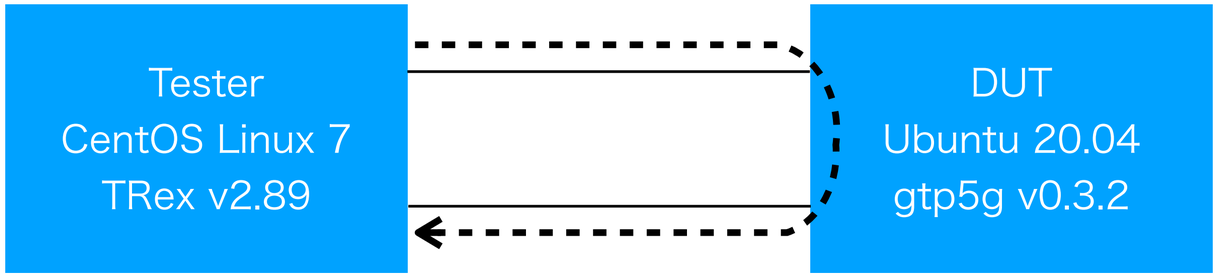
UPFの下周り性能ネタ(gtp5g他)

浅間正和 (有限会社銀座堂) @ OMNI Meetup #4 (2021/10/12)

試験構成

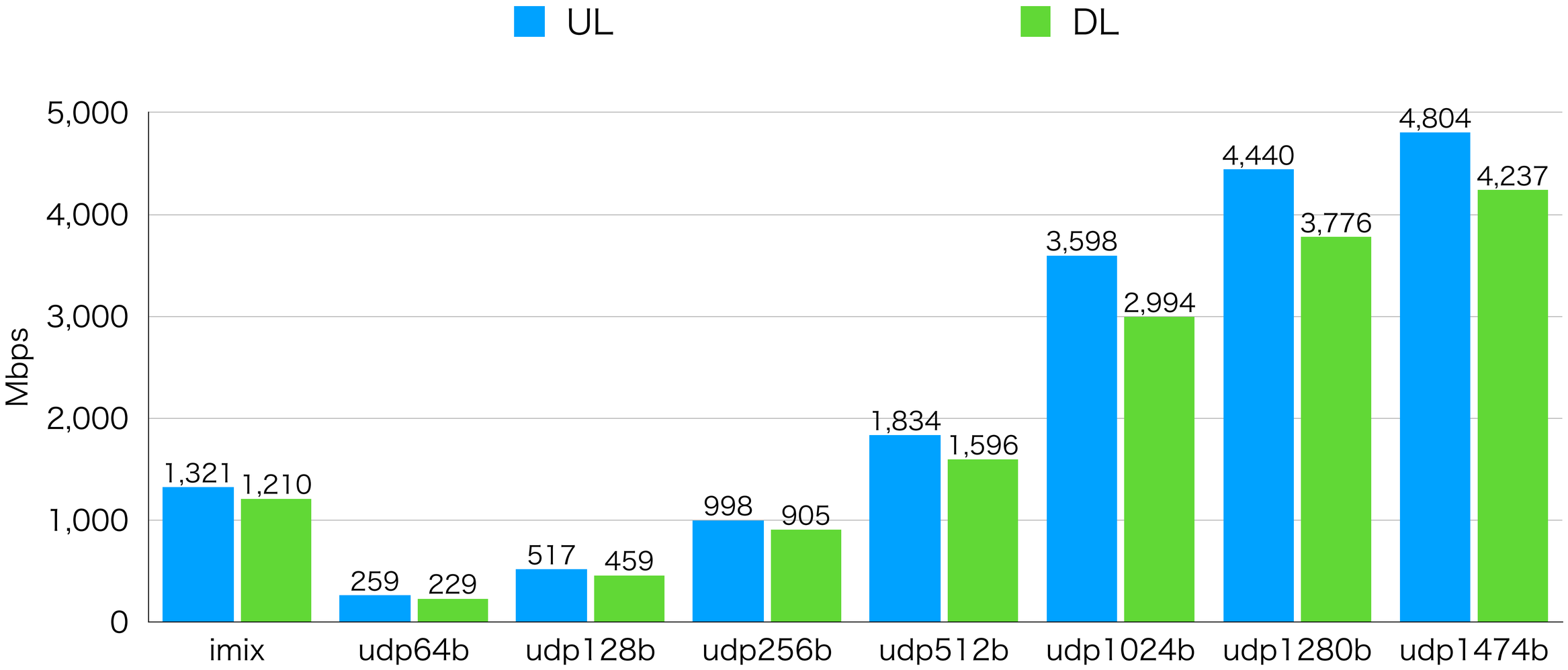
```
ip addr add 172.16.1.1/24 dev ens1f0
ip addr add 10.1.1.1/24 dev ens1f1
ip link set ens1f0 up
ip link set ens1f1 up
gtp5g-link add gtp5gtest &
gtp5g-tunnel add qer gtp5gtest 1 --qer-id 1 --qfi 1
gtp5g-tunnel add far gtp5gtest 1 --action 2
gtp5g-tunnel add far gtp5gtest 2 --action 2 --hdr-creation 0 87 172.16.1.2 2152
gtp5g-tunnel add pdr gtp5gtest 1 --pcd 1 --hdr-rm 0 --ue-ipv4 192.168.1.11 --f-teid 78 172.16.1.1 --far-id 1 --qer-id 1
gtp5g-tunnel add pdr gtp5gtest 2 --pcd 2 --ue-ipv4 192.168.1.11 --far-id 2 --qer-id 1
ip route add 192.168.1.0/24 dev gtp5gtest
ip neigh add 172.16.1.2 lladdr 0c:c4:7a:b7:59:3a dev ens1f0
ip neigh add 10.1.1.11 lladdr 0c:c4:7a:b7:59:3b dev ens1f1
```



Testerは送信ポートからパケットを送信し
受信ポートで受信した数を確認する
例えば最初に10Gbps流してロストがあったら
5Gbpsに落として全パケット受信できたら7.5Gbpsにして
というふうにパケットをロストしない帯域幅を調べる
1試行30秒間とし16~20試行繰り返す

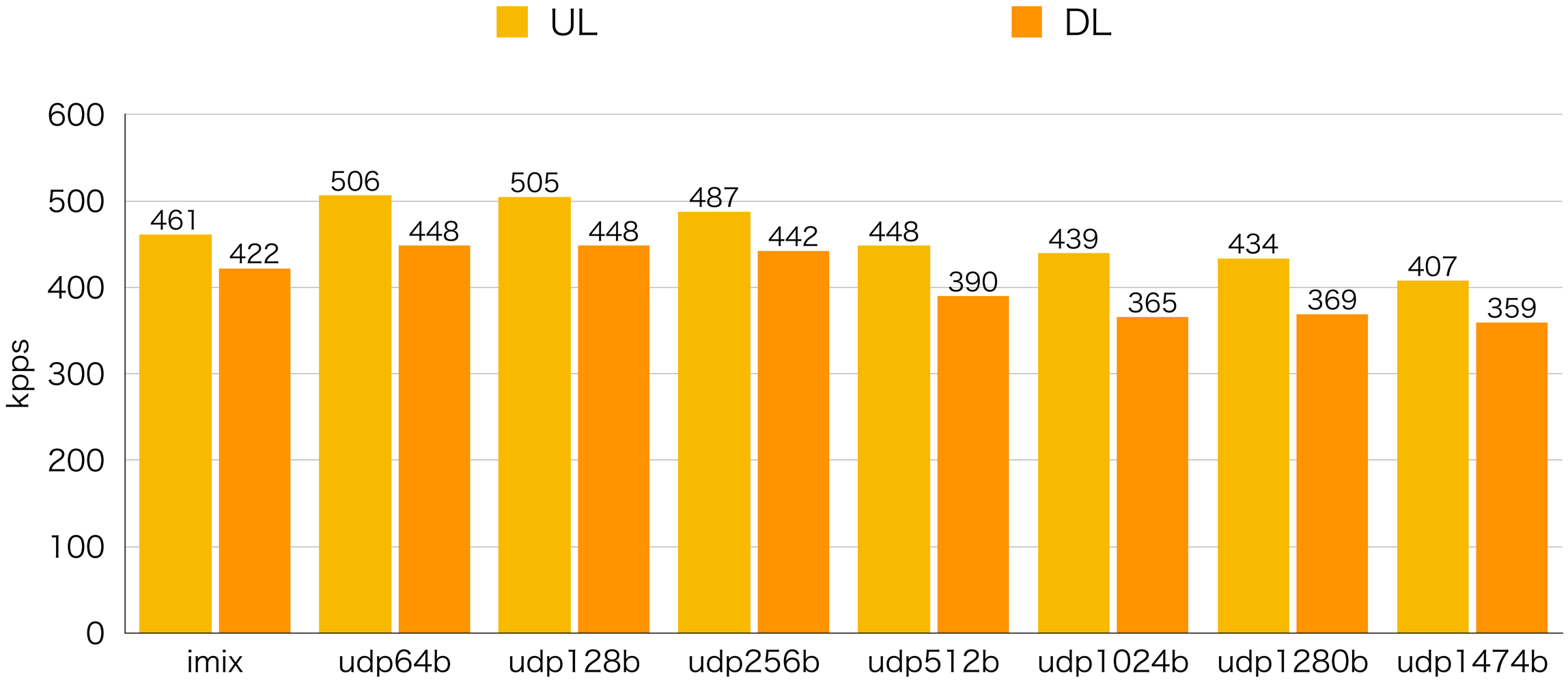
ハードウェア構成:
Intel Xeon E5620 2.40GHz 4C/8T
Intel 5520 chipset
DDR3 1066 MHz 48GB
Intel 82599ES 10-Gigabit SFI/SFP+ x 2

ベンチマーク結果: gtp5g



※ bps には Ethernet ヘッダ 14 byte を含む

ベンチマーク結果: gtp5g



nff-go-upf

- NFF-Go(後述) で UPF を実装したらどれくらいの性能が出るのか気になったのでとりあえず実装してみた
- ただし
  - PFCP は未実装(右のような YAML で設定を書いて読み込む)
  - ARP やルーティングも未実装(右のように宛先 MAC アドレスをハードコード)
  - Router on a stick 構成のみサポート
  - SDF Filter 未実装(どうせやるならちゃんと IPFilterRule 型のパーサ書きたい)
- ただし
  - QER はキッチリやる

