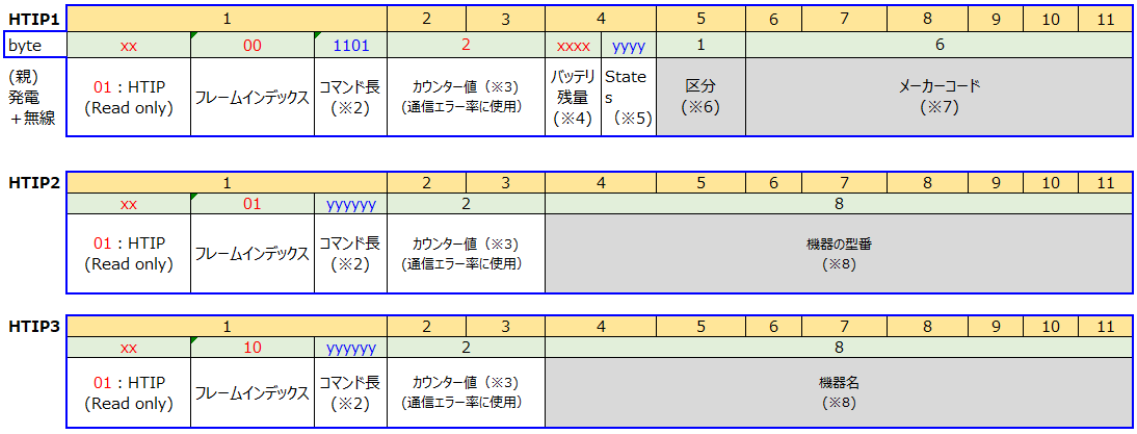


図 24　提案規格の Uplink11 フォーマット

4.6 みなしシリアルによる伝送方式の検討

4.6.1 RS485 による実装

HTIP エンド機器で動作する HTIP エージェントの実装を行った。実装には岡田による C 言語 HTIP Agent 実装である lwhtip をベースとした。

エージェントが生成するキャラクタデータは、HTIP フレームをそのままキャラクタデータしたものである。このデータには、フレームの区切りとなる情報が存在しない。そのため、受信プログラムにおいて連続して送信されるフレームデータから 1 つ分を取り出す方法が存在しない。この問題の解決のために、バイトスタッフィングを使った。

バイトスタッフィングとは、ある特定のバイトをフレームの区切りバイトとし、その区切りバイトがデータ本体中に出現した場合、その前にエスケープバイトと呼ばれるバイトを挿入することで区別する方法である。当実装で使用した COBS(Consistent Overhead Byte Stuffing)の概要を図 25 に示す。

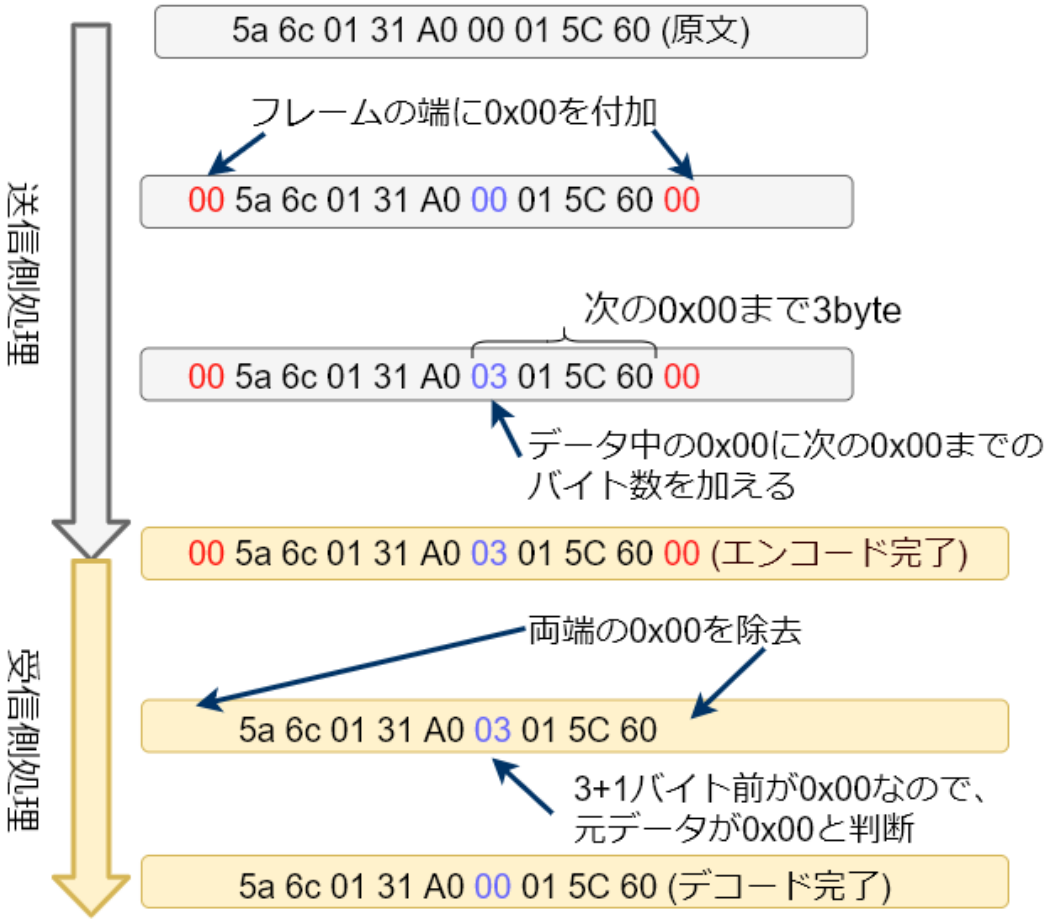


図 25 COBS の概要

4.6.2 HTIP Manager の実装

当研究で実装した HTIP マネージャーの実装環境について述べる。

HTIP マネージャーの実装は岡田が開発したマネージャーの実装である node-htip のライブラリを使用した。開発・動作環境は、Intel NUC NUC6i3SYK、debian9 である。

みなしシリアル方式に対応した非 Ethenet・非 IP 向け HTIP マネージャーの実装方法として、以下の二つの方法が考えられる。

1.既存の HTIP Manager を改良し、みなしシリアル方式に対応させる方法

2.変換機を通常 HTIP ネットワークと非 EthernetHTIP ネットワークの間にプロトコル変換器を置き、みなしシリアル方式の HTIP フレームを通常の HTIP フレームに変換する方式

