Vol.2019-ARC-240 No.35
Vol.2019-SLDM-191 No.35
Vol.2019-EMB-53 No.35
2019/2/28

組込み Linux の Time-Sensitive Networking 性能評価

出原 章雄[†1]　桐村 昌行[†1]

概要：近年，汎用 Ethernet デバイスの低価格化・高機能化から，産業用などの組込み機器間の通信に，独自の専用バスではなく，汎用 Ethernet を使いたいという要求が高まっている．しかし，Ethernet は一般的にスループット向上を目指すベストエフォート型であり，通信の遅延時間が保証されておらず，定周期で通信を行うには課題がある．

そこで IEEE では，TSN (Time-Sensitive Networking)にて，Ethernet 上で通信のリアルタイム性を保証するための仕様を定義している．特に TSN の重要な機能として，通信タイミングのスケジューリング機能(IEEE 802.1Qbv，以降 Qbv)がある．Linux は Qbv 対応に向け，2018 年 10 月に ETF (Earliest TxTime First), 2019 年 9 月に TAPRIO(Time Aware Priority Shaper)を導入した．ETF と TAPRIO を組み合わせることで簡易的に Qbv を実現可能となる．このことから，組込み Linux 上に ETF/TAPRIO 環境を構築し，定周期送信アプリとベストエフォート型アプリ混在時の評価を実施した．

結果，H/W offload 無効の場合，定周期送信アプリの送信周期からのずれは 40us 超に対し，H/W offload 有効の場合は 50ns 以下となった．これにより，マイクロ秒の精度を要求する TSN において，H/W offload 有効の場合，組込み Linux を適用可能となる見込みを得た．次に，ベストエフォート型アプリの TCP スループットについて，ETF/TAPRIO 無効時は約 990Mbps に対し，ETF/TAPRIO 有効時は約 360Kbps となり，数 Mbps 以上の帯域を要求するアプリで問題となることが判明した．原因調査の結果，ETF は送信期限を超過したベストエフォート型アプリの送信フレームを破棄しており，これにより TCP の輻輳処理が動作していたためであった．これらの課題解決に向けては ETF/TAPRIO に対し，フレーム送信タイミングの再スケジュール機能が必要であり，今後はこうした課題解決を図る．

キーワード：Linux, PTP, Ethernet, Earliest TxTime First, ETF, Time Aware Priority Shaper, TAPRIO, Txtime-Assist, NIC, Qbv

Performance Evaluation of Time-Sensitive Networking on Embedded Linux System

Akio Idehara[†1]　Masayuki Kirimura[†1]

Keywords: Linux, PTP, Ethernet, Earliest TxTime First, ETF, Time Aware Priority Shaper, TAPRIO, Txtime-Assist, NIC, Qbv

1. はじめに

近年, 汎用 Ethernet デバイスの低価格化･高機能化から，組込み機器間の通信に，独自の専用バスではなく，汎用 Ethernet を使いたいという要求が高まっている．しかし，Ethernet は一般的にスループット向上を目指すベストエフォート型であり，通信の遅延時間が保証されておらず，定周期で通信を行うには課題がある．

こうした用途では，近年 Time Sensitive Networking(TSN)の適用が検討されており，Linux®においても，TSN が簡易的に実装されている．

そこで, 本稿では組込み Linux の TSN について評価する．次節で関連技術，3 節で実現性，4 節で評価方法，5 節で結果．6 節で考察について述べ，最後に本論文をまとめる．

2. 関連技術

TSN を用いる場合の重要な機能としては，IEEE Std 802.1AS(Generalized Precision Time Protocol，通称 gPTP)と IEEE Std 802.1Qbv(以降 Qbv)が存在する[1]．gPTP はノード間で時刻を高精度に同期するプロトコルである．Qbv は定周期フレーム送信に必要な機能である．

Linux における gPTP について, 既にオープンソースによる linuxptp[2]，gPTPd[3]などの実装が存在しており，汎用的に利用可能である．対して，Linux における Qbv 対応はまだ始まったばかりである．そこで本稿では，Qbv について論じる．

2.1 Qbv の概要

Qbv は IEEE Std 802.1Q の一機能として仕様策定が進んでいた機能であり，IEEE Std 802.1Q-2018[4]にて，正式に IEEE 標準として取り込まれた．図 1 は Qbv の概念を示している(参考文献[4]の 8.6.8.4 節 Figure8.14 をベースに筆者ら改変).

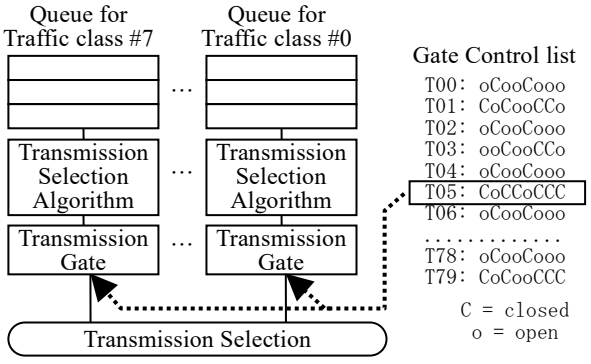

図 1　Qbv の概念図

†1 三菱電機株式会社情報技術総合研究所
Mitsubishi Electric Corp. Information Technology R&D Center

Vol.2019-ARC-240 No.35
Vol.2019-SLDM-191 No.35
Vol.2019-EMB-53 No.35
2019/2/28

図 1 に示す通り，Qbv では，各 Traffic Class に対してキューが割り当たっており(図 1 中 Queue for Traffic class #0～#7)，各キューに対するゲートの Open/Closed がスケジュールされている(図 1 中 Gate Control list を用いて Transmission Gate を制御)．また，図 1 中 Transmission Selection Algorithm にて，優先度ベースなどのアルゴリズムでフレーム送出を制御可能である．これらの仕組みにより，特定のタイミングで，どのキューからフレームを送出するかを制御可能となる．