60 lines (60 sloc) | 1.25 KB

```yaml
global:
  cpuList: 0-3
  local:
    port: 0
    address: xx:xx:xx:xx:xx:xx
    teAddress: 172.16.1.1
  n3n9:
    vlanId: 11
    address: xx:xx:xx:xx:xx:xx
  n6:
    vlanId: 12
    address: xx:xx:xx:xx:xx:xx
sessions:
  - fseid:
      seid: 1
      address: 127.0.0.8
    pdrs:
      - pdrid: 1
        precedence: 255
        pdi:
          sourceInterface: access
          fteid:
            teid: 1
            address: 172.16.1.1
          networkInstance: ""
          ueIpAddress: 192.168.0.1
        outerHeaderRemoval: true
        farid: 1
        qerids:
          - 1
      - pdrid: 2
        precedence: 255
        pdi:
          sourceInterface: core
```

https://github.com/m-asama/nff-go-upf/blob/main/config.yaml.sample から引用

NFF-Go

- Intel さんがオープンソースで公開している Go 言語のための Network Function Framework
- 内部で(同じく Intel さんがオープンソースで公開している) DPDK を用いている(ので ARP やルーティングなどは全て自前で実装する必要がある)
- パケットの流れを“フロー”として扱う
- フローを分割したり(Separator)処理を施したり(Handler)集約したり(Merger)破棄したり(Stopper)といったことができるようになっている
- フローをバッファする方法はないっぽい？

```go
func main() {
	// Initialize NFF-GO library to use 8 cores max.
	config := flow.Config{
		CPUCoresNumber: 8,
	}
	flow.CheckFatal(flow.SystemInit(&config))

	// Get filtering rules from access control file.
	L3Rules, err := packet.GetL3ACLFromTextTable("Firewall.conf")
	flow.CheckFatal(err)

	// Receive packets from zero port. Receive queue will be added automatically.
	inputFlow, err := flow.SetReceiver(uint8(0))
	flow.CheckFatal(err)

	// Separate packet flow based on ACL.
	rejectFlow, err := flow.SetSeparator(inputFlow, L3Separator, nil)
	flow.CheckFatal(err)

	// Drop rejected packets.
	flow.CheckFatal(flow.SetStopper(rejectFlow))

	// Send accepted packets to first port. Send queue will be added automatically.
	flow.CheckFatal(flow.SetSender(inputFlow, uint8(1)))

	// Begin to process packets.
	flow.CheckFatal(flow.SystemStart())
}

// User defined function for separating packets
func L3Separator(currentPacket *packet.Packet, context flow.UserContext) bool {
	currentPacket.ParseL4()
	// Return whether packet is accepted or not. Based on ACL rules.
	return currentPacket.L3ACLPermit(L3Rules)
}
```

https://github.com/intel-go/nff-go から引用

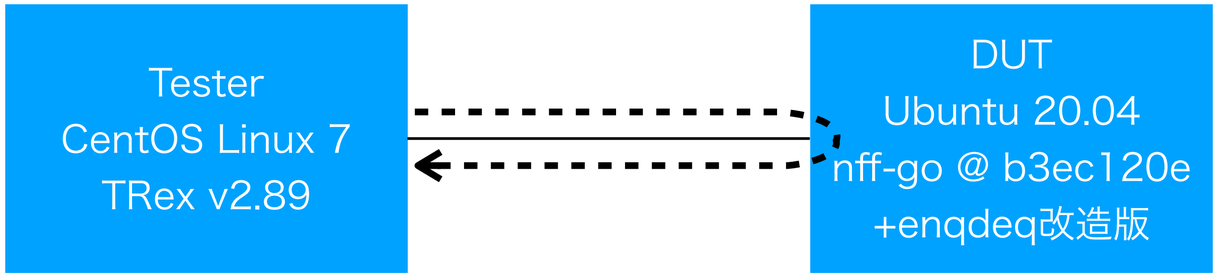
NFF-Go へバッファ機能追加

- NFF-Go をいじってフローをバッファする機能を実装
- 利用者側(例: nff-go-upf)は
  - バッファするか否かを判定しバッファする場合は内部でそのパケットをバッファする関数 enqf を定義
    - バッファしなかった場合は即座に破棄
  - バッファしたパケットに送信可能なものがあるかを判定しあった場合は次の処理に回す関数 deqf を定義
    - 取り出して破棄するケースもある
- を定義し設定することでそれらがループで回される

```
2011        for {
2012                select {
2013                case <-stopper[0]:
2014                        // It is time to close this clone
2015                        stopper[1] <- 1
2016                        return
2017                default:
2018                        for {
2019                                bufOut[0] = 0
2020                                deqf(&bufOut[0], &deqed)
2021                                if bufOut[0] != 0 {
2022                                        if !deqed {
2023                                                low.DirectStop(1, bufOut)
2024                                        } else {
2025                                                safeEnqueue(OUT[0], bufOut, 1)
2026                                        }
2027                                } else {
2028                                        break
2029                                }
2030                        }
2031                        for q := int32(0); q < inIndex[0]; q++ {
2032                                n := IN[q].DequeueBurst(bufIn, 1)
2033                                if n == 0 {
2034                                        continue
2035                                }
2036                                enqf(bufIn[0], &enqed)
2037                                if !enqed {
2038                                        low.DirectStop(1, bufIn)
2039                                }
2040                        }
2041                }
2042        }
2043  }
2044
2045  // This function tries to write elements to input ring. However
2046  // if this ring can't get these elements they will be placed
```

https://github.com/m-asama/nff-go/blob/enqdeq2/flow/flow.go から引用

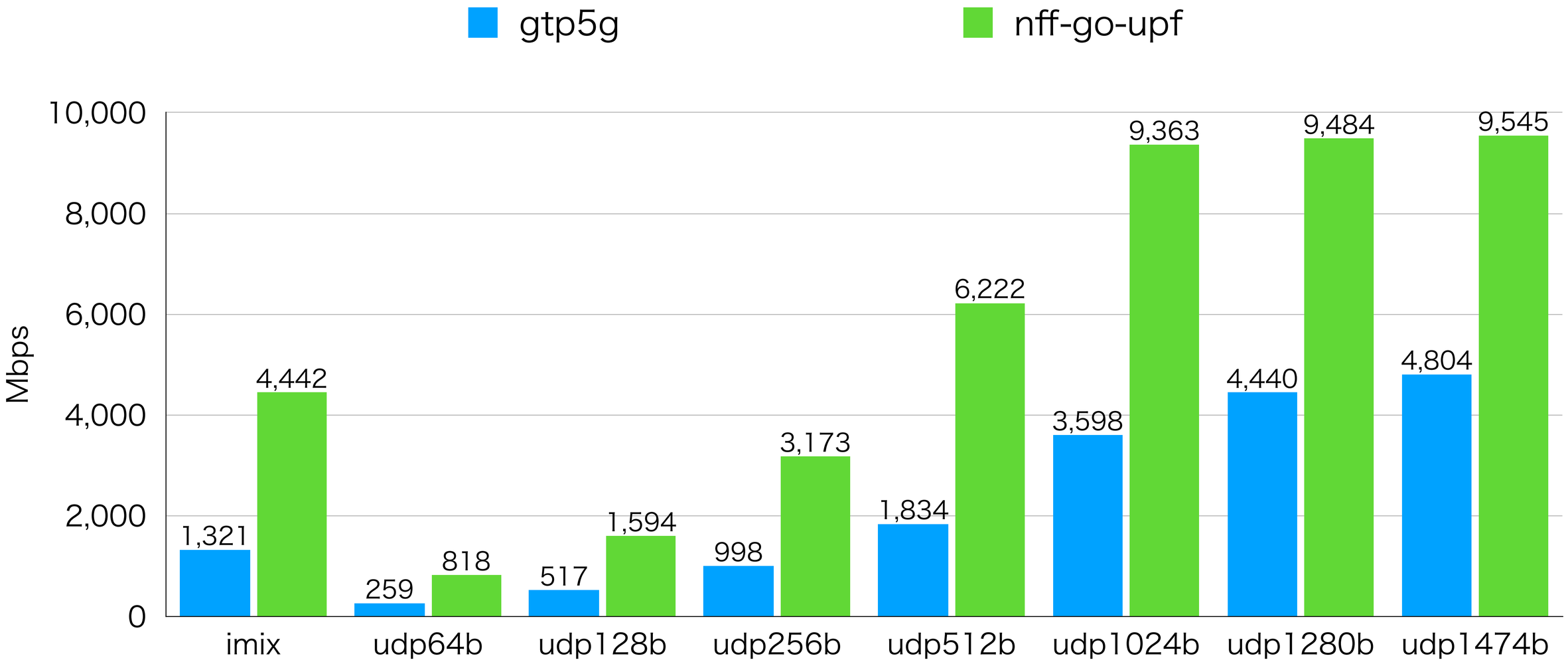
試験構成



Testerは送信ポートからパケットを送信し
受信ポートで受信した数を確認する
例えば最初に10Gbps流してロストがあったら
5Gbpsに落として全パケット受信できたら7.5Gbpsにして
というふうにパケットをロストしない帯域幅を調べる
1試行30秒間とし16~20試行繰り返す