当研究では 2 の手法を採用した。この手法ではマネージャーのプログラムに手を加えず、みなしシリアル方式に対応できる。このみなしシリアル方式によるフレームを Ethernet フレームに変換するプログラムを、Serial/Ethernet 変換プログラムと呼ぶ。

Serial/Ethernet 変換プログラムについて述べる。当プログラムは,、一つの PC 上で HTIP マネージャーと同時にマルチタスク動作を行う。この 2 つのプログラムの仮想ネットワークインターフェース(tap interface)と PC の Ethernet インターフェースは Linux Virtual Bridge で接続されている。Linux Virtual Bridge は Linux Karnel が提供する仮想スイッチ機能である。このシステムでは、変換プログラムがみなしシリアル方式のフレームを変換することで、マネージャーはそれを Ethernet の HTIP と同等に扱うことができる。Manager の概要を図 26 に示す。

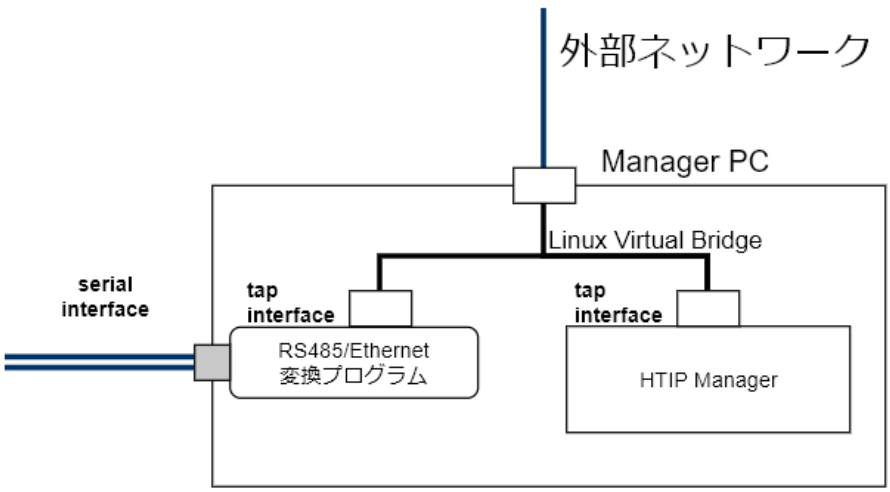


図 26 Serial/Ethernet 変換機の概念

4.6.3 みなしシリアル方式 HTIP の動作実験

前述のマネージャーとエージェント実装を使いみなしシリアル方式をもちい

た非 Ethenet・非 IP 向け HTIP の動作確認を行った。接続構成を図に示す。3 台の実験用小型 PC を用意し、それぞれにマネージャー、Serial/Ethernet 変換機と L2 エージェントを動かす。エージェントが生成するフレームをマネージャーが受け取り、管理情報を取得することでネットワークトポロジを生成する。このトポロジ情報の取得を確認する。

実験環境を表 11 に示す。

表 11 実験環境

PC	Intel NUC DE3815TYKE, Debian 9(L2 Agent, Serial/Ether 変換プログラム) Intel NUC NUC6i3SYK Windows 10 (HTIP Manager)
RS485 ケーブル	FTDI Chip USB-RS485-WE-1800-BT

実用的な HTIP Manager の実装では、前述の通り 1 つの PC の内部で Serial/Ethernet 変換プログラムと HTIP Manager がマルチタスクとして動作するべきであるが、必要となる仮想ネットワークインターフェース tap の設定に失敗した。そのため当実験では Serial/Ethernet 変換プログラムと HTIP Manager を別の PC で実行し、その間を Ethernet で接続する形をとる。実験の実態接続を図 27 に示す。

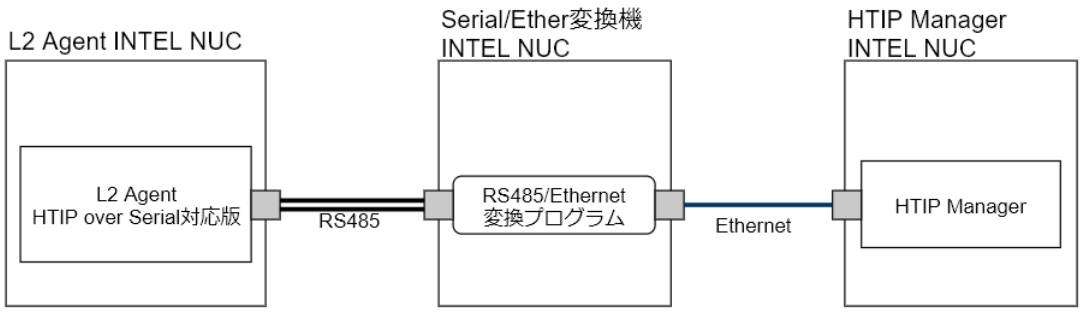


図 27 実験における実態接続図

動作確認の結果、HTIP Manager で L2 Agent が表示されるが、それらのトポロジ関係が表示されないことが判明した。これを図に示す。左側赤色正方形で表現される Manager と右側黄緑色円形で表現される HTIP Manager と L2 Agent

の間に接続を表す線が存在しない。Serial/Ethernet 変換機に関しても接続関係が現れない。Manager の表示を図 28 に示す。

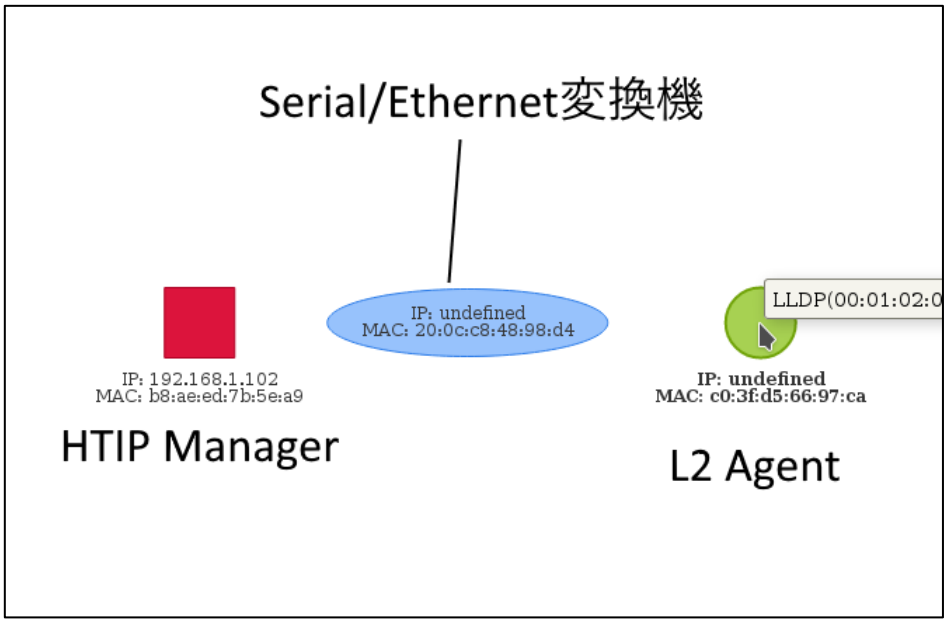


図 28 Manager のトポロジ表示

4.6.4 HTIP over Serial の HTIP-NW 機器化

前章の実験で Manager がトポロジを表示しない原因として、正しく動作する HTIP-NW 機器が存在しないことがあげられる。HTIP Manager がトポロジを把握するためには、HTIP-NW が生成するリンク情報が必要となる。このリンク情報は、スイッチのフォワーディングテーブルを元に作られる。