2.2 Qbv の H/W サポート

前節に記載のとおり，Qbv は複数の送信キュー，送信アルゴリズム，および，送信タイミングを制御するゲートで構成される．これらの機能を高精度に実現するためには，次のような H/W サポートが必要となる．

・優先度付き H/W キュー
・送信タイミングの絶対時刻指定
・送信キューの再スケジューリング

2.3 Linux における Qbv サポート状況

本節では，Linux における Qbv のサポート状況を述べる．

(1) **ETF**

2018 年 10 月にリリースされた Linux 4.19 にて，ETF (Earliest TxTime First, Time-based Packet Transmission とも呼ばれる)と呼ばれるフレーム送信制御機能が導入された[5]．ETF を用いることで，アプリケーションは NIC に対し，フレームの送信タイミングを絶対時刻で指定可能となる．すなわち，Qbv のうち，フレーム送信タイミングの指定が可能となる．我々の ETF に関する評価では，送信タイミングからのぶれは数十 ns 以下となり，ETF を用いることで Qbv が実現可能となる見込みを得た[1]．

ただし，ETF を用いていないアプリの送信タイミングは不定となる．すなわち，ETF のみを用いて Qbv を実現する場合，ユーザが全てのフレーム送信タイミングを制御する必要がある．しかし，Linux においては，Web サーバなど，既存のベストエフォート型アプリが多数存在している．ユーザがこうしたアプリの送信タイミングを都度指定するのは現実的ではない．

(2) **TAPRIO**

ETF におけるベストエフォート型アプリの送信タイミングの問題を解決するため，2019 年 9 月にリリースされた Linux 5.3 にて，Txtime-Assist に対応した TAPRIO(Time Aware Priority Shaper)が導入された[6]．TAPRIO を用いることで，ユーザは既存アプリの送信タイミングを意識することなく，ETF を利用可能となる．すなわち，ETF を用いたアプリは，ユーザ指定のタイミングでフレーム送信される．対して，ETF を用いていないアプリは，TAPRIO が設定したベストエフォート型アプリ向けのタイミングでフレーム送信される．

ここで，TAPRIO の送信タイミング制御について，Txtime-Assist 未対応の TAPRIO は，OS のシステムタイマを利用していたのに対し，Txtime-Assist 対応の TAPRIO は，NIC の H/W タイマを利用する．これにより，高精細に送信タイミングの制御が可能となっている．

なお，Txtime-Assist 対応 TAPRIO は，Qbv の簡易実装の位置づけであり，H/W サポートが必要な機能のうち，送信タイミングの絶対時刻指定機能のみを使用している．省略されている機能としては，複数ゲートのオープン，および，送信キューの再スケジュール等がある．

IEEE 仕様に記載の Qbv を全て実現するには，優先度付き H/W キュー，および，送信タイミングの再スケジュールなどの H/W サポートが必要となる．

3. 組込み Linux における Qbv の実現性

前節で述べたように，Linux において，ETF と TAPRIO を用いて基本的な Qbv が実現可能である．

そこで本稿では，よりマシンパワーの低い組込み機器向けの H/W 構成(Intel Atom)において，ETF と TAPRIO を組み合わせた場合の，定周期送信タイミングのぶれを評価し，組込み Linux での Qbv の実現可能性を判断する．

4. 評価

4.1 評価概要

今回の評価では，産業用途を意識し 1ms で定周期送信を行うアプリ 2 つと，ベストエフォート型アプリが共存する環境を想定した．定周期送信アプリは，ほぼ同一のアプリであり，それぞれ異なる制御プロトコルの定周期動作を想定する．定周期送信に割り当てる時間はリアルタイム処理を想定して各 300us とし，残りの 400us をベストエフォート型アプリに割り当てる．

以降，各周期の割り当てについて Qbv の定義に基づき TC(Traffic Class)と呼ぶ．TC0，TC1 は ETF を用いてユーザが送信タイミングを指定する．TC2 ではベストエフォート型アプリを想定し，iperf3[7]を動作させる．今回の評価で想定する Traffic Class を表 1，および図 4 に示す．

表 1　Traffic Class 設定

Traffic Class #	長さ[us]	フレーム種別
TC0	300	ETF による定周期送信
TC1	300	ETF による定周期送信
TC2	400	TAPRIO による iperf3 のベストエフォート送信

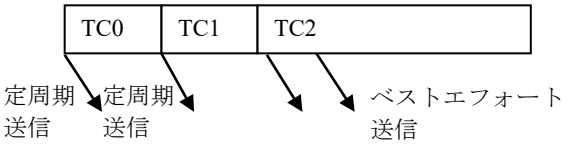

図 2　Traffic Class 設定

Vol.2019-ARC-240 No.35
Vol.2019-SLDM-191 No.35
Vol.2019-EMB-53 No.35
2019/2/28

4.2 評価項目

ETF と TAPRIO を組みわせた場合の性能評価として，次の内容を評価する．

(1) TC0 の送信周期(1ms)からのぶれ
(2) TC1 の送信周期(1ms)からのぶれ
(3) TC0 と TC1 の送信タイミング差(300us)からのぶれ
(4) TC2 の UDP/TCP 通信スループット

