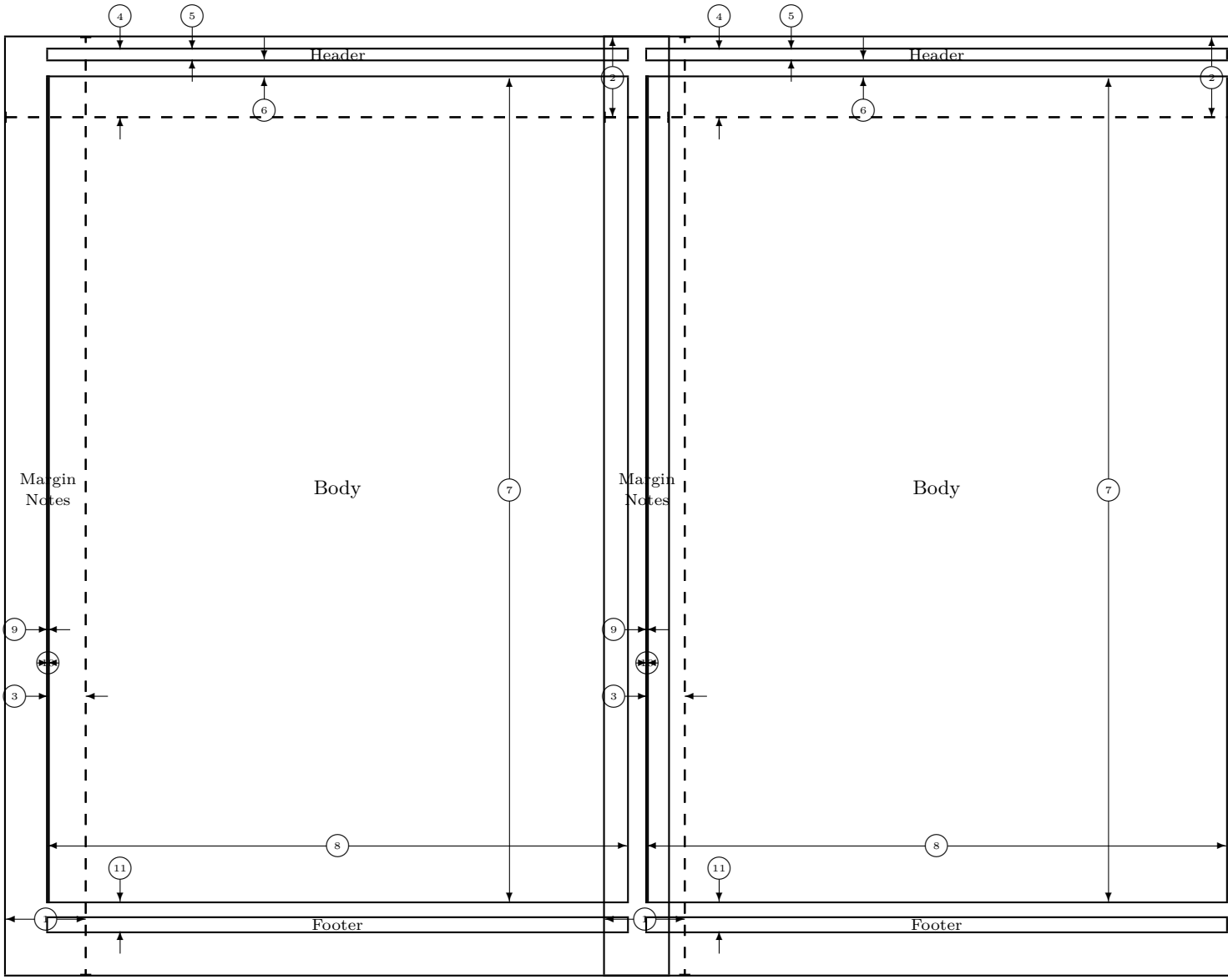


```
 1  one inch + \hoffset
 2  one inch + \voffset
 3  \evensidemargin = -34pt
 4  \topmargin = -61pt
 5  \headheight = 9pt
 6  \headsep = 16pt
 7  \textheight = 743pt
 8  \textwidth = 522pt
 9  \marginparsep = 0pt
10  \marginparwidth = 0pt
11  \footskip = 27pt
    \marginparpush = 3pt (not shown)
    \hoffset = 0pt
    \voffset = 0pt
    \paperwidth = 597pt
    \paperheight = 845pt
```

```
 1  one inch + \hoffset
 2  one inch + \voffset
 3  \evensidemargin = -34pt
 4  \topmargin = -61pt
 5  \headheight = 9pt
 6  \headsep = 16pt
 7  \textheight = 743pt
 8  \textwidth = 522pt
 9  \marginparsep = 0pt
10  \marginparwidth = 0pt
11  \footskip = 27pt
    \marginparpush = 3pt (not shown)
    \hoffset = 0pt
    \voffset = 0pt
    \paperwidth = 597pt
    \paperheight = 845pt
```