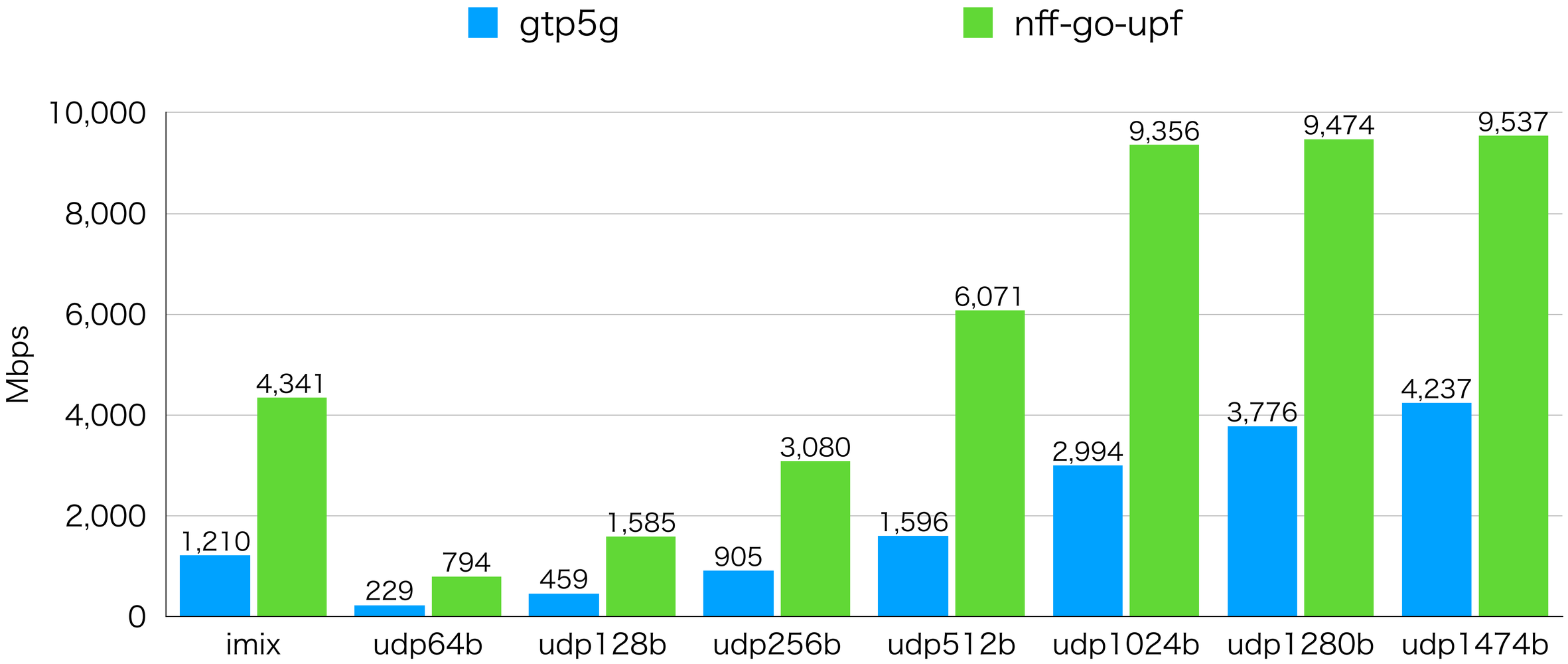
ハードウェア構成:
Intel Xeon E5620 2.40GHz 4C/8T
Intel 5520 chipset
DDR3 1066 MHz 48GB
Intel 82599ES 10-Gigabit SFI/SFP+ x 2

ベンチマーク結果: UL



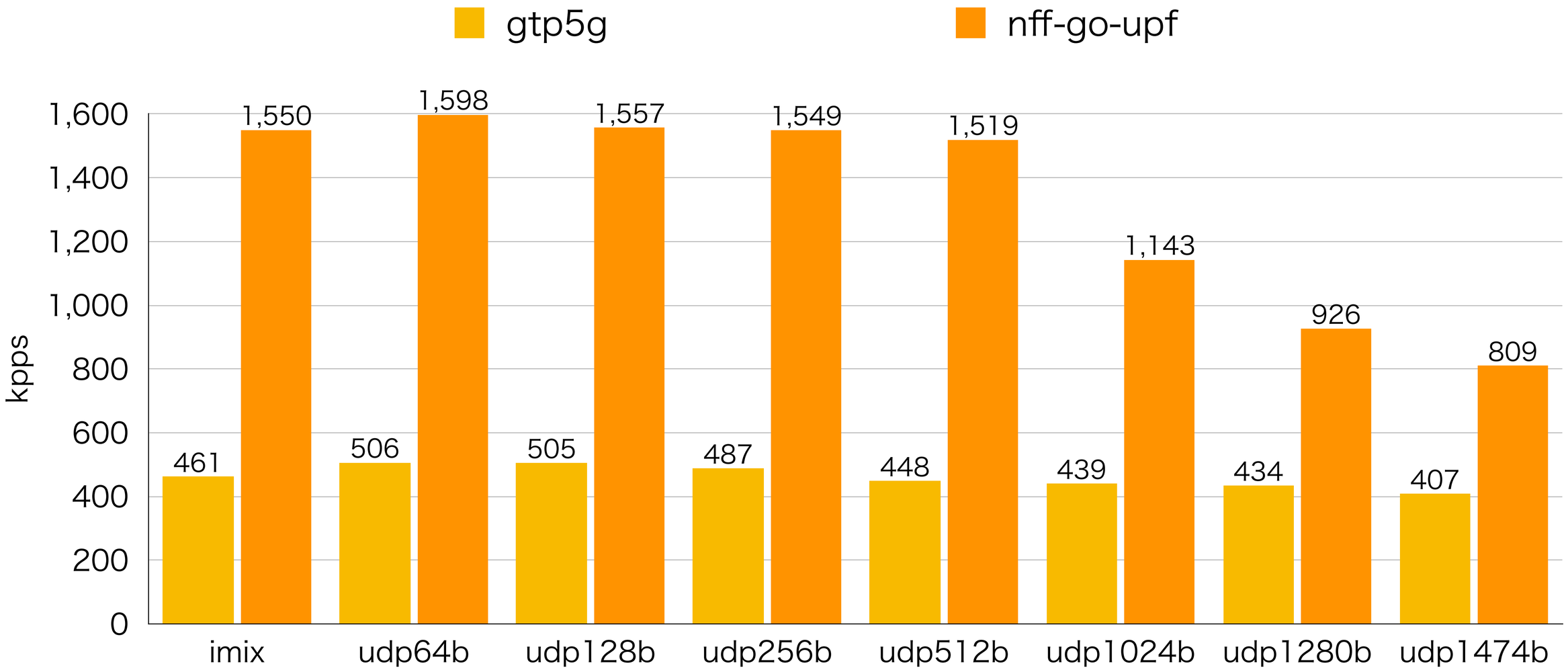
※ bps 計算時の Ethernet ヘッダは 14 byte で換算

ベンチマーク結果: DL

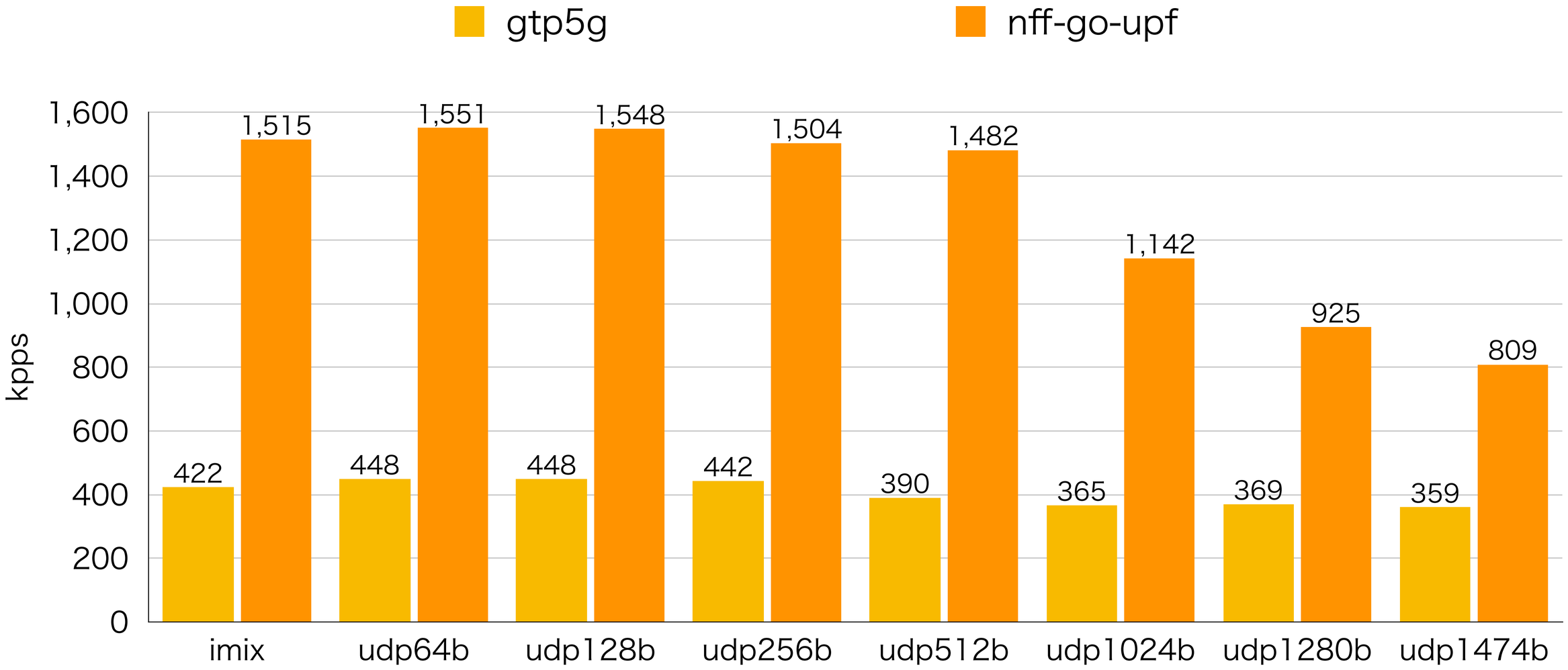


※ bps 計算時の Ethernet ヘッダは 14 byte で換算

ベンチマーク結果: UL


gtp5g
nff-go-upf
kpps
1,600
1,400
1,200
1,000
800
600
400
200
0
1,550
1,598
1,557
1,549
1,519
1,143
926
809
461
506
505
487
448
439
434
407
imix
udp64b
udp128b
udp256b
udp512b
udp1024b
udp1280b
udp1474b

ベンチマーク結果: DL


gtp5g
nff-go-upf
kpps
1,600
1,400
1,200
1,000
800
600
400
200
0
422
1,515
448
1,551
448
1,548
442
1,504
390
1,482
365
1,142
369
925
359
809
imix
udp64b
udp128b
udp256b
udp512b
udp1024b
udp1280b
udp1474b