4.3 評価環境

H/W，S/W を含めた評価環境の全体構成を図 3 に示す．

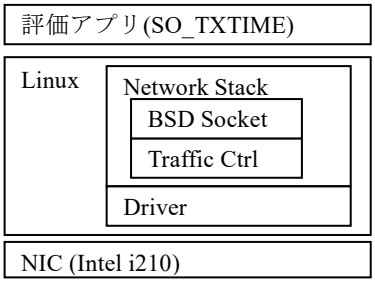

図 3　全体構成

評価環境を表 2，表 3，および，表 4 に示す．

表 2　H/W 環境

H/W: Minnow Board Turbot Dual Ethernet Quad-Core	
CPU	Intel Atom E3845 1.91GHz (4Core)
Memory	2GB
NIC	Intel i210 (2port)

表 3　S/W 環境

S/W	
OS	Linux 5.3.11 (CONFIG_NET_SCH_ETF, CONFIG_NET_SCH_TAPRIO を有効化)
libc	glibc 2.30
gcc	gcc 9.2.1
iproute2	5.3.0+Txtime-Assist 追加

表 4　評価用ツール

LAN アナライザ:
Profishark 1G plus (PROFITAP 社製) Timestamp Resolution: 8ns (HW Firmware Version: 0313)

ターゲットの接続を図 2 に示す．

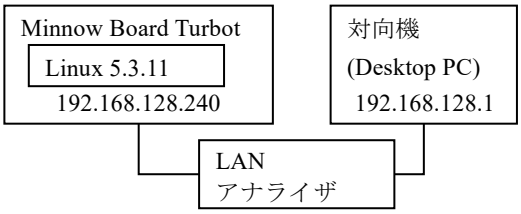

図 4　ターゲット接続

4.4 評価内容詳細

以降，本節では評価内容の詳細を記載する．

4.4.1 評価アプリ動作

(1) TC0，および，TC1 向けアプリ

TC0 用と TC1 用の評価アプリを用意し，それぞれ，1ms 周期でフレーム送信を 60 秒(60,000 フレーム)行い, LAN アナライザにて送信タイミングを取得する．

評価アプリの詳細について，clock_nanosleep システムコールを用いて, 送信時刻より前に起床する(送信マージン(今回 400us))．起床後，SO_TXTIME を使用し，送信時刻を指定してフレーム送信を行う．以降，起床時刻および，送信時刻を 1ms ずつ増加させて，この処理を繰り返す(図 11).

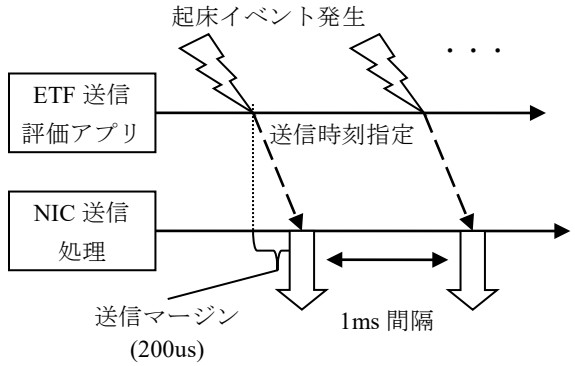

図 5　ETF 評価アプリ動作

上記アプリを TC0，TC1 のタイミングで送信されるように設定する．

(2) TC2 向けアプリ

素の状態で iperf3 を実施し，ETF/TAPRIO の有効性とスループットを確認する．比較として，ETF/TAPRIO を無効化し，スループットを確認する．

4.4.2 評価設定

評価準備として, Linux の ETF 設定手順を図 6 に示す[8]. また，各種の設定値の関係を表 5 にまとめる．

図 6(1)にて，phc2sys コマンドを使用し NIC 時刻(/dev/ptp0)と，システム時刻(CLOCK_REALTIME)を同期させる．

図 6 (2)にて，tc コマンドを使用し Traffic Control を表 6 に設定する．

図 6 (3)，(4)にて，Traffic Control を表 7 に設定する．

以上の設定により，フレーム送信時に，ソケットに対し SO_TXTIME，SO_PRIORITY=3 or 2，および送信時刻を設定すると Intel i210 の H/W キュー#0 に対し，送信時刻が設定される．Intel i210 は設定された送信時刻を経過すると，該当フレームを送出する．また，通常のベストエフォート通信の場合は TC2 のタイミングでフレームが送信される．

Vol.2019-ARC-240 No.35
Vol.2019-SLDM-191 No.35
Vol.2019-EMB-53 No.35
2019/2/28

```
phc2sys -s /dev/ptp0 -c CLOCK_REALTIME -O 0 & << (1)

tc qdisc replace dev enp2s0 parent root ¥
  handle 100 taprio ¥
  num_tc 3 ¥
  map 2 2 1 0 2 2 2 2 2 2 2 2 2 2 2 2 ¥
  queues 1@0 1@0 1@0 ¥
  base-time 0 ¥
  sched-entry S 01 300000 ¥
  sched-entry S 02 300000 ¥
  sched-entry S 04 400000 ¥
  flags 0x1 ¥
  txtime-delay 40000 ¥
  clockid CLOCK_TAI << (2)

tc qdisc replace dev enp2s0 parent 100:1 etf ¥
  skip_sock_check ¥
  offload delta 200000 clockid CLOCK_TAI << (3)

tc qdisc replace dev esp2s0 parent 100:2 etf ¥
  skip_sock_check ¥
  offload delta 200000 clockid CLOCK_TAI << (4)
```