解決方法として、Serial/Ethernet 変換機に HTIP リンク情報を送信する機能を追加し、Serial/Ethernet 変換機を HTIP-NW 機器とすることが有効である。

当実装では、node-htip の実装をベースに任意 HTIP リンク情報をもつ LLDPDU を生成できる関数を定義し、それを HTIP Manager に追加した。Manager のトポロジ画面を図 29 に示す。みなしシリアル方式伝送対応の L2 エージェント(HTIP エンド機器)とマネージャーの間に HTIP-NW 機器と認識されている Serial/Ethernet 変換機が挟まれることでトポロジが正しく表示されている。

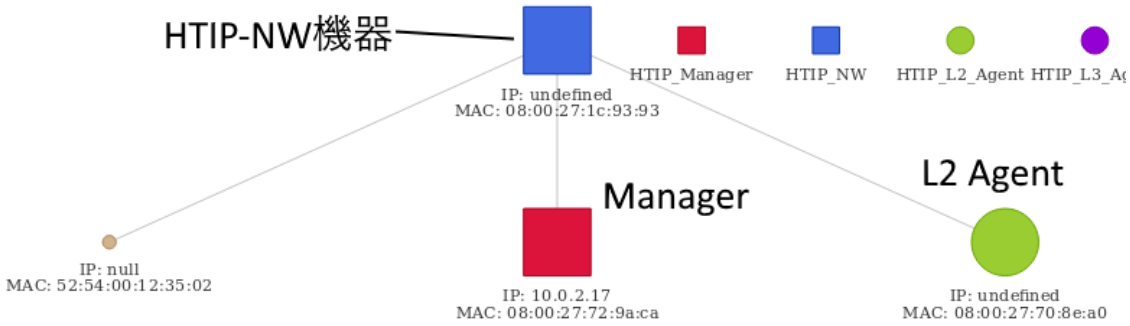


図 29 Manager の正確なトポロジ表示

4.6.5 通信速度・帯域の考察

みなしシリアル伝送方式によるフレームの占有帯域と同時に存在しえるエージェント数について述べる。1 フレームの長さは 100byte 前後となる。シリアル通信のビットレートは設定によって変更できるが、ここでは RS232/485 で一般的な 9600bps と仮定する。参考として、Bluetooth Classic において RS232 のエミュレーションを行う SPP においては、128kbps まで対応することが定められている。 [3]フレームに識別アドレスなどを表すヘッダはなく、LLDPDU の内容をそのまま流すものとする。

1 フレームの送信を完了する時間は

$$\frac{800[\mathrm{bit}]}{9600[bit\ per\ sec]} = 0.8\dot{3}[s]$$

となる。よって通信の衝突を考慮しない場合、最大存在できる Agent 数は、約 20 となる。

HTIP フレームが全帯域を占める割合は、LLDPDU の送信間隔を 20[s]とすると、

$$\frac{0.83[s]}{20[s]} = 0.00416\dot{} \approx 0.42\%$$

となり、他の通信のための帯域は十分確保されているといえる。

RS232・485 などのレガシーなシリアル通信規格において、一般的に使われるビットレートで最も低速なのは 110bps であるが、この場合の占有帯域は、

$$\frac{800[\mathrm{bit}]}{110[bit\ per\ sec]} = 7.27[s]$$

$$\frac{7.27[s]}{20[s]} = 0.364 \approx 36.4\%$$

となり、HTIP フレームの比率がかなり大きくなる。センサネットワークなどのデータ量と頻度が小さいケースであれば十分有用であると考えられる。ビットレートを下げて通信を行うことで、消費電力の削減も見込められる。

4.6.6 対応するプロトコル・トポロジの考察

みなしシリアル方式による対応表を表 12 に示す。

表 12 みなしシリアル伝送方式の HTIP 対応するシリアル通信規格

規格名	みなしシリアル方式 HTIP 適用可・不可	運用可能なトポロジ
有線規格		
RS232	可	1 対 1
RS422	可	1 対 1、数珠繋ぎ、スター
RS485	可	1 対 1、数珠繋ぎ、スター
SPI	可	1 対 1、数珠繋ぎ、スター
I2C	可	1 対 1、数珠繋ぎ、スター
1-wire	可	1 対 1、数珠繋ぎ、スター
無線規格		
BLE	可(SPP プロファイル)	1 対 1,スター
Zigbee	可(アプリケーション層で シリアルシミュレーション)	1 対 1,スター
LoWPAN 規格群	可(アプリケーション層で シリアルシミュレーション)	1 対 1,スター

みなしシリアル伝送方式の HITP は、ホームネットワークや IoT 向けエリアネットワーク規格に対応すると言える。ただし、現在の HTIP フレームをそのままシリアルに流す方式の場合、仮想シリアル通信路が 1vs 1 の通信路を提供する場合のみ対応となる。RS422・485 といったマルチノードトポロジに対応するためには、あるフレームがどのノードからどのノードへ向かうかを表す識別子

が必要となる。

4.6.7 フレームの識別方法についての考察

前述のマルチノードトポロジシリアル通信路の対応の手段として、フレームに識別子を付与する手法について述べる。考えられる手法は以下の通りである。

1. フレームの前後に識別子を格納する独自形式ヘッダを収納する
2. Ethernet フレームをそのまま伝送し、MAC アドレスを識別子とする

非 Ethernet・非 IP ネットワークにおける運用を考慮すると、1 の方式が望ましいと考えられる。なぜなら、非 Ethernet・非 IP ネットワーク上に存在する HTIP エンド機器に、Ethernet のアドレスである MAC アドレスを持たせるのは不自然であると考えられるからである。