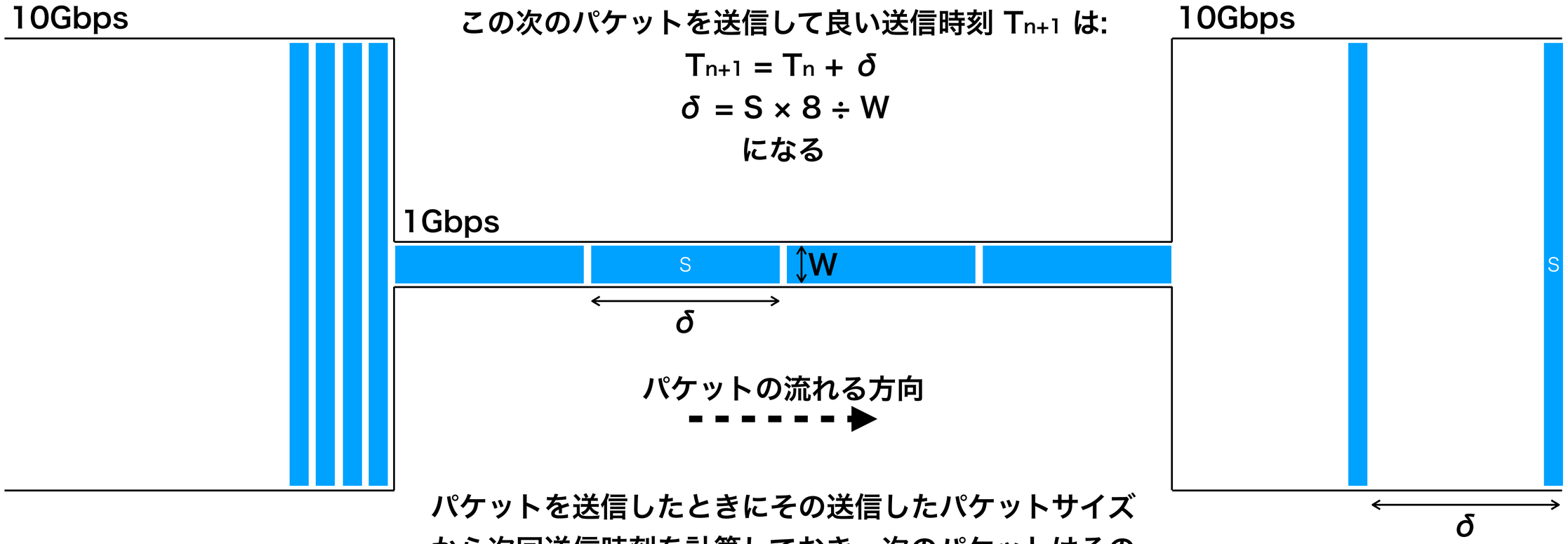
絞り方

帯域幅が W[bit/s]、パケットサイズが S[byte] で
このパケットの送信時刻を T_n としたとき
この次のパケットを送信して良い送信時刻 T_{n+1} は:

$$T_{n+1} = T_n + \delta$$

$$\delta = S \times 8 \div W$$

になる

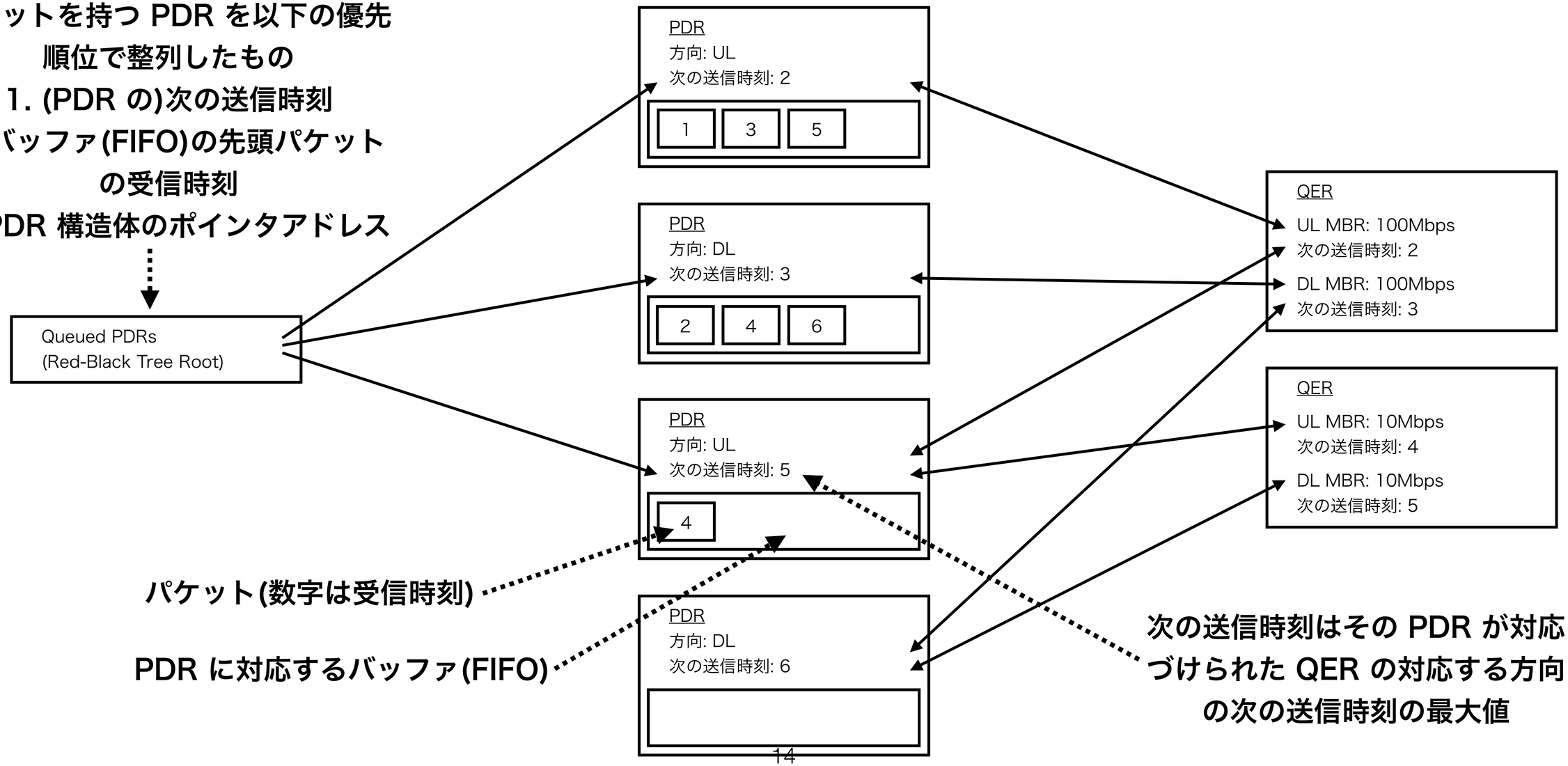


パケットを送信したときにその送信したパケットサイズ
から次回送信時刻を計算しておき、次のパケットはその
時刻まで遅延させてから送信するようにすれば良い？

バッファの仕方

バッファ(FIFO) に少なくとも1つパケットを持つ PDR を以下の優先順位で整列したもの

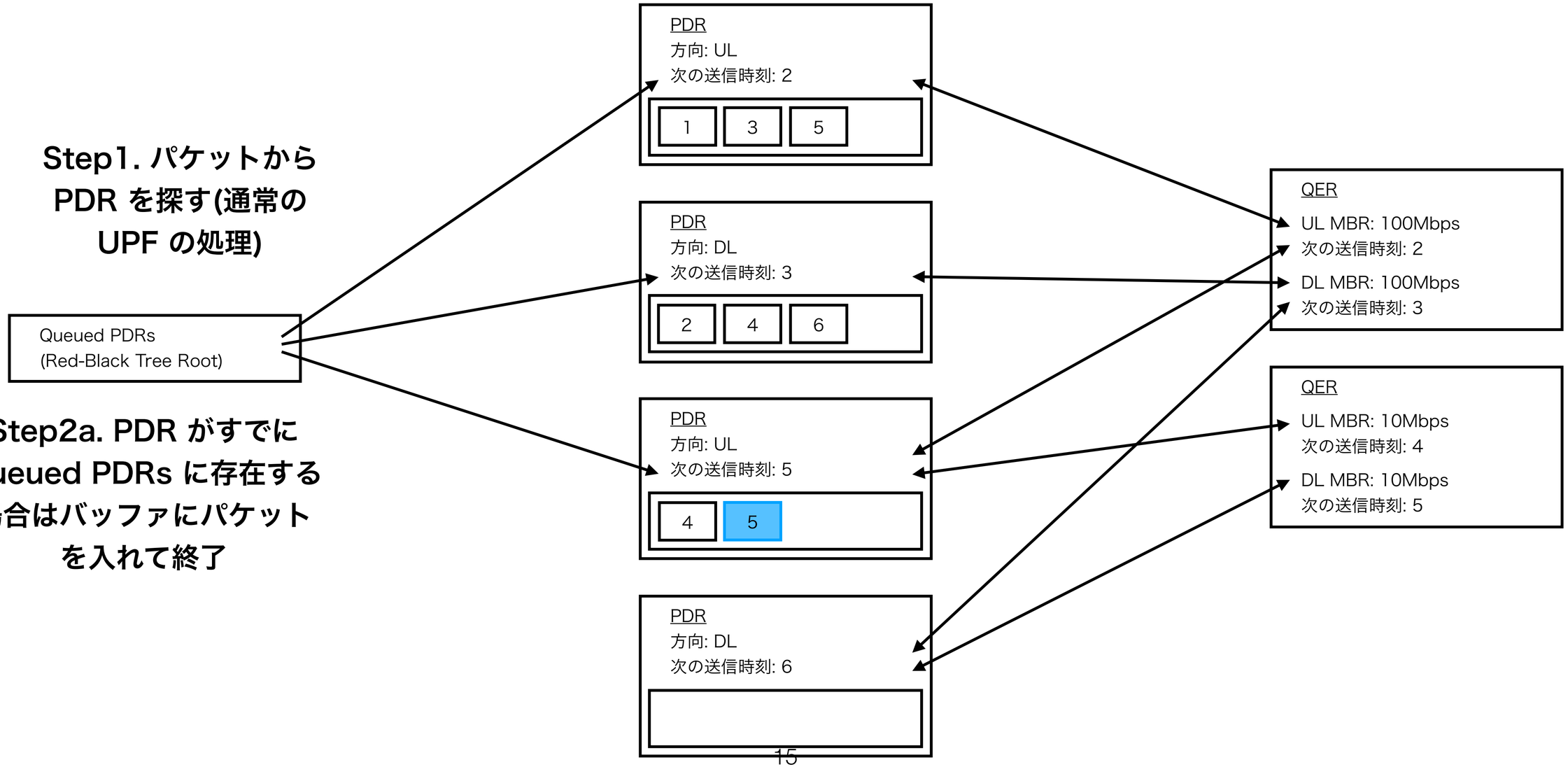
1. (PDR の)次の送信時刻
2. バッファ(FIFO)の先頭パケットの受信時刻
3. PDR 構造体のポインタアドレス