フレッツの閉域 IPv6 網で EVPN/VXLAN

浅間 正和

概要

参考文献 [1] では **a.** OSS の EVPN 実装である FRRouting[2] を IPv6 による EVPN 対応に改修し **b.** Linux カーネルを MTU 1500 の Ethernet フレームをフラグメントして送信できるように改修し **c.** VXLAN の処理を高速に行える DPDK[3] アプリケーションを開発することで NTT 東西社の提供するフレッツサービスの閉域 IPv6 網上で EVPN/VXLAN による L2VPN を構築する方法を紹介しました。本文書ではこのうち **c.** の DPDK アプリケーションの詳細について説明します。

1 gdp: DPDK を用いたデータプレーン

gdp は DPDK を用いて書かれたデータプレーン実装です。図 1 に gdp の動作概要を示します。gdp は起動時に物理 NIC を Linux カーネルから切り離し DPDK で直接操作できる状態にし、それに対応する仮想 NIC(tap デバイス) を作成します (図 1 の①)。仮想 NIC へは物理 NIC に対して行うのと同じように IP アドレスを設定することが出来ます (図 1 の②)。Linux カーネルは NIC に IP アドレスが設定された際にそれを Netlink でアプリケーションに通知する仕組みが用意されており、gdp は Netlink により仮想 NIC に IP アドレスが設定されたことを知りその IP アドレスを覚えます (図 1 の③)。Linux カーネルは IP アドレスの設定の他にもルーティングテーブルやネイバーテーブル等の変更時にも同様に Netlink でそれを通知します。gdp はそれらの情報を元に Linux カーネルの設定と状態を覚え、物理 NIC に届いたパケットを Linux カーネルが処理するのと同じように Linux カーネルをバイパスして処理します (図 1 の④)。

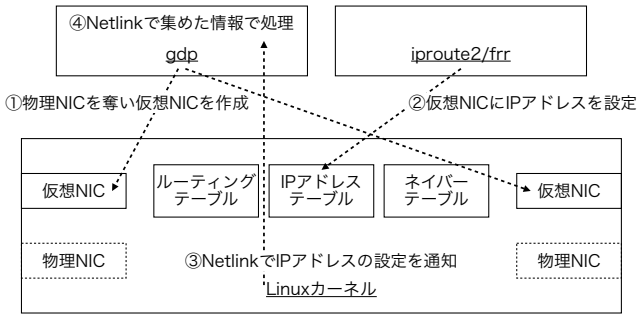

図 1 gdp の動作概要

gdp が受信したパケットを gdp が自身の持つ情報だけでは送信できないようなケース (例: 宛先が自分自身のパケットやネイバー情報がまだ存在せず転送できない等) の場合は仮想 NIC に対応する tap デバイスに対してそのパケットを書き込むことで Linux カーネルに処理を依頼します。逆に Linux カーネルが仮想 NIC からパケットを送信しようとした場合 gdp は tap デバイスからパケットを読み込みそれを物理 NIC から送信します。

gdp は自身で処理できなかったパケットを tap 経由で Linux カーネルへ処理を依頼するため、パケットのオリジナルデータは復元可能な状態で保持した上で書き換えたい部分 (主にヘッダ部分) を書き換える必要があります。そのため gdp ではオリジナルパケットを表すパケットバッファ (パケットを表す構造体) から書き換えたい部分をコピーしたパケットバッファを用意しそこを書き換え、そのパケットバッファにデータの先頭位置 (data_off) をずらしたオリジナルパケットをつなげることで書き換え後のパケットを表現します (図 2)。もし途中で gdp のもつ情報だけでは処理することが不可能とわかった場合は覚えておいたオリジナルの data_off を復元し前段のパケットバッファを取り除きオリジナルパケットを tap デバイスへ書き込み Linux カーネルに処理を依頼します。