図 6　評価準備用の ETF 設定

表 5　TC，SO_PRIORITY，および HW キューの関係

TC	SO_PRIORITY	H/W キュー
0	3	0 (送信時刻指定可，すべての TC が同じキューを使用)
1	2	
2	0,1,および，4~15	

表 6　TC 設定 1

引数	意味
qdisc	Queueing Discipline
replace dev enp2s0	NIC 名 enp2s0 の設定を変更
parent root	root を親とする
handle 100	ハンドル値を 100 とする
taprio	Time Aware Priority Shaper を設定する
num_tc 3	Traffic Classes(TC)数を 3 に指定
map 2 2 1 0 2 2 ...	SO_PRIORITY と TC#の関係を定義 SO_PRIORITY=2 (2 番目の map)を TC1 SO_PRIORITY=3 を TC0 SO_PRIORITY=0,1,および,4～15 を TC2
queues 1@0 1@0 1@0	H/W キューに対して次を実施 TC0 に 1 つのキュー(Q#0)を割り当て TC1 に 1 つのキュー(Q#0)を割り当て TC2 に 1 つのキュー(Q#0)を割り当て 本設定は Txtime-Assist のみで有効
base-time 0	本設定の開始時刻を一周期後に設定
sched-entry S 01 300000...	表 1，および図 4 の設定を実施
flags 0x1	Txtime-Assist(bit0)を有効化
txtime-delay 100000	フレーム送信までの最大時間を 40000 ns に設定する
clockid CLOCK_TA	TAPRIO で使用するクロック源を CLOCK_TA とする

表 7　TC 設定 2

引数	意味
qdisc	Queueing Discipline
add dev enp2s0	NIC 名 enp2s0 の設定を追加
parent 100:1 / parent 100:2	ハンドル値 100:1(TC0)を親とする / ハンドル値 100:2(TC1)を親とする
etf	Early Txtime First を設定する
skip_sock_check	SO_TXTIME による時刻指定が無くても送信する
offload	H/W オフロード機能を有効にし，送信タイミングを高精度化
delta 200000	送信時刻の 200000 ns 前に送信キューに追加する
clockid CLOCK_TAI	ETF で使用するクロック源を CLOCK_TAI とする

4.4.3 評価アプリ実装

(1) TC0，および，TC1 向け処理内容

今回の評価アプリケーションのうち，TC0 による定周期送信アプリで実施する初期化処理を図 7 に示す. TC1 もほぼ同様である

```
so_priority = 3;
setsockopt(fd, SOL_SOCKET, SO_PRIORITY, &so_priority,
sizeof(so_priority)) << (1)

sk_txtime.clockid = clkid;
setsockopt(fd, SOL_SOCKET, SO_TXTIME, &sk_txtime,
sizeof(sk_txtime)) << (2)
```

図 7　初期化処理

図 7 (1)にて, SO_PRIORITY を 3 に設定する(TC1 の場合は 2). 前節の設定から, TC0, TC1 共に Intel i210 の H/W キュー#0 が利用される.

図 7 (2)にて，SO_TXTIME の設定を有効にする.

次に，送信処理を図 8 に示す.

```
msg.msg_control = control;
msg.msg_controllen = sizeof(control);

cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SCM_TXTIME; << (1)
cmsg->cmsg_len = CMSG_LEN(sizeof(__u64));
*((__u64 *) CMSG_DATA(cmsg)) = txtime; << (2)
sendmsg(fd, &msg, 0); << (3)
```

図 8　送信処理

図 8 (1)にて，SCM_TXTIME を設定し，送信時刻指定を有効化する.

図 8 (2)にて，送信時刻 txtime を指定する.

図 8 (3)にて，(1)，(2)の情報を含めた上で semdmsg システムコールを用いて，フレームを送信する.

Vol.2019-ARC-240 No.35
Vol.2019-SLDM-191 No.35
Vol.2019-EMB-53 No.35
2019/2/28

(2) TC2 向け処理内容

N/W スループットのベンチマーク S/W である iperf3 を用いて，TC 設定の有無でスループットの変化を確認する．プロトコルによる差異を確認するために UDP と TCP で実施する．クライアント側(MinnowBoard)の UDP 設定値を図 9，TCP 設定値を図 10，サーバ側(対向機 LinuxPC)設定値を図 11 に示す(UDP/TCP 共通)．

```
# iperf3 -c 192.168.128.1 -t 600 -u -b 1000M
```

図 9　クライアント側(Minnow Board)の iperf3 設定 (UDP)

```
# iperf3 -c 192.168.128.1 -t 600
```

図 10　クライアント側(Minnow Board)の iperf3 設定 (TCP)

```
# iperf3 -s
```

図 11　サーバ側(対向機 Linux PC)の iperf3 設定

5. 結果

本節では評価結果を記載する．ここで，H/W offload 機能(Txtime-Assist)の比較として，offload 無効時(OS のタイマ使用)の評価も記載する．offload 無効設定は図 6 の(3)，(4)にて"offload"を削除する．