また別の問題として、それらの識別子のユニークをどう保証するかの問題がある。1 の方式であれば、UUID や製造識別番号を使うことでユニークなアドレスを生成できる。2 の方式であれば、Serial/Ethernet 変換機が MAC アドレスプールを持ち、非 Ethernet・非 IP ネットワークに存在する HTIP エージェントに割り当てる方法が考えられる。そのためには、エージェントが変換機のアドレスプールに要求を送り、変換機がアドレスを割り当てる DHCP のようなシステムが必要となる。

以上の考察から、非 Ethernet・非 IP ネットワークにおいて、みなしシリアル方式による伝送方式を用いる HTIP で管理される HTIP END 機器は独自の識別子を持ち、それをヘッダとしてフレームに付加する手法が望ましいと考えられる。

第5章 既存 NW 機器の HTIP 対応化手法

HTIP を低コストで実現する目標のため、既存ネットワークを HTIP に対応させる手法について検討する。ここでは、ネットワークを構成する要素であるエンド機器、マルチポートノードとトポロジ変換器に対して、HTIP アダプタもしくは HTIP Agent を接続させる手法について述べる。

5.1 ネットワークの HTIP 対応の概要

既存ネットワークを HTIP 対応するにあたり、ネットワーク上のユーザー端末(PC やネットワークセンサ)に加えて、L2 スイッチやルーターなどのネットワーク機器の HTIP 対応も必要となる。ここでは、それらの機器のうち、パケットのフォワーディングを行うマルチポート機器と、他のトポロジとの接続を行うブリッジの HTIP 化について述べる。

5.2 マルチポート機器の HTIP 対応化

マルチポート機器の実例として、Managed L2 Switch の HTIP 化について述べる。Managed L2 Switch は、管理機能をもつ L2 スイッチのことを指す。管理機能とは、シリアルもしくは SSH/Telnet による接続とコマンド制御が可能なコンソール設定機能のことを指す。この Managed L2 Switch は、HTIP Agent を持たないものとする。

Managed L2 Switch の HTIP 化の手法として、HTIP Agent を L2 Switch と接続する方法が考えられる。HTIP Agent は、L2 Switch の管理機能にアクセスし、HTIP フレーム(LLDPDU)を生成する。HTIP Agent と管理機能の間の接続方法は、次の 2 種類が考えられる。

(a)Managed L2 Switch に HTIP Agent を Ethernet 接続する場合

HTIP Agent は、Managed L2 スイッチの管理機能に対して、Ethernet 経由

の http もしくは ssh で接続する。HTIP Agent は、L2 Switch の機器情報と FDB などの接続構成情報を取得し HTIP フレームを生成、これを HTIP Manager に送信する。HTIP フレームは、L2 Switch 内部の仮想ブリッジを経由して伝達される。

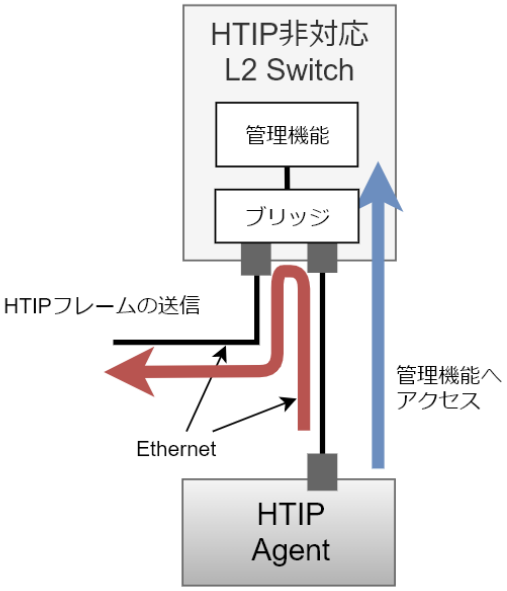


図 30　パターン(a)L2 スイッチに Agent を Ethernet で接続する場合

(b)Managed L2 Switch に HTIP Agent を Serial 接続する場合

HTIP Agent は、L2 Switch の管理機能にシリアルで接続し、L2 Switch の機器情報と FDB などの接続構成情報を取得し HTIP フレームを生成、これを HTIP Manager に送信する。

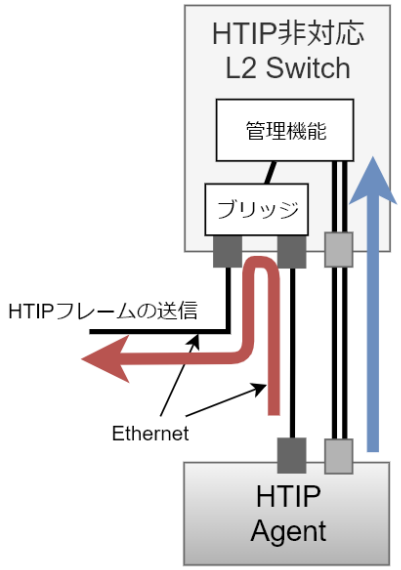


図 31 パターン(b.2) L2 スイッチに Agent をシリアルで接続する場合

5.3 ブリッジの HTIP 対応化

ブリッジの実例として、PLC モデムの HTIP 対応化について述べる。PLC モデムは、PLC と Ethernet それぞれに対応するポートをもち、そこに到着したパケットに対して、PLC—Ethernet 間のプロトコルを行う機器である。ここでは、電源線のコネクタを 1 つ持ち、Ethernet のコネクタを 1 つ持つものとする。PLC モデムの内部では、Ethernet と PLC の変換を行う GW と、SSH/Telnet もしくはシリアルで接続・コマンド制御が可能な管理機能が存在し、これらは仮想ブリッジで接続されているものとする。