バッファの仕方: enq

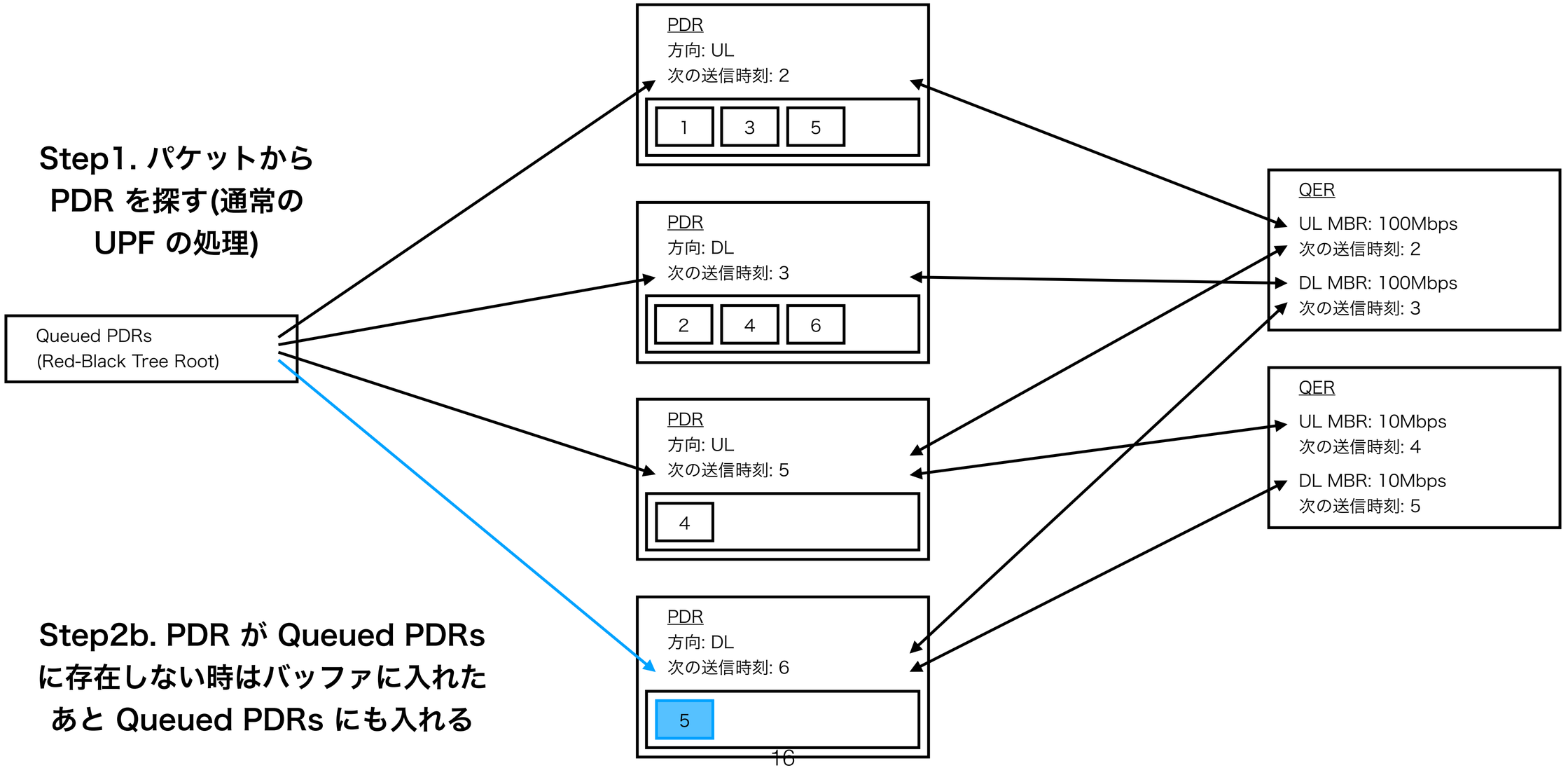
Step1. パケットから PDR を探す(通常の UPF の処理)

Step2a. PDR がすでに Queued PDRs に存在する場合はバッファにパケットを入れて終了



バッファの仕方: enq

Step1. パケットから PDR を探す(通常の UPF の処理)



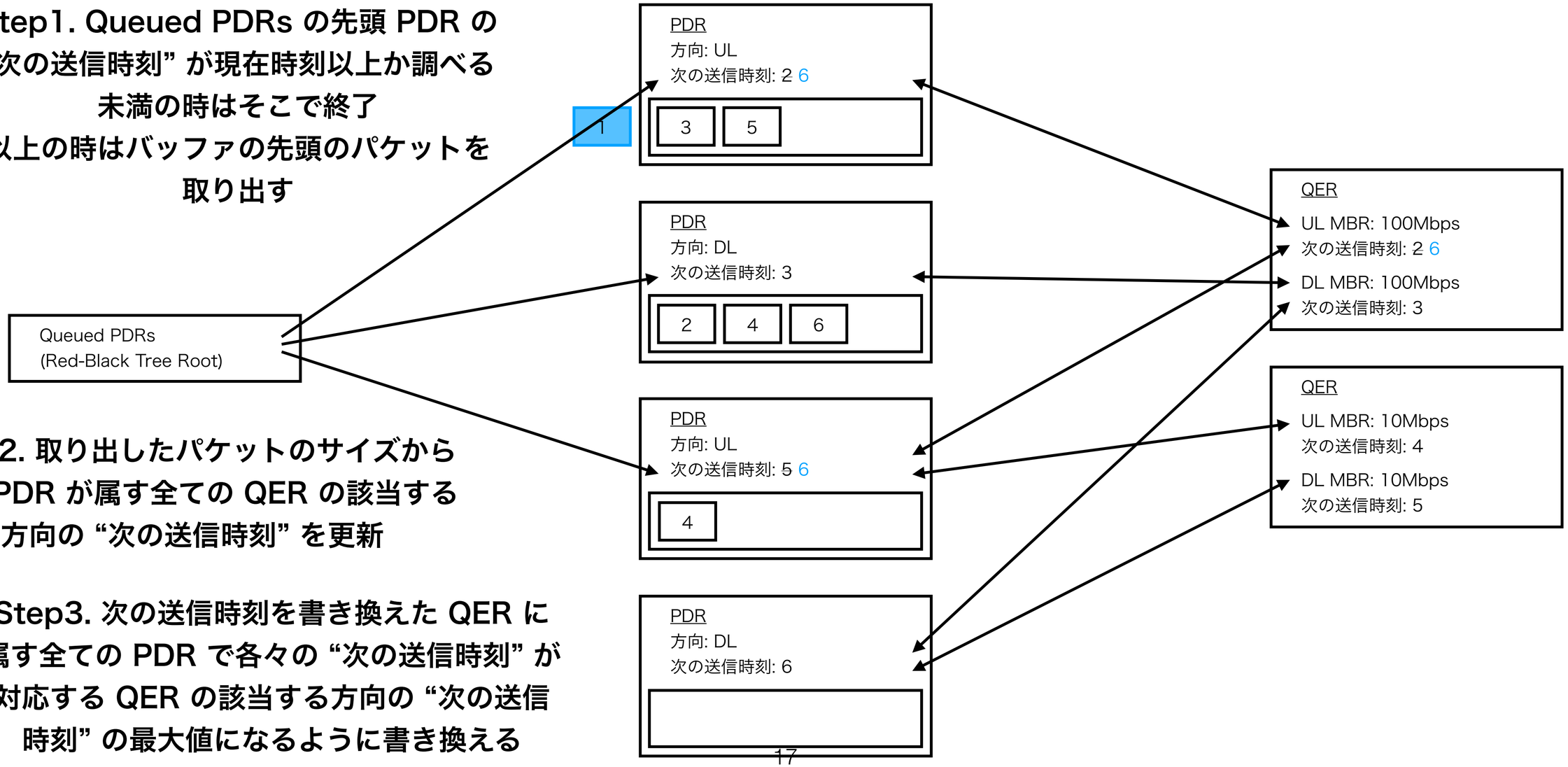
Step2b. PDR が Queued PDRs に存在しない時はバッファに入れたあと Queued PDRs にも入れる

バッファの仕方: deq

Step1. Queued PDRs の先頭 PDR の “次の送信時刻” が現在時刻以上か調べる
未満の時はそこで終了
以上の時はバッファの先頭のパケットを取り出す