なお，今回の結果について，SMM (System Management Mode)移行による 600us 以上の遅延を除外した．SMM の移行について，今回の評価ボードでは温度変化(Thermal Monitor)による SMI (System Management Interrupt)発生等が契機となっている．SMI 処理は評価ボードに特有の処理であり，ETF，および，TAPRIO とは無関係のため，今回の評価結果から除外した．

5.1 (1) TC0 の送信周期(1ms)からのぶれ

LAN アナライザにて TC0 における送信フレームを取得し，各フレームの通信周期(1ms)に対するぶれを算出した．測定は 15 秒程度である．結果を表 8 に示す．

表 8　TC0 の評価結果

	offload 有効		offload 無効	
	UDP	TCP	UDP	TCP
平均 [ns]	2.2	6.2	2.3	6.5
最大 [ns]	24.0	24.0	213824.0	4736.0
最小 [ns]	-24.0	-16.0	-209952.0	-3288.0
最大 - 最小 [ns]	48.0	40.0	423776.0	8024.0
標準偏差	10.6	8.7	15191.4	431.3

また，UDP の場合の，個々の送信フレームにおける周期からのぶれを図 12，および図 13 に示す．TCP の場合はほぼ同様の形状であり，図を省略する．

評価の結果，UDP/TCP 共に offload 有効で ETF を用いた場合，10 ナノ秒オーダのぶれで通信が可能と判明した．特に最大と最小の差は 48ns となった．一般的な TSN のユースケースではマイクロ秒の精度が求められるが，Intel Atom を使用した場合においても，H/W サポートのある NIC を用いることで，組込み Linux 上で TSN 通信を実現可能という見込みを得た．

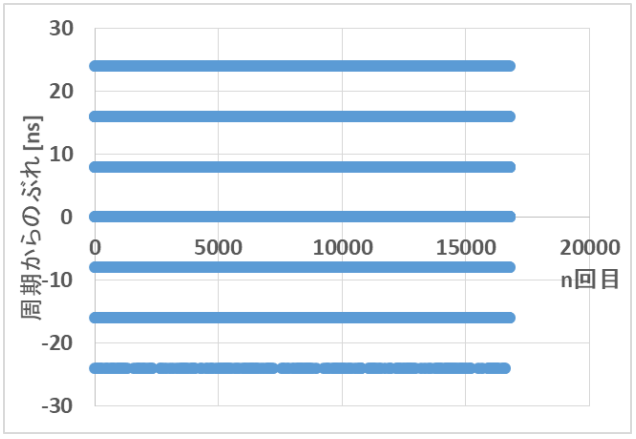

図 12　offload 有効(UDP)の送信結果 (TC0)

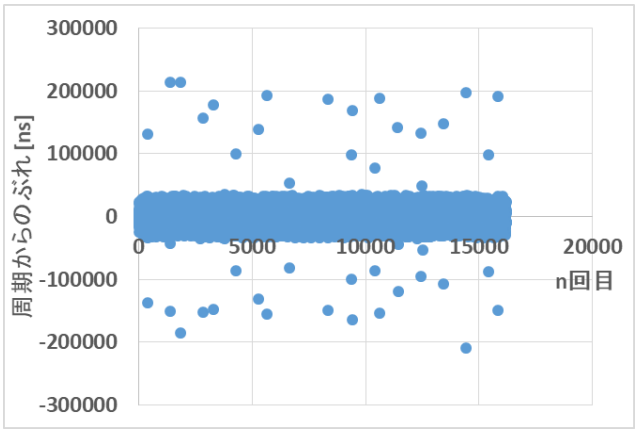

図 13　offload 無効(UDP)の送信結果 (TC0)

対して，UDP/TCP 共に offload 無効で ETF を用いた場合，マイクロ秒オーダ以上で遅延が生じる．一般的な TSN 通信のユースケースでは，マイクロ秒の精度が求められるため，TSN 通信を実現できない可能性が高い．特に最大と最小の差について，TCP が 8us 程度に対して，UDP の場合は 400us 程度となった．本件については 6.3 節で考察する．

5.2 (2) TC1 の送信周期(1ms)からのぶれ

前節同様，LAN アナライザにて TC1 における送信フレームを取得し，各フレームの通信周期(1ms)に対するぶれを算出した．測定は 15 秒程度である．結果を表 9 に示す．

また，UDP の場合の，個々の送信フレームにおける周期からのぶれを，図 14 および図 15 に示す．TCP の場合はほぼ同様の形状であり，図を省略する．

評価の結果，UDP/TCP 共に offload 有効で ETF を用いた場合，TC0 と同様 48ns のぶれで通信が可能と判明した．複数の TC で ETF を用いた場合でも，H/W による offload が

Vol.2019-ARC-240 No.35
Vol.2019-SLDM-191 No.35
Vol.2019-EMB-53 No.35
2019/2/28

可能であれば，数マイクロ秒オーダの精度を実現可能という見込みを得た．

表 9　TC1 の評価結果

	offload 有効		offload 無効	
	UDP	TCP	UDP	TCP
平均 [ns]	2.2	6.2	3.2	6.6
最大 [ns]	24.0	24.0	20920.0	27232.0
最小 [ns]	-24.0	-16.0	-20648.0	-27736.0
最大 - 最小 [ns]	48.0	40.0	41568.0	54968.0
標準偏差	10.7	8.7	4152.1	1038.7