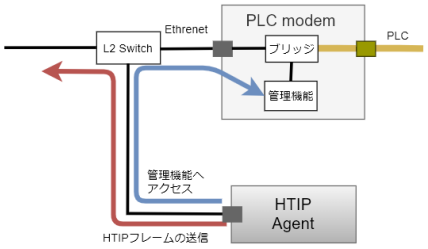
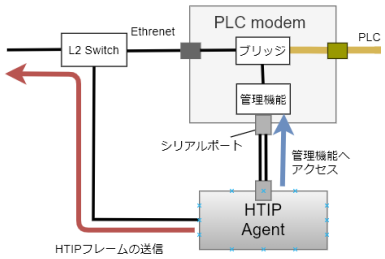
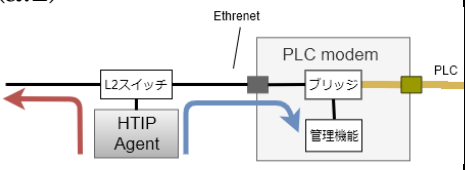
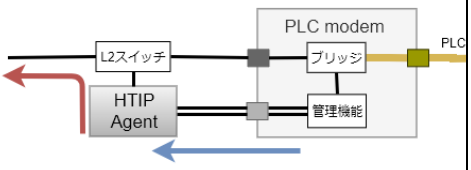
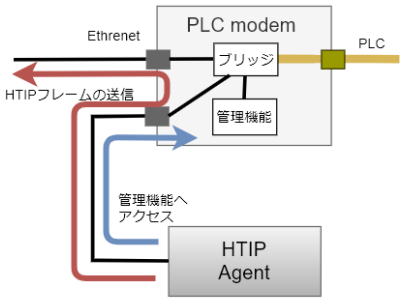
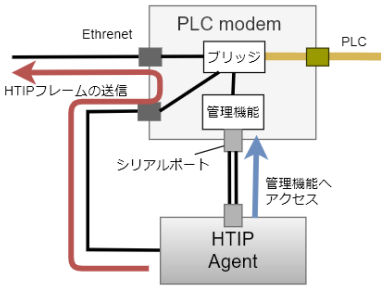
PLC モデムの HTIP 対応化の手法として、マルチポートノードと同じく HTIP Agent を接続する方法を用いる。ただし、PLC モデムはポートを 2 つしか持たないため、HTIP Agent を接続する別の方法が必要となる。その方法として、L2 スイッチを接続する方法が挙げられる。その L2 スイッチの配置としては、単純に PLC モデムに接続する方法の他に、HTIP Agent や PLC モデムに内蔵する方法も考えられる。

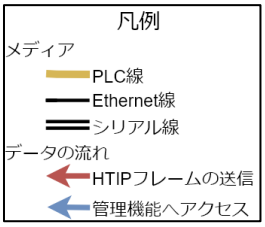
この考察から、PLC モデムの HTIP 対応化手法は、次の 2 つの指標で分類できる。

- HTIP Agent と PLC モデムの接続プロトコル
- HTIP Agent と PLC モデムを接続する L2 スイッチの位置

これらの指標で分類した PLC の接続手法を表 13 に示す。

表 13 HTIP Agent のプロトコルと接続方法で
分類した PLC モデムの HTIP 対応化手法

		接続プロトコル	
		Ethernet	シリアル
L2 Switch 位置	通常	(a.1) 	(b.1) 
	HTIP Agent 内部	(a.2) 	(b.2) 
	PLC モデム内部	(a.3) 	(b.3) 



5.3.1 (a) PLC モデムの管理機能にシリアルで接続する方式

HTIP Agent は Ethernet 経由で PLC モデムの管理機能にアクセスし、PLC モデムの機器情報を取得する。このとき、PLC モデムは、HTIP によるトポロジ上では、PLC モデムは HTIP NW 機器として認識されなくてはならない。そのため、PLC モデムは、その内部の GW がもつフォワーディングテーブル内容を接続構成情報として取得する。これら情報をもとに、HTIP Agent は HTIP フレームを生成し、Manager に向けて送信する。

この接続手法は、HTIP Agent と PLC モデムの接続方法によって、(a.1)から(a.3)の三種類に分類される。

a.1 は、L2 スイッチを介して、PLC モデムと HTIP Agent を接続する手法である。この手法は、6.3(a)の Agent を Ethernet 接続した HTIP 対応 L2 スイッチを加えたものである。HTIP Agent は、L2 スイッチと PLC モデムの両方の HTIP フレームを行う。