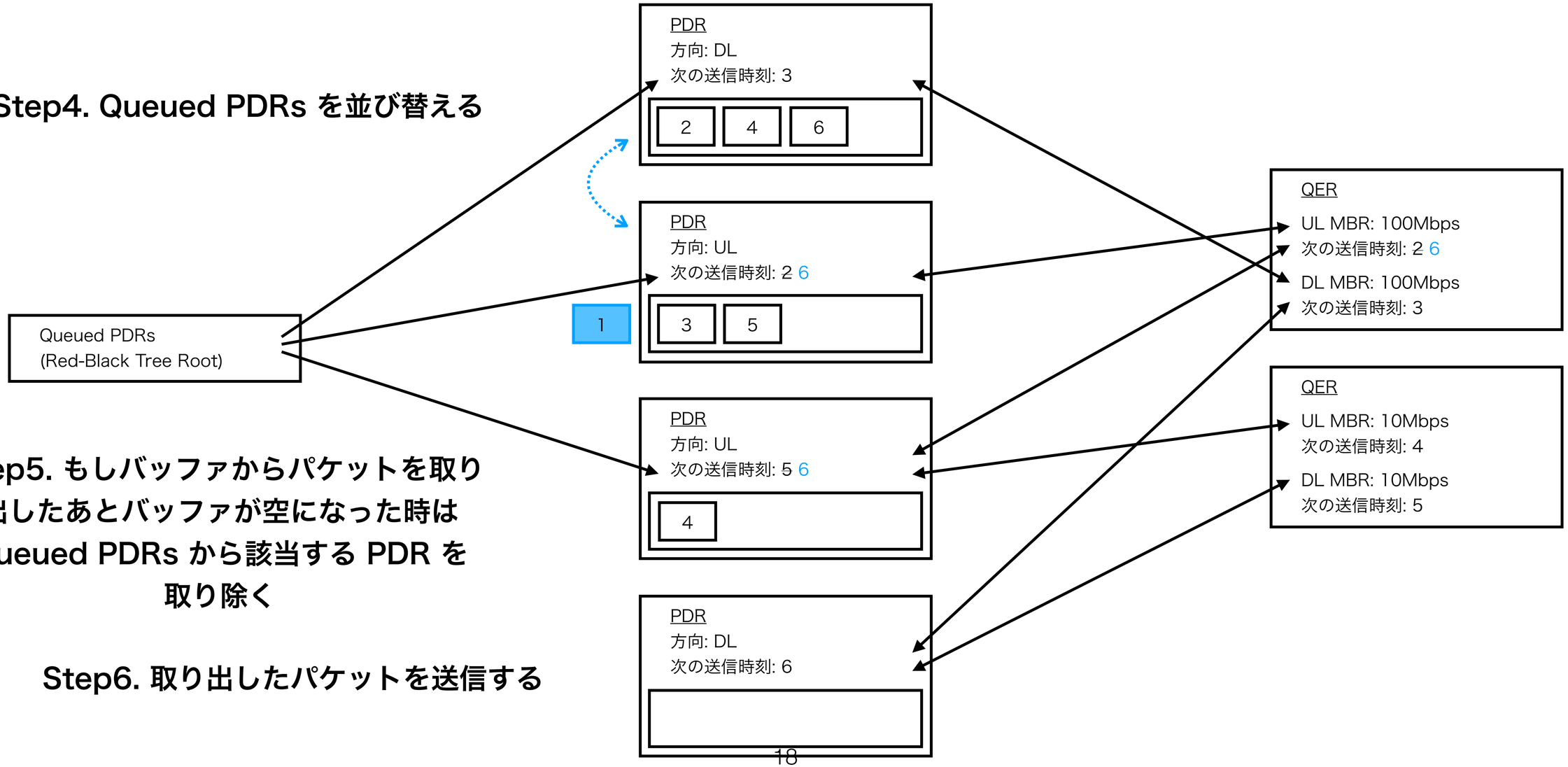
Step2. 取り出したパケットのサイズからその PDR が属す全ての QER の該当する方向の “次の送信時刻” を更新

Step3. 次の送信時刻を書き換えた QER に属す全ての PDR で各々の “次の送信時刻” が対応する QER の該当する方向の “次の送信時刻” の最大値になるように書き換える