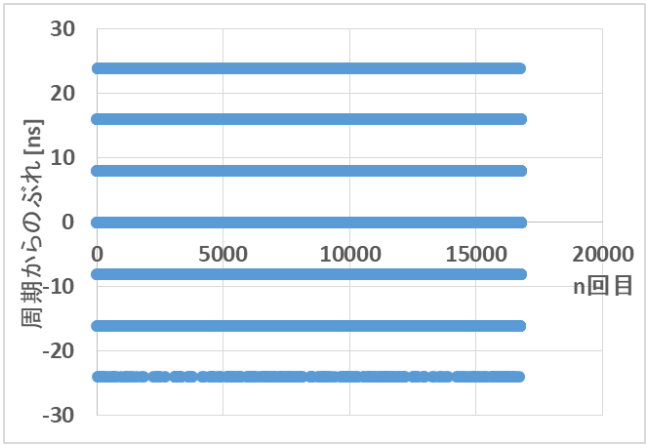

図 14　offload 有効(UDP)の送信結果 (TC1)

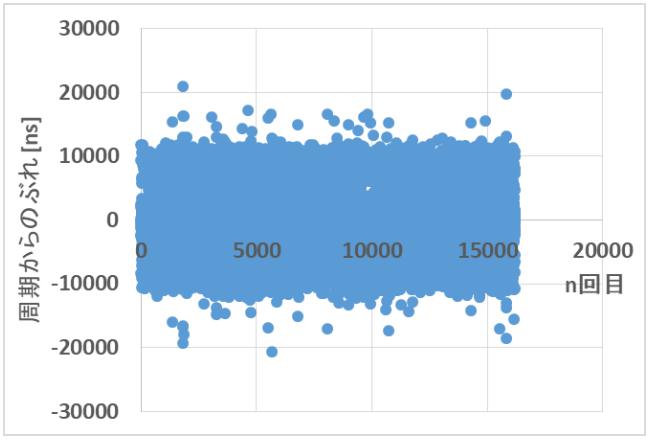

図 15　offload 無効(UDP)の送信結果 (TC1)

対して，UDP/TCP 共に offload 無効で ETF を用いた場合，マイクロ秒オーダで遅延が生じ，最大と最小の差は 55us 程度となった．なお，TC0 の UDP では 430us であったのに対し，TC1 では 41us 程度となっており，10 分の 1 程度，短くなっている．これについては 6.3 節で考察する．

5.3 (3) TC0 と TC1 の送信タイミング差(300us)からのぶれ

LAN アナライザにて TC0 と TC1 の送信フレームを取得し，各フレームの送信タイミングの差(300us)に対するぶれを算出した．測定は 15 秒程度である．結果を表 10 に示す．

また，UDP の場合の，送信フレーム送信タイミングのぶれを，図 16 および図 17 に示す．TCP の場合はほぼ同様の形状であり，図を省略する．

評価の結果，UDP/TCP 共に offload 有効で ETF を用いた場合，TC0 と TC1 の送信タイミングは 10 ナノ秒オーダと判明した．特に最大と最小の差は 48ns となった．offload 有効で ETF を用いた場合は TC 間のぶれも抑えることが可能である．

表 10　TC0 と TC1 の送信タイミングぶれの評価結果

	offload 有効		offload 無効	
	UDP	TCP	UDP	TCP
平均 [ns]	0.3	1.9	-7426.4	-1570.9
最大 [ns]	24.0	24.0	12448.0	11464.0
最小 [ns]	-24.0	-16.0	-229872.0	-28776.0
最大 - 最小 [ns]	48.0	40.0	242320.0	40240.0
標準偏差	12.6	11.8	11528.7	829.1

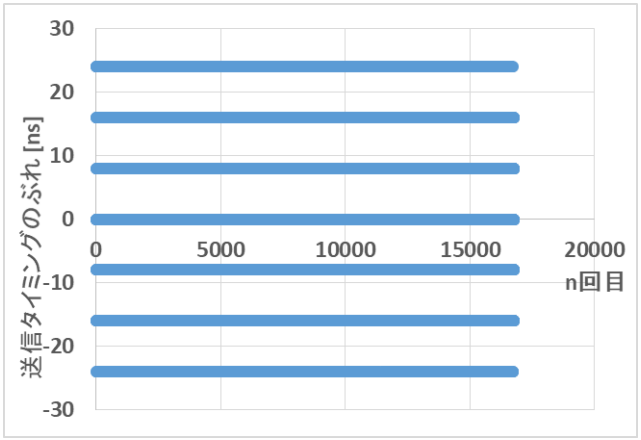

図 16　offload 有効(UDP)の送信結果 (TC1-TC0)

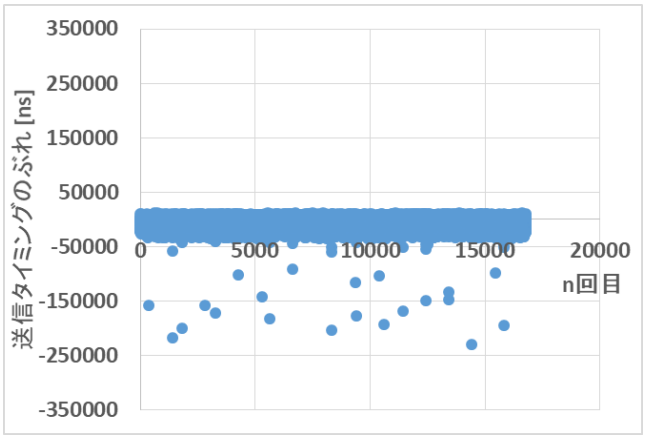

図 17　offload 無効(UDP)の送信結果 (TC1-TC0)

対して，UDP/TCP 共に offload 無効で ETF を用いた場合，マイクロ秒オーダで遅延が生じる．特に UDP の最大と最小の差は 250us 程度であり．特にマイナス方向に悪化がみられる．これについては 6.3 節で考察する．

Vol.2019-ARC-240 No.35
Vol.2019-SLDM-191 No.35
Vol.2019-EMB-53 No.35
2019/2/28

5.4 (4) TC2 の UDP/TCP 通信スループット

通信スループット評価ツール iperf3 におけるサーバ側の通信スループットを表 11 に示す.

iperf3 ベンチマークについて，機能的には動作しているものの，ETF/TAPRIO 無効時と比較して，スループット性能は悪化している．本件については 6.2 節で考察する.