a.2 は、HTIP-Agent 機器の内部にブリッジを内蔵する手法である。HTIP Agent を PLC モデムとネットワークの間に挿入する形となる。

a.3 は、PLC モデム内部にブリッジを内蔵する方式である。PLC は HTIP-Agent 専用の端子を持ち、そこに HTIP-Agent を直接接続する。

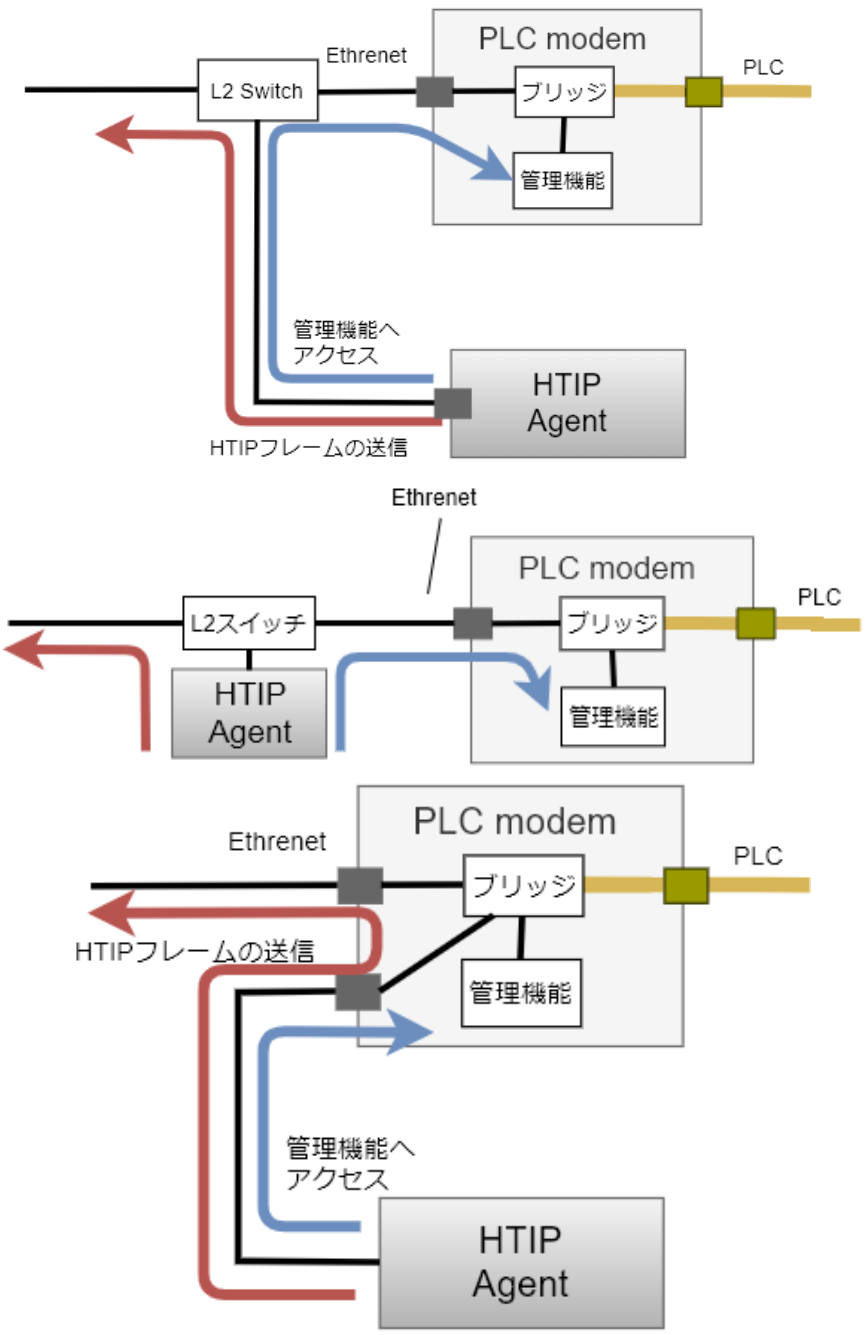


図 32　パターン a.PLC モデムの管理機能に Ethernet で接続する場合

5.3.2 (b). PLC モデムの管理機能にシリアルで接続する方式

HTIP Agent はシリアルで PLC モデムの管理機能にアクセスし、PLC モデムの機器情報を取得する。(a)と同様、PLC モデムは、その内部の GW がもつフォワーディングテーブル内容を接続構成情報として取得する。これら情報をもとに、HTIP Agent は HTIP フレームを生成し、Manager に向けて送信する。

6.4.1(a)と同様に、HTIP Agent と PLC モデムの接続方法によって、(b.1)から(b.3)の三種類に分類される。詳細は、6.4.1(a)と同様である。

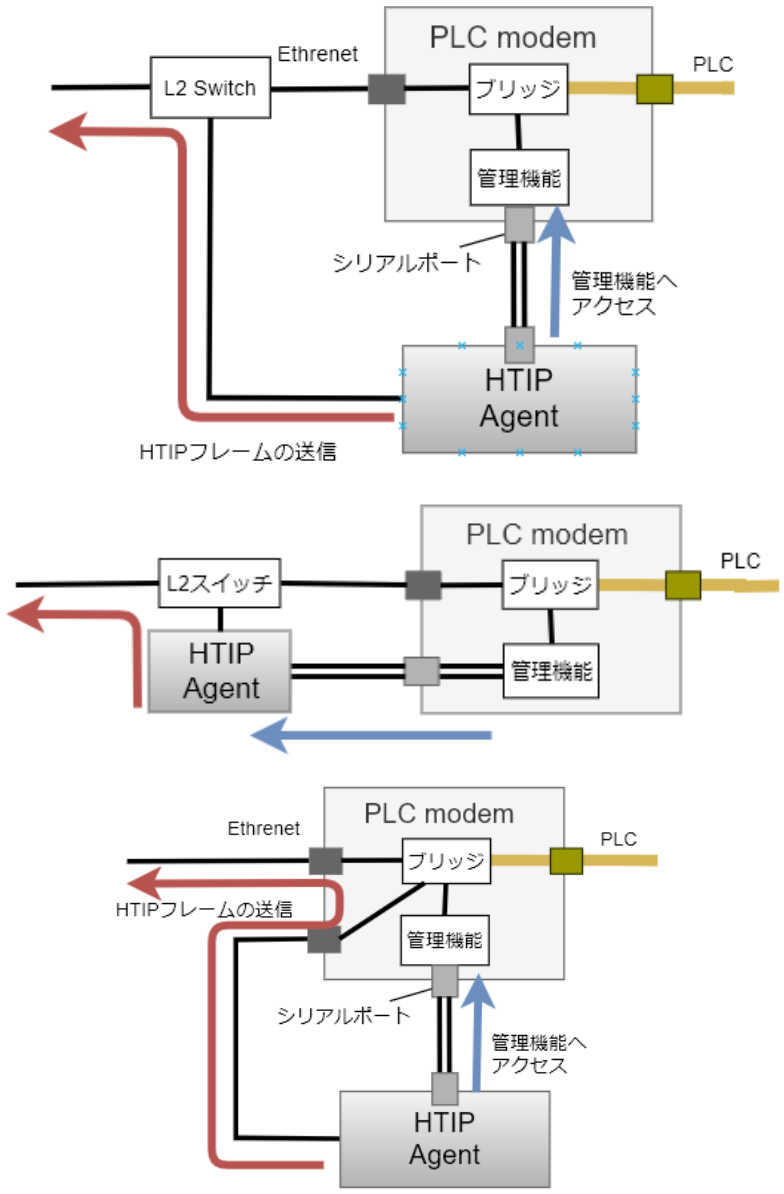


図 33　パターン b.LC モデムの管理機能にシリアルで接続する場合

5.3.3 方式ごとの考察

6.4.1、6.4.2 で述べた方式について考察する。

(a) PLC モデムの管理機能にシリアルで接続する方式は、HTIP Agent の必要な配線数が 1 つのみという利点があり、後述の(b)と比べ、ネットワークの管理が用意である。そのため、一般家庭・小規模店舗などトラフィックの少ないネットワークに向いていると考えられる。

(b) PLC モデムの管理機能にシリアルで接続する方式は、通信を行う Ethernet 線と、情報の取得を行うシリアル線が独立している点より、ネットワークのトラフィックへの影響が少ない利点が存在する。よって、企業などの比較的大規模ネットワークに向いていると考えられる。

第6章 まとめ

当研究では、IoT エリアネットワークにおける低コスト管理運用技術の実現を目的とした。この IoT エリアネットワークは、従来から使われている Ethernet・IP ネットワークと整合性を保つように連携する必要がある。この管理運用技術として、その動作の軽さの点から HTIP を採用し、これを IoT エリアネットワークで運用するための技術課題を 2 つ挙げ、それらの解決を目指した。

課題 1 の非 Ethernet・非 IP ネットワークにおける管理運用技術としての HTIP の適用検討では、Ethernet・IP 前提の HTIP をエリアネットワークに適用するための考察を行った。種類の違うネットワーク同士を接続したときのアーキテクチャを列挙し、それらを基にデータの伝送方式を決定した。

ネイティブフレーム伝送方式のフレーム形式は、現在 TTC の HTIP ワーキンググループにおいて制定中の規格をベースとして改良を加えたものを提案した。

みなしシリアル方式の HTIP エージェント、マネージャーと変換器を実装し、正しくトポロジを取得できることを確認した。シリアル通信のトポロジが、エージェントが複数存在する場合、データに識別子が必要となる。子の識別子の種類や識別子をエージェントに割り当てる方法について考察した。