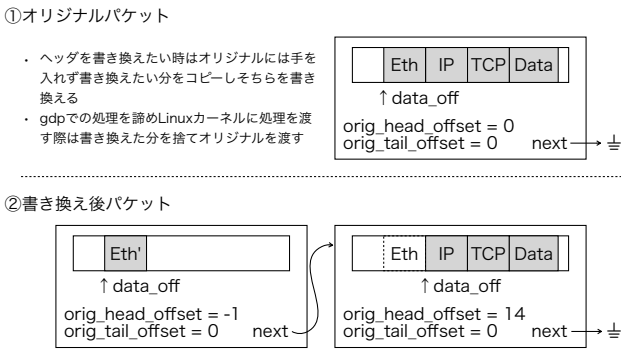

図 2 gdp のバケット処理

2 ベンチマーク試験

ベンチマーク試験は Cisco T-Rex[4] を用いて行いました。Tester/DUT のハードウェア構成は 表 1 の通りです。Tester から 30 秒間試験パケットを送信し送信した全てのパケットを Tester が受信できる転送レートを二分探索で求めるという方法をとっています。

udp$X$b は Ethernet ヘッダと FCS も含めて X バイトの UDP データグラムを流すというものです。imix は udp64b と udp594b と udp1518b を 7 : 4 : 1 の割合で流すというものです。

試験に用いたスクリプトや試験結果の生データなどの詳細は参考文献 [1] を参照してください。

CPU	Intel N100 @ 3.4GHz 4C/4T
メモリ	Crucial DDR5 4800MHz 32GB
NIC	Intel 82599ES (5GT/s ×4) ×2
SFP	FS.com SFP-10GSR-85

表 1 Tester/DUT ハードウェア仕様

[pps]		imix	udp64b	udp128b	udp256b	udp512b	udp1024b	udp1280b	udp1518b
encap	linux	497,261	539,012	536,857	535,822	535,705	539,947	529,881	265,178
	gdp 1core	1,738,388	1,962,340	1,934,296	1,887,503	1,690,000	1,122,151	912,464	734,424
	gdp 2core	1,728,429	1,917,257	1,943,046	1,896,103	1,686,451	1,122,151	912,409	734,424
decap	linux	599,549	673,765	678,596	676,282	655,849	648,636	651,172	194,317
	gdp 1core	1,799,114	2,101,551	2,089,684	2,067,434	1,926,359	1,123,315	913,578	735,112
	gdp 2core	1,757,358	2,089,166	2,054,337	2,051,337	1,913,305	1,123,315	913,467	735,254
both	linux	393,596	454,791	447,614	448,503	446,104	441,395	438,328	157,744
	gdp 1core	823,326	979,436	950,525	920,378	833,277	682,399	563,628	400,448
	gdp 2core	1,458,686	1,838,486	1,797,457	1,693,240	1,201,820	681,646	563,740	399,531
[kbps]		imix	udp64b	udp128b	udp256b	udp512b	udp1024b	udp1280b	udp1518b
encap	linux	1,518,967	275,974	549,742	1,097,363	2,194,248	4,423,250	5,425,981	3,220,326
	gdp 1core	5,310,196	1,004,718	1,980,720	3,865,607	6,922,238	9,192,662	9,343,636	8,918,846
	gdp 2core	5,279,775	981,636	1,989,679	3,883,219	6,907,703	9,192,662	9,343,066	8,918,846
decap	linux	1,831,422	344,968	694,882	1,385,025	2,686,358	5,313,622	6,667,997	2,359,787
	gdp 1core	5,495,694	1,075,994	2,139,837	4,234,104	7,890,365	9,202,199	9,355,041	8,927,201
	gdp 2core	5,368,142	1,069,653	2,103,641	4,201,138	7,836,898	9,202,199	9,353,901	8,928,930
both	linux	1,202,304	232,853	458,357	918,533	1,827,243	3,615,911	4,488,481	1,915,640
	gdp 1core	2,514,988	501,471	973,337	1,884,935	3,413,102	5,590,215	5,771,556	4,863,044
	gdp 2core	4,455,800	941,305	1,840,596	3,467,757	4,922,654	5,584,044	5,772,696	4,851,905