バッファの仕方: deq

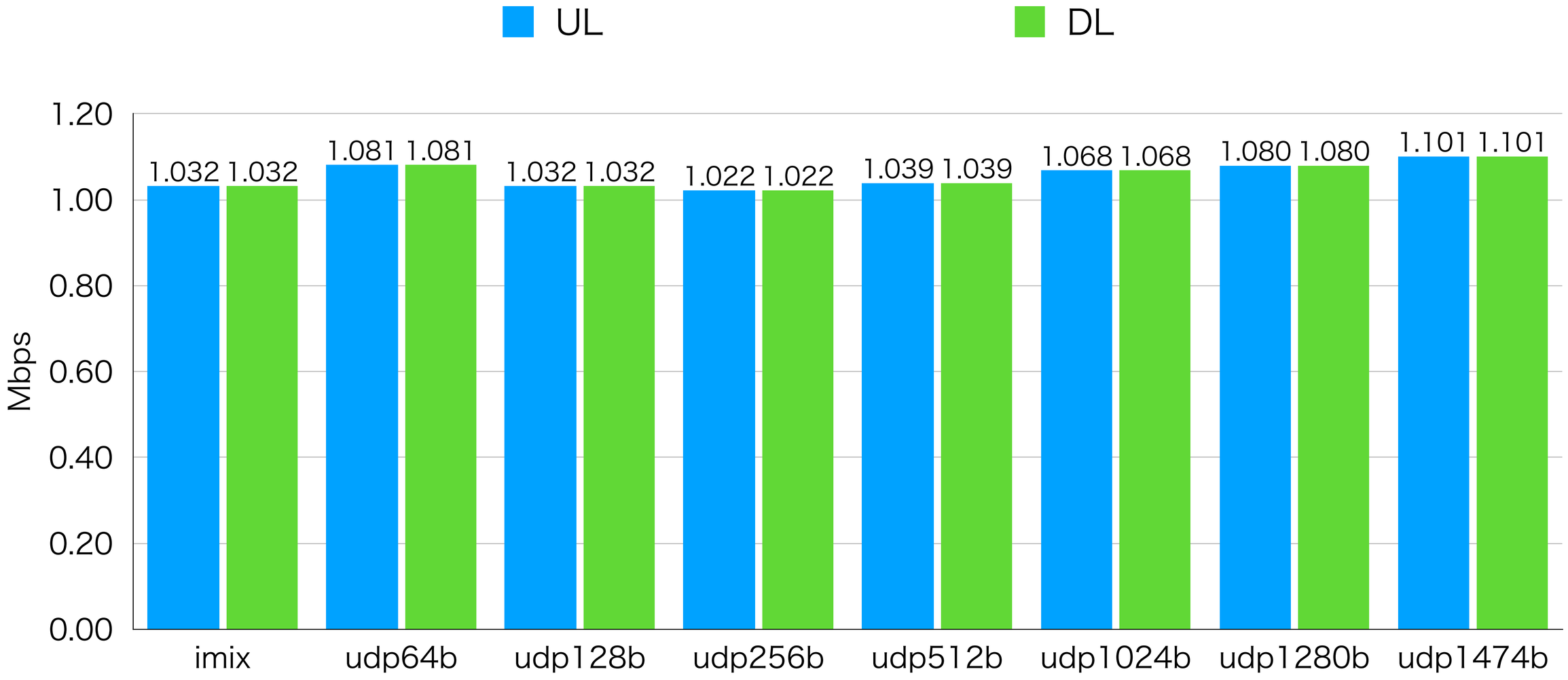
Step4. Queued PDRs を並び替える



Step5. もしバッファからパケットを取り出したあとバッファが空になった時は Queued PDRs から該当する PDR を取り除く

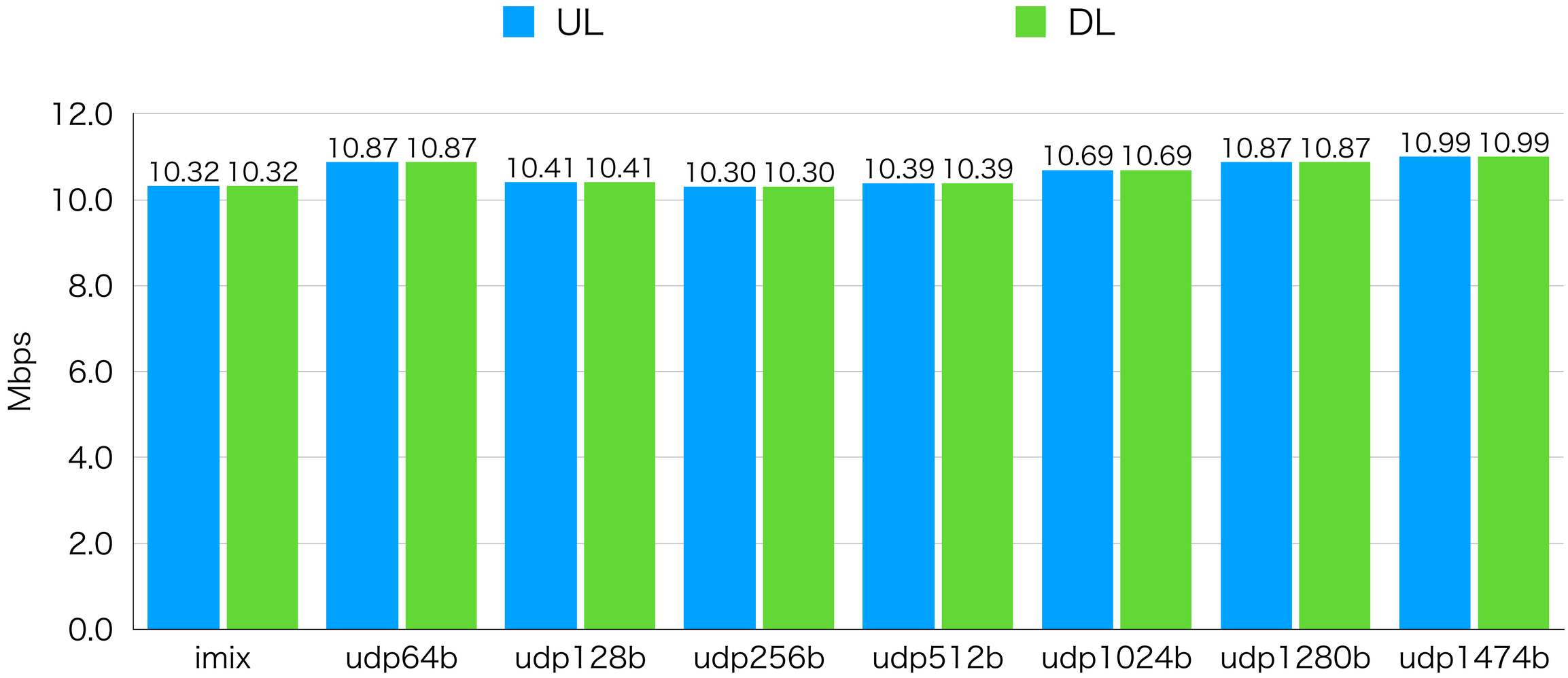
Step6. 取り出したパケットを送信する

ベンチマーク結果: MBR 1Mbps 1Flow



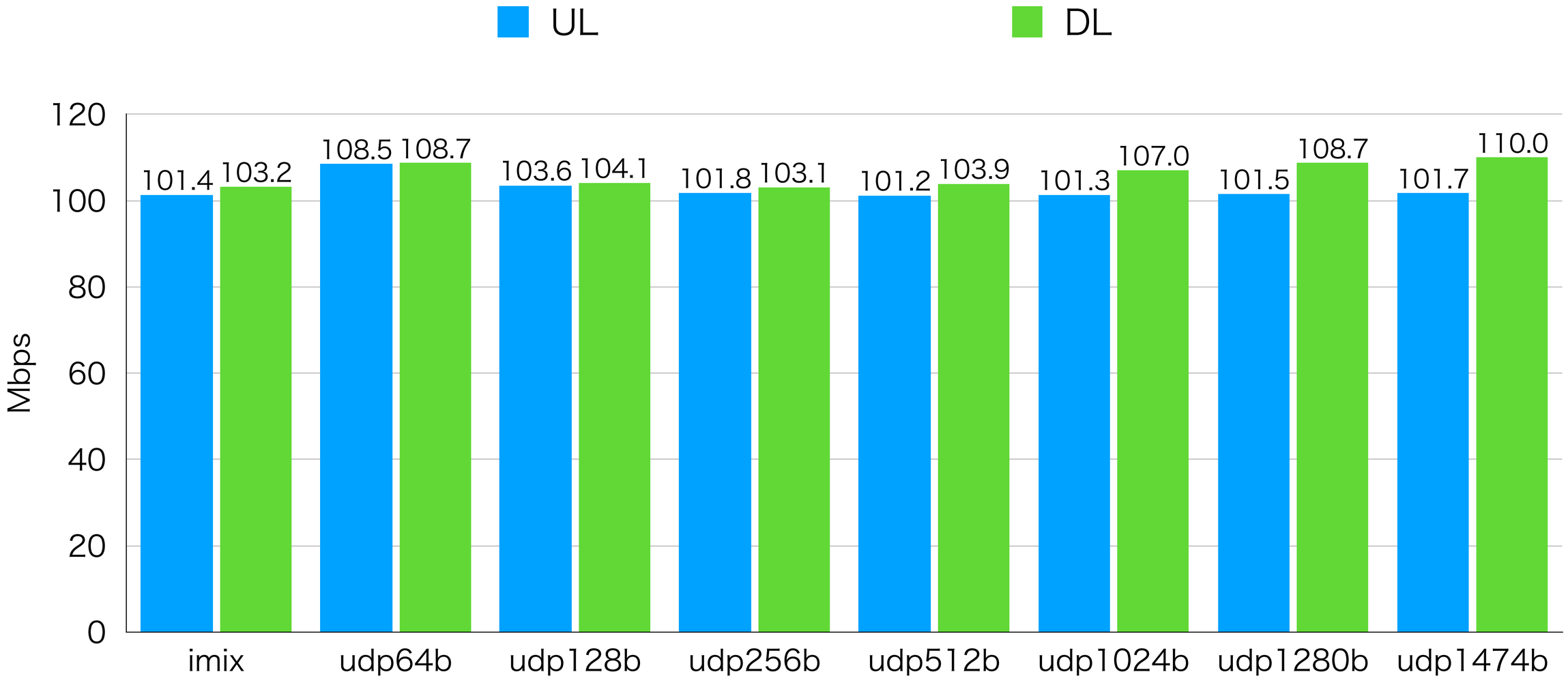
※ bps は IP ヘッダ以降で計算

ベンチマーク結果: MBR 1Mbps 10Flows



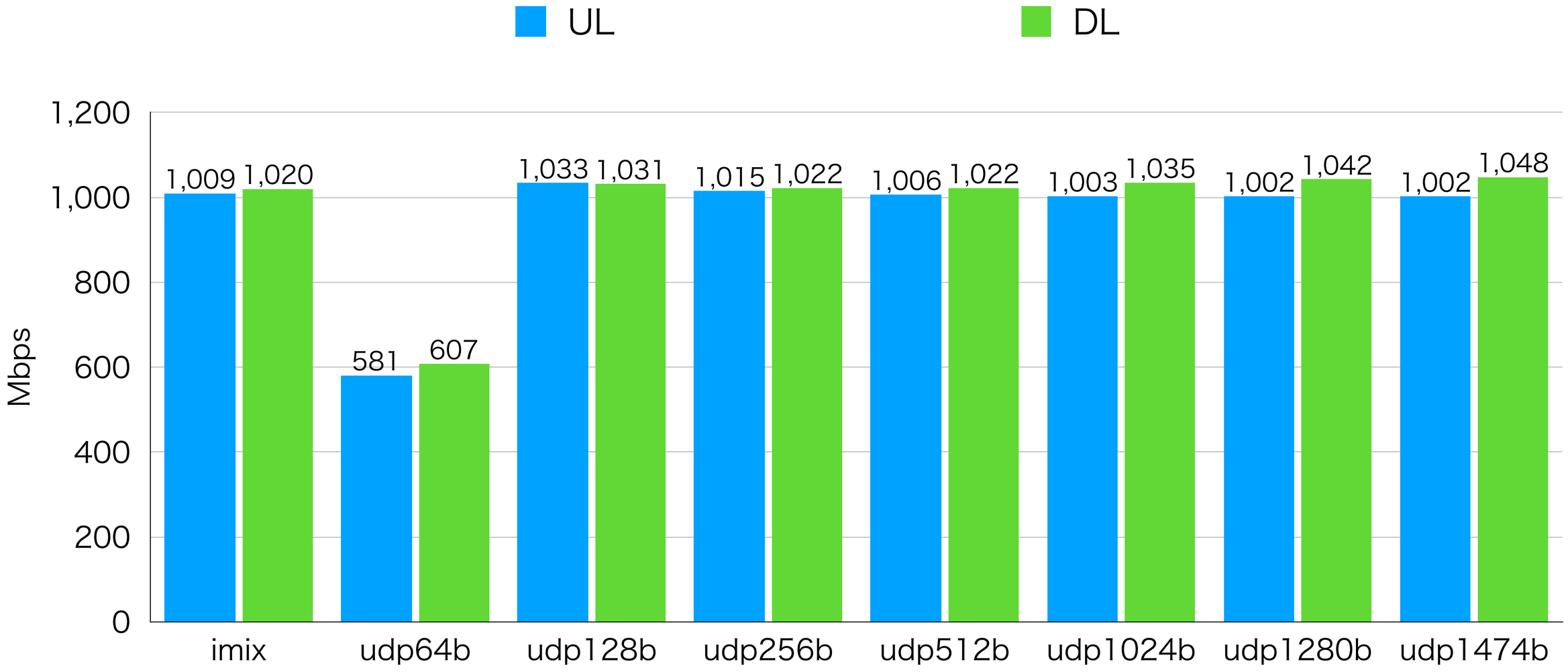