課題 2 の既存ネットワーク機器の HTIP 対応手法の検討では、既存の管理機能を有するネットワーク機器類の HTIP 対応手法として、外付けの HTIP エージェントによる手法を検討した。この手法では、既存ネットワーク機器に HTIP エージェントを実装した外付け HTIP エージェント機器を、Ethernet もしくはシリアルで接続する。これにより、既存のネットワーク資源のリプレイス必要としない HTIP 対応が可能となる。当研究では、スイッチとブリッジと HTIP エージェントの接続方法を列挙し、それらの利点について述べた。

今後の課題として、HTIP のフレーム送信間隔などのパラメータに通信品質の変化などの評価をする必要がある。そのため、非 Ethernet・非 IP 向け HTIP のフル実装が求められる。

謝辞

本研究に当たっては、終始ご指導を頂いた丹康雄教授に深く感謝申し上げます。また、研究上の相談に乗っていただいた丹研究室の牧野義樹様にも感謝申し上げます。

当研究は、内閣府事業 PRISM(官民研究開発投資拡大プログラム)IoT 共通基盤技術の確立・実証の一部となる。

参考文献

[1] “Wikipedia Link Layer Discovery Protocol,” 2007. [オンライン]. Available: https://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol. [アクセス日: 2019].

[2] TCC, “JJ-300.00 ホーム NW 接続構成特定プロトコル”.

[3] Bluetooth SIG, “Bluetooth Profile Specification：Serial Port Profile ver.12,” 2012.

[4] TCC, “JJ-300.01 端末区分情報リスト”.

[5] TCC, “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”.

[6] 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”.

[7] TCC, “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～”.

[8] TCC, “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”.

[9] IEC, “62608-1:2014 Multimedia home network configuration - Basic reference model - Part 1: System model”.

[10] IEC, “62608-2:2017 Multimedia home network configuration - Basic reference model - Part 2: Operational model”.

[11] 富士エレクトロニクス株式会社, [オンライン]. Available: https://www.fujiele.co.jp/semiconductor/step/engineering/20171219-1/. [アクセス日: 1 2019].

付録

非 Ethernet・非 IP 向け HTIP エージェント実装

LLDPDU_Serial.c

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>

#include "datalink.h"
#include "ifinfo.h"
#include "tlv.h"
#include "htip.h"
#include "fdb.h"
#include "cobs.h"

#define DEVICE_NAME
"/dev/ttyUSB0"
#define LLDPDU_INTERVAL 3
//LLDPDU interval of sending
#define MEM_LEN 512

#define TRUE 1
#define FALSE 0

volatile int STOP=FALSE;
int fd;
struct termios oldtio,newtio;

void sig_handler(int sig){

  /* return termios struct
from backup */
  tcsetattr(fd, TCSANOW,
&oldtio);
  close(fd);
  exit(0);
}

int main(int argc, char*
argv[]){

  int len, rlen;
  u_char* payload;
  u_char* encoded;
  cobs_encode_result
res_encoded;

  /* define interface address
and name  */
  u_char macaddr[] = { 0x0,
0x1, 0x2, 0x3, 0x4, 0x5 };
  u_char ifname[] =
"lovelyPort";
  u_char device_category[] =
"PC";
  u_char manufacturer_code[]
="JAIST";
  u_char model_name[] =
"JAIST_YAMAMOTO_INTELNUC";
  u_char model_number[] =
"6/15";


  fd = open(DEVICE_NAME,
O_RDWR | O_NOCTTY );
  if( fd<0 ){

    perror(DEVICE_NAME);
    exit(-1);
  }

  /* allocating memory */
  if ((payload =
malloc(ETH_DATA_LEN)) ==
NULL){
    perror("malloc");
    return -1;
  }

  if ((encoded =
malloc(ETH_DATA_LEN * 2)) ==
```

```
NULL){
    perror("malloc");
    return -1;
  }

  memset(payload, 0,
ETH_DATA_LEN);
  memset(encoded, 0,
ETH_DATA_LEN * 2);


  /* set interface info,ifip
*/



  /* creating TLV */
  len = 0;
  len +=
create_lldp_tlv(payload,
macaddr, ETHER_ADDR_LEN,
ifname, strlen(ifname));

  if ((rlen =
create_basic_htip_device_info_
tlv(

             payload + len,
macaddr, ETHER_ADDR_LEN,

             ifname,
strlen(ifname),

             device_category,
strlen(device_category),
manufacturer_code,

             model_name,
strlen(model_name),
model_number,



      strlen(model_number)))
== 0) {
    fprintf(stderr,
"create_required_htip_device_i
nfo_tlv() failed¥n");
    return -1;
  }

  len += rlen;
  len +=
create_end_of_lldpdu_tlv(paylo
ad + len);

  /* encode COBS */
  res_encoded =
cobs_encode(encoded, MEM_LEN,
payload, len);
  if(res_encoded.status !=
COBS_ENCODE_OK){


printf( "cobs_encode:status=%i
¥n", res_encoded.status);
    return -1;
  }

#ifdef DEBUG
  printf("  htip frame
created: %d bytes using
macaddr: %02x:%02x:%02x:%02x:%
02x:%02x, ifname: %s.¥n",
        len, macaddr[0],
macaddr[1], macaddr[2],
macaddr[3], macaddr[4],
macaddr[5], ifname);
#endif /* DEBUG */

  /* set signal handler, abort
Ctrl + C */
  signal(SIGINT, sig_handler);

  tcgetattr(fd, &oldtio);
      /* escape old termio
struct */
  bzero(&newtio,
sizeof(newtio)); /* initialize
new termios */

  /* set baudrate, hadware
flow at output, 8bit no parity
```

```
stop bit 1 */
  /* ignore modem control,
enable receiver */
  newtio.c_cflag = CS8 |
CLOCAL | CREAD;

  /* igonere parity error, tie
CR to NL */
  newtio.c_iflag = IGNPAR |
ICRNL;

  newtio.c_oflag = 0;

  /* enable canonical input,
dont send util Enter */
  // newtio.c_lflag = ICANON;
  newtio.c_lflag = 0;

  /* set baudrate */
  cfsetispeed(&newtio, B9600);
  cfsetospeed(&newtio, B9600);

  /* flush unread received
data in tty device */
  tcflush(fd, TCIFLUSH);
  /* activate setting of
newtio, change immediately */
  tcsetattr(fd, TCSANOW,
&newtio);

  while(1){

    //if(res>0) buf[res-1]=0;
    printf("writing...¥n");
    write(fd, encoded, len+1);
    write(fd, "¥0", 1);
#ifdef DEBUG
    printf("encoded:");
    for(int i=0;
i<res_encoded.out_len; i++){


printf("%x,",encoded[i]);
    }
    printf("¥t len=%i¥n",
res_encoded.out_len);
#endif  /* DEBUG */
    sleep(LLDPDU_INTERVAL);