表 2　1 秒間あたりに処理可能なパケット数とビット数

図 3 は 1 秒間あたりに転送可能なパケット数を表す pps 性能のうち imix と udp64b のみを抽出したグラフです。Linux カーネルで処理する場合に対し gdp ではおおよそ 3 倍程度の性能となっています。gdp は、片方向では 1 CPU コアと 2 CPU コアで違いがありませんが、双方向になった際に 2 CPU コアのほうが 2 倍程度性能がよくなっています。これは単純に 1 CPU コアでは 1 つの CPU コアが 2 ポート分のエンキャップとデキャップを両方処理しなければならないものが 2 CPU コアでは各 CPU コアはエンキャップかデキャップのどちらか一方の処理のみを行えば良い為です。

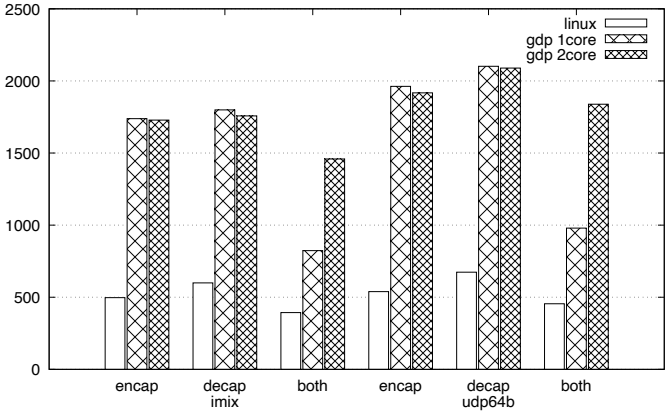

図 3　1 秒間あたりに転送可能なパケット数 [kpps]

図 4 は 1 秒間あたりに転送可能なビット数を表す bps 性能のうち imix と udp1518b のみを抽出したグラフです。こちらも Linux カーネルで処理する場合に対し gdp ではおおよそ 3 倍程度の性能となっています。ただ udp64b や imix と異なり udp1518b では 1 CPU コアと 2 CPU コアの違いはありません。これはショートパケットと異なりロングパケットでは CPU 処理性能よりも NIC の帯域の方が先にボトルネックとなる為です。

udp1518b は、エンキャップ時はフラグメント処理が、デキャップ時はリアセンブル処理が、それぞれ発生します。これらの処理は比較的重い処理となるため Linux カーネルではフラグメントとリアセンブルの発生しない udp1280b の方が転送性能は良くなっています。gdp でも転送性能的には udp1518b よりも udp1280b の方が良くなっていますが、これはフラグメントする際に IPv6 ヘッダと IPv6 フラグメント拡張ヘッダ等のオーバーヘッドが生ずるためです。詳しくは表 2 を参照してください。

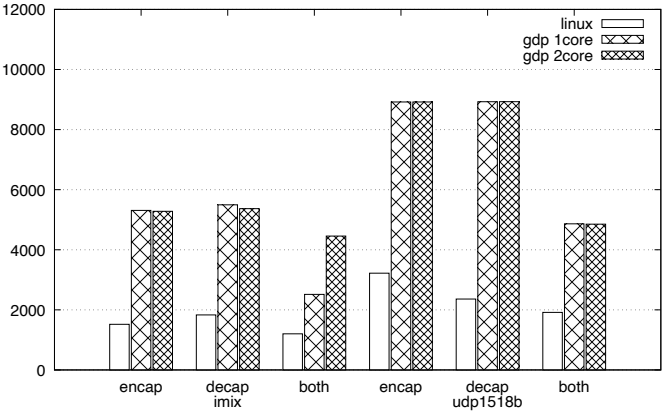

図 4　1 秒間あたりに転送可能なビット数 [Mbps]

3　まとめ

gdp を用いることで Linux カーネルに対しておおよそ 3 倍程度の転送処理を行うことが出来るようになります。1 CPU コアのみでも十分性能が出せますが、 2 CPU コアを用いることで双方向通信の性能を向上させることも出来ます。

gdp は Netlink を用いることで Linux カーネルの設定情報を収集し Linux カーネルが行う処理と同じことを行おうとするので透過的に利用することが出来ます。インターフェース名を物理 NIC から仮想 NIC に変えるだけでよく、一般に使われている iproute2 コマンドや FRRouting などのソフトウェアもそのまま利用できます。

gdp で Linux カーネルをバイパスできる機能はまだ限られていますが今後少しずつ機能追加する予定です。

参考文献

[1] 浅間正和. フレッツの閉域 IPv6 網で EVPN/VXLAN. https://github.com/m-asama/evpnvxlan6/.

[2] FRRouting. https://frrouting.org/.

[3] DPDK. https://www.dpdk.org/.

[4] TRex. https://trex-tgn.cisco.com/.