※ bps は IP ヘッダ以降で計算

ベンチマーク結果: MBR 1Mbps 100Flows



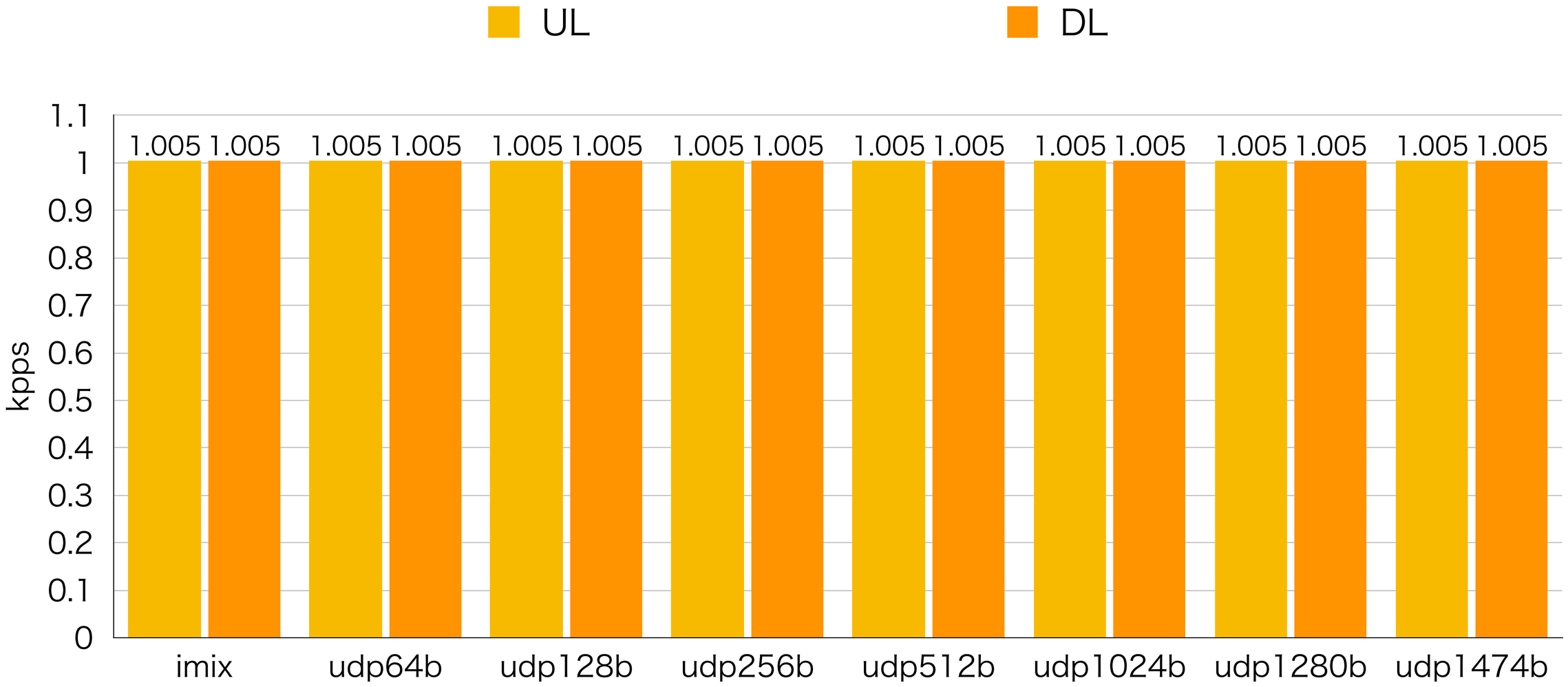
※ bps は IP ヘッダ以降で計算

ベンチマーク結果: MBR 1Mbps 1kFlows

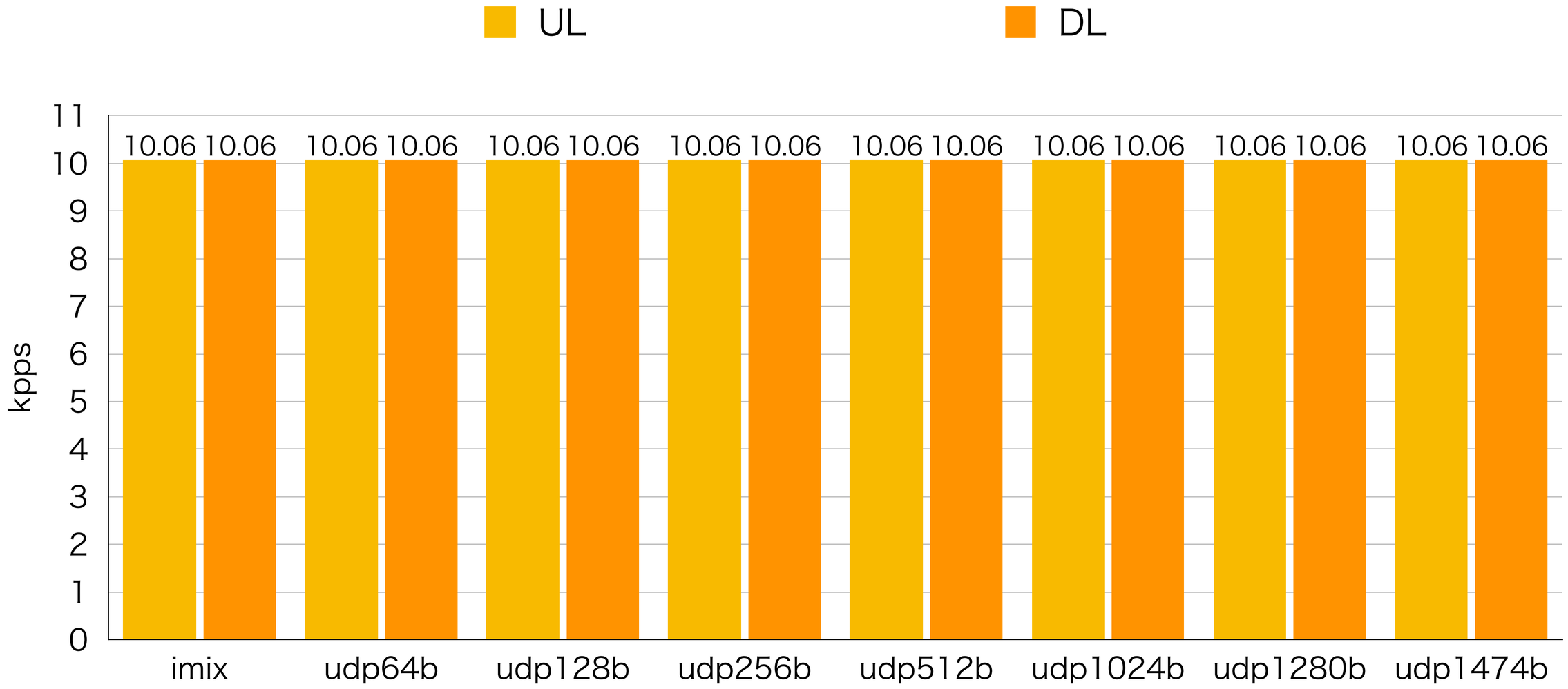


※ bps は IP ヘッダ以降で計算

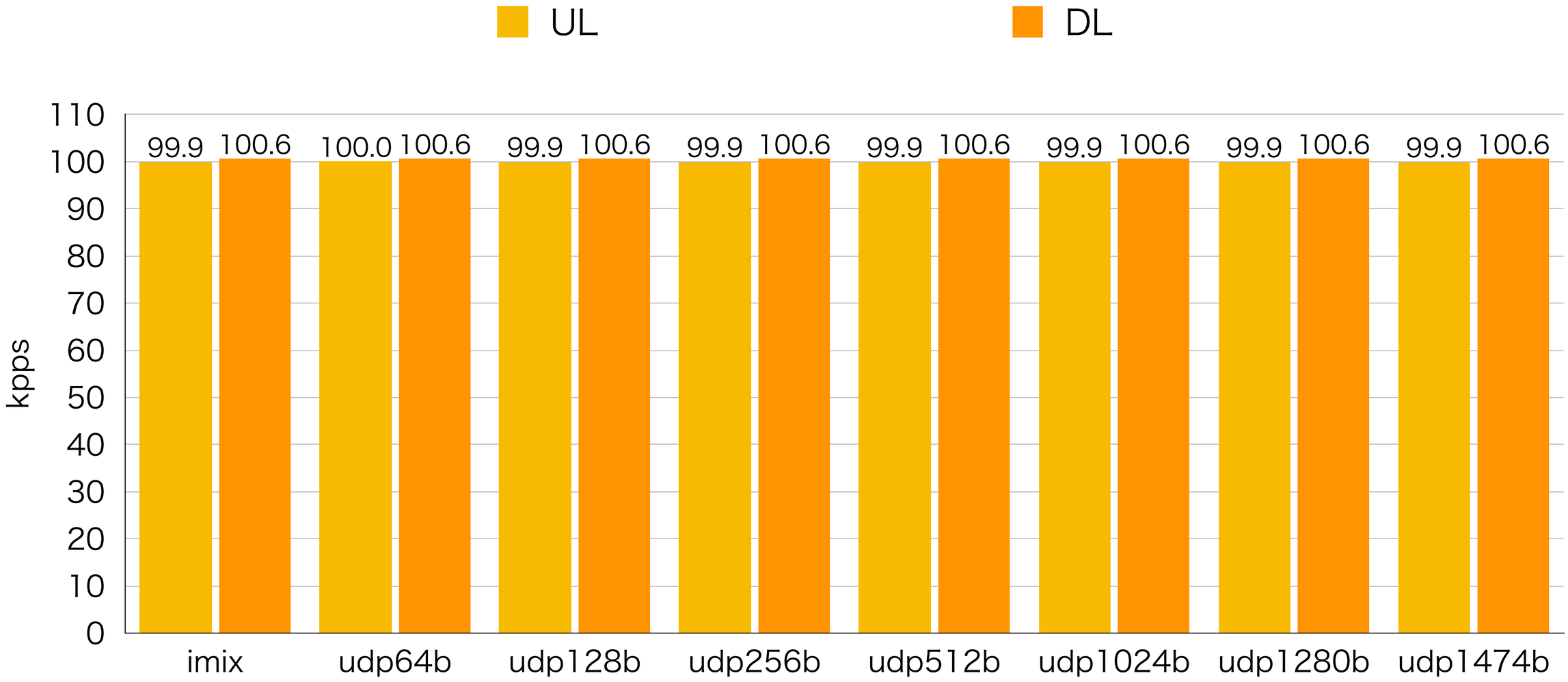
ベンチマーク結果: PacketRate 1kpps 1Flow



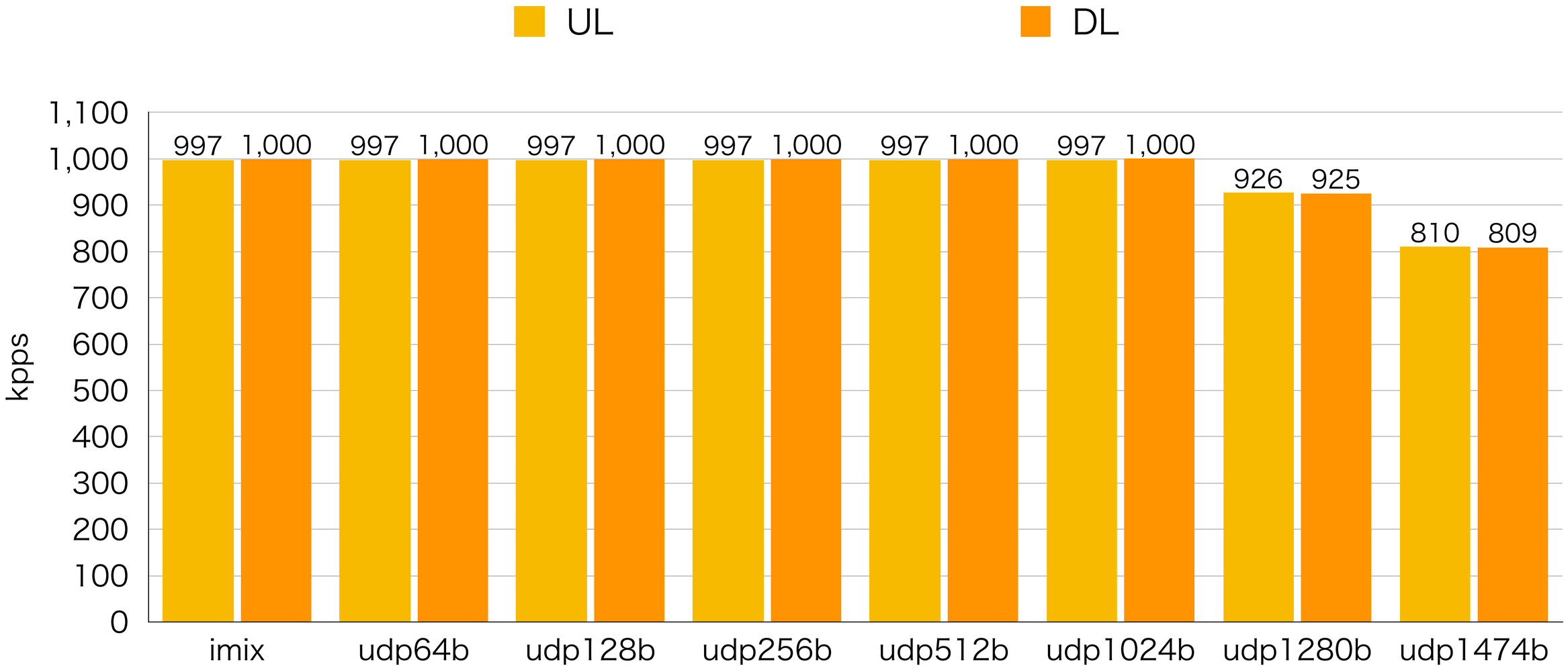
ベンチマーク結果: PacketRate 1kpps 10Flows