  }

}
```

Serial/Ethernet 変換機実装

LLDP_GW.c

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>

#include "datalink.h"
#include "ifinfo.h"
#include "tlv.h"
#include "htip.h"
#include "fdb.h"
#include "cobs.h"
#include "ringbuf.h"

#define DEVICE_NAME
"/dev/ttyUSB1"
#define INTERFACE_NAME "eno1"
#define RINGBUF_SIZE 512
#define BUF_SIZE 512

#define TRUE 1
#define FALSE 0

#define DEVICE_CATEGORY_LEN
256
#define MANUFACTURER_CODE_LEN
6
#define MODEL_NAME_LEN 32
#define MODEL_NUMBER_LEN 32

volatile int STOP=FALSE;
int fd,sockfd;
```

```
struct termios oldtio,newtio;

void finalizer();

void sig_handler(int sig){

  finalizer();
}

void finalizer(){

  /* return termios struct
from backup */
  tcsetattr(fd, TCSANOW,
&oldtio);
  close_netif();
  free_ifinfo_list();
  close(fd);
  close(sockfd);
  exit(EXIT_SUCCESS);
}

int main(int argc, char*
argv[]){

  u_char* payload;        /*
COBS encoded LLDPDU */
  u_char* decoded;        /*
raw LLDPDU */
  u_char* disposal;       /*
/dev/null for memcpy */

  cobs_decode_result
res_decoded;
  ringbuf_t rbuf;
  size_t index;

  struct ifinfo* inf =
get_empty_ifinfo();
  u_char dstaddr[] =
HTIP_L2AGENT_DST_MACADDR;
  /* initialize res_decoded */
  res_decoded.out_len = 0;
  res_decoded.status = 0;

  /* store network interface
information */
  if (read_ifinfo() < 0) {
    fprintf(stderr,
"read_ifinfo() failed¥n");
    return (EXIT_FAILURE);
  }

  /* store network interface
type */
  if (read_net_type() == -1) {
    fprintf(stderr,
"read_net_type() failed.¥n");

  }

  printf("open_netif()¥n");
  /* get network interfaces */
  if (open_netif() < 0) {
    fprintf(stderr,
"get_netif_osx() failed¥n");
    finalizer();
  }


printf("set_promiscuous_mode()
¥n");
  /* set promiscas mode */

if(set_promiscuous_mode( INTER
FACE_NAME ) == -1){


perror("set_promiscuous_mode_f
ailure¥n");
    finalizer();
  }

  /* check stored network
interface list */
  print_ifinfo();

  inf =
search_ifinfo_by_ifname( INTER
FACE_NAME);
  if( inf == NULL){
```

```
perror("search_ifinfo_by_ifnam
e failure¥n");
    finalizer();
  }

  /* open serial  */
  fd = open(DEVICE_NAME,
O_RDWR | O_NOCTTY );
  if( fd<0 ){

    perror(DEVICE_NAME);
    exit(-1);
  }

  /* allocating memory */
  if ((payload =
malloc(BUF_SIZE)) == NULL){
    perror("malloc");
    return -1;
  }
  memset( payload, 0,
BUF_SIZE);

  if ((decoded =
malloc(BUF_SIZE)) == NULL){
    perror("malloc");
    return -1;
  }
  memset( decoded, 0,
BUF_SIZE);

  if(( disposal =
malloc(BUF_SIZE)) == NULL){
    perror("malloc");
    return -1;
  }
  memset( disposal, 0,
BUF_SIZE);

  /* initialize ring buffer */
  rbuf =
ringbuf_new(RINGBUF_SIZE);
  if(rbuf == NULL){

    perror("ringbuf_new");
    return -1;

  /* set signal handler, abort
Ctrl + C */
  signal(SIGINT, sig_handler);

  /* setting of Termios */

  tcgetattr(fd, &oldtio);
      /* escape old termio
struct */
  bzero(&newtio,
sizeof(newtio)); /* initialize
new termios */

  /* set baudrate, hadware
flow at output, 8bit no parity
stop bit 1 */
  /* ignore modem control,
enable receiver */
  newtio.c_cflag = CS8 |
CLOCAL | CREAD;

  /* igonere parity error, tie
CR to NL */
  newtio.c_iflag = IGNPAR |
ICRNL;

  newtio.c_oflag = 0;

  /* enable canonical input,
dont send util Enter */
  // newtio.c_lflag = ICANON;
  newtio.c_lflag = 0;

  /* set baudrate */
  cfsetispeed(&newtio, B9600);
  cfsetospeed(&newtio, B9600);

  /* flush unread received
data in tty device */
  tcflush(fd, TCIFLUSH);
  /* activate setting of
newtio, change immediately */
  tcsetattr(fd, TCSANOW,
&newtio);
```

```
  }

  /* read all buffer and cut
off imcomplete frames */
  while(ringbuf_read(fd, rbuf,
RINGBUF_SIZE)>0){
    index =
ringbuf_findchr(rbuf,0, 0);

ringbuf_memcpy_from(disposal,
rbuf, index);
    /* remove extra '0' from
ring buffer */

ringbuf_memcpy_from(disposal,
rbuf, 1);
  }


  for(;;){

    sleep(1);
    ringbuf_read(fd, rbuf,
RINGBUF_SIZE);
    index =
ringbuf_findchr(rbuf, 0, 0);
    if( (index <
ringbuf_bytes_used(rbuf)) &&
(0 < index)){

      /* find separator '¥0'
*/

ringbuf_memcpy_from(payload,
rbuf, index);

      /* remove extra '0' from
ring buffer */

ringbuf_memcpy_from(disposal,
rbuf, 1);
      /* decode LLDPDU */
#ifdef DEBUG
      printf("payload:");
      for(int i=0; i<index;
i++){

      printf("%x,",payload[i])
;
      }
      printf("¥t len=%li¥n",
index);
#endif /* DEBUG */
      res_decoded =
cobs_decode((u_char*)decoded,
BUF_SIZE, payload, index);

      if(res_decoded.status !=
COBS_DECODE_OK){

      perror("cobs_decode");
      printf("status=%li¥n",
res_decoded.status);
      continue;
      }

#ifdef DEBUG
      printf("index=%i¥n",
index);
      printf("decoded=%s¥n",
decoded);
#endif  /* DEBUG */

      if( write_frame( inf-
>fd, INTERFACE_NAME, dstaddr,
inf->macaddr, decoded,
res_decoded.out_len) < 0 ){

      perror("write_frame_fail
ure¥n");
      return -1;
      }
      printf("sending...¥n");
    }
  }
}
```