表 11　通信スループット

offload 有効		offload 無効		(参考) 通常時 (ETF/TAPRIO 無効時)	
UDP	TCP	UDP	TCP	UDP	TCP
353 Mbps	361.5 Kbps	366 Mbps	1.36 Mbps	946 Mbps (※)	993 Mbps

(※) 並列数 20(-P 20)で実行時

6. 考察

ETF/TAPRIO について，次の観点で考察を行う.

(1) TC0 と TC1 の送信フレームに依存性がある場合
(2) TC2 の送信スループット
(3) offload 無効時の挙動

6.1 (1) TC0 と TC1 の送信フレームに依存性がある場合

今回, TC0 と TC1 の送信フレームには依存性がない前提で評価した．しかし，実際のシステムでは，複数の制御プロトコルを連携させ，TC0 で送信したフレームに対応した応答フレームの受信内容を利用して TC1 の送信フレームを生成することが考えられる(図 18).

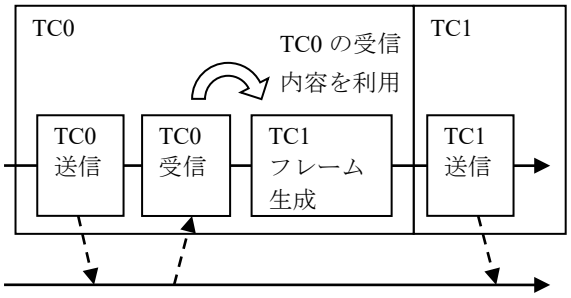

図 18　送信フレームの依存性

この場合，TC1 のフレーム生成に間に合うように，TC0 の長さを設定する等，タイミングの調整が必要となる.

6.2 (2) TC2 の送信スループット

iperf3 で計測した送信スループットについて，ETF/TAPRIO 無効時と比較して悪化していた．本件について，UDP/TCP それぞれで考察する.

6.2.1 UDP のスループット

ETF/TAPRIO 無効時の UDP スループットは 950Mbps 程度の性能となり，物理 I/F の限界である 1Gbps に近い性能となった．これに対し，ETF/TAPRIO 有効時は 350Mbps 程度となった.

この理由について，周期全体の長さ 1ms に対して，TC2 の長さは 400us である．すなわち，TC2 の帯域を最大限利用した場合でも最大 400Mbps となる．実際には，UDP ヘッダや送信フレーム間隔などのオーバヘッド要因により，350Mbps 程度の性能になったと考えられる.

6.2.2 TCP のスループット

ETF/TAPRIO 無効時の TCP スループットは 993Mbps 程度の性能となり, 物理 I/F 限界の 1Gbps に近い性能となった．これに対し，ETF/TAPRIO 有効時において，offload 有効時に 361.5Kbps, offload 無効時に 1.36Mbps と明らかに低い値となった．この結果について，数 Kbps の帯域で十分な小規模 Web サーバ等であれば問題とならないが，数 Mbps 超の帯域を要求する映像データ等で問題となる可能性が高い.

スループット低下の理由について，iperf3 の TCP の通信フレームを解析したところ，iperf3 開始の初期段階でターゲットボードからの送信フレームがドロップし，その後，輻輳処理により帯域を制限して通信している様子が観測された．特に，offload 有効時では，より厳密にフレーム破棄を判断しているため，スループット性能が悪化していた.

今回の ETF は, すでに超過したタイミングでフレームを送信しようとした場合，カーネル内部でフレームを破棄する[9]．これは，次のような状況で発生しうる.

① TAPRIO により iperf3 フレームの送信タイミングが TC2 内に収まるように調整
② カーネル内部で割り込み処理などのオーバヘッド処理が発生
③ ETF によりフレームをドライバのバッファに設定する際，②にて，フレームの送信タイミングが超過(expired)した場合，ETF は送信フレームを破棄
④ フレームの破棄により，TCP の輻輳処理が発生

本来，iperf3 のフレームはベストエフォート型のフレームであり，TAPRIO が設定した送信タイミングを超過しても，次のタイミングでフレームを送信すべきである．しかし，ETF は，送信フレームについて，ベストエフォート型アプリのものであるか，あるいは，ユーザが意図的に送信タイミングを指定したものであるかを判断していない．すなわち ETF は一律でフレームの送信タイミングを判断し，超過していれば破棄してしまう.

これらの課題の改善策として，TAPRIO にて，フレームに対し，ベストエフォート型のラベリングを行い，ETF にて，ベストエフォート型であれば，フレームを破棄するのではなく，次のタイミングで送信するように送信タイミングを変更する．これにより，ベストエフォート型のフレームが破棄されることはなくなる．こうした，送信フレーム

Vol.2019-ARC-240 No.35
Vol.2019-SLDM-191 No.35
Vol.2019-EMB-53 No.35
2019/2/28

の再スケジュール機能の検討が今後の課題である．

6.3 (3) offload 無効時の挙動

5.1 節，5.2 節，および，5.3 節の評価において，offload 無効時，かつ，iperf3 の UDP 動作時に，TC0 の定周期送信タイミングが大幅に悪化する現象が確認された．

この理由について，offload 無効時の各 TC におけるフレーム送信状況を確認したところ，図 19 に示すように，TC2 に割り当てられた iperf3 用の送信フレームが，最大で 200us 程度，TC0 の時間を侵食しており，これにより TC0 用の ETF フレームの送信タイミングが遅延している状況が確認できた．

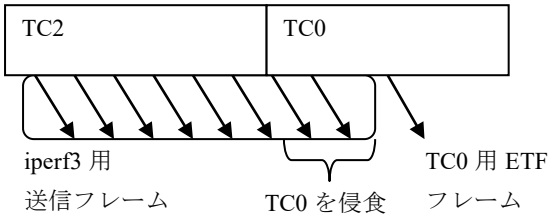

図 19　offload 無効時

offload 無効時は OS のタイマを使用してタイミングが調整される．そのため，offload 有効時と比較して，厳密なタイミング制御が難しく，本来 TC2 に割り当てられた送信フレームが TC0 のタイミングで送信されてしまったものと考えられる．

なお，TC1 について，TC0 では単一の ETF 用フレームを送信するだけであり，TC1 を侵食することはない．そのため，TC1 の送信タイミングのぶれは TC0 ほど悪化していない．以上から，UDP においては TC0 と TC1 の送信タイミングに 200us 程度の差が出ている．

本件に関連して，5.3 節で送信タイミング差のぶれについても同様に，TC2 に侵食されて TC0 の送信タイミングが 200us 程度遅れるのに対し，TC1 の送信タイミングは 20us 程度のため，マイナス方向にぶれが大きくなっている．

以上の結果から，offload 無効で ETF/TAPIRO を利用しても TSN 通信で求められるマイクロ秒オーダの精度は満足できないと判明した．

最後に，表 8 の offload 無効時の最大と最小の差について，UDP の 424us に対し，TCP では 8us であり，性能悪化は限定的であった．この理由について，6.2.2 節に記載した通り，TCP ではフレーム破棄による輻輳処理が発生しており TC2 の送信帯域を最大限まで使用しておらず，TC0 への侵食が発生していないためである．

7. おわりに

今回，組込み Linux 上で TSN 通信の実現性を確認するため，Linux に追加された TAPRIO と ETF を併用した環境にて，定周期送信アプリとベストエフォートアプリが共存可能となるかを確認した．

結果，ベストエフォート型のアプリを動作させつつ，定周期送信アプリの送信タイミングのずれが 50ns 以下となった．これにより，定周期送信アプリとベストエフォート型アプリが共存可能となることを確認した．一般的な TSN のユースケースではマイクロ秒の精度が求められるが，今回の環境はこれを満足可能と判明した．すなわち，組込み機器であっても，H/W サポート機能を持つ NIC を利用することで，TSN 通信を実現可能という見込みを得た．

ただし，ベストエフォート型アプリの送信フレームについて，TAPRIO が設定した送信タイミングを超過した場合，ETF がフレームを破棄していることを確認した．特に，通信プロトコルとして TCP を使用した場合に，フレーム破棄により輻輳処理が発生し，ベストエフォート向けの帯域を有効に活用できない状況を観測した．ベストエフォート型アプリとして映像データ等を扱うような，数 Mbps 超の帯域を要求する場合には問題となる可能性が高い．本課題の改善策として，ベストエフォート型アプリの送信フレームの場合，送信タイミングを超過しても次の送信タイミングで送信可能とするような，送信フレームの再スケジュール機能の検討が今後の課題である．

今後も引き続き，通信帯域を効率的に利用しつつ，ベストエフォート型アプリと定周期送信アプリを共存可能とするように，組込み Linux 向け TSN 対応開発を行う．

参考文献

[1] 出原章雄，水口武尚．組込み Linux の Real-Time Ethernet 性能評価．研究報告 組込みシステム(EMB),2019-EMB-50(1),1-7 (2019-03-10) , 2188-868X (ETNET2019)
[2] “linuxptp”. http://linuxptp.sourceforge.net/, (参照 2019-12-17)
[3] “gPTPd”. https://github.com/AVnu/gptp, (参照 2019-12-17)
[4] IEEE Standard for Local and metropolitan area networks—Bridges and Bridged Networks (IEEE Std 802.1Q-2018)
[5] “The first half of the 4.19 merge windows”. https://lwn.net/Articles/762566/, (参照 2019-12-17)
[6] “KernelNewbies: Linux_5.3”. https://kernelnewbies.org/Linux_5.3, (参照 2019-12-17)
[7] “iperf3”. https://iperf.fr/, (参照 2019-12-17)
[8] “[TSN] Scheduled Tx Tools - Examples and Helpers for testing SO_TXTIME, and the etf and taprio qdiscs”. https://gist.github.com/jeez/bd3afeff081ba64a695008dd8215866f, (参照 2019-12-17)
[9] "etf: Drop all expired packets", https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/commit/?h=v5.3.11&id=37342bdaf5b363cf2e1bd170ce7d1de34ecf57e7 (参照 2019-12-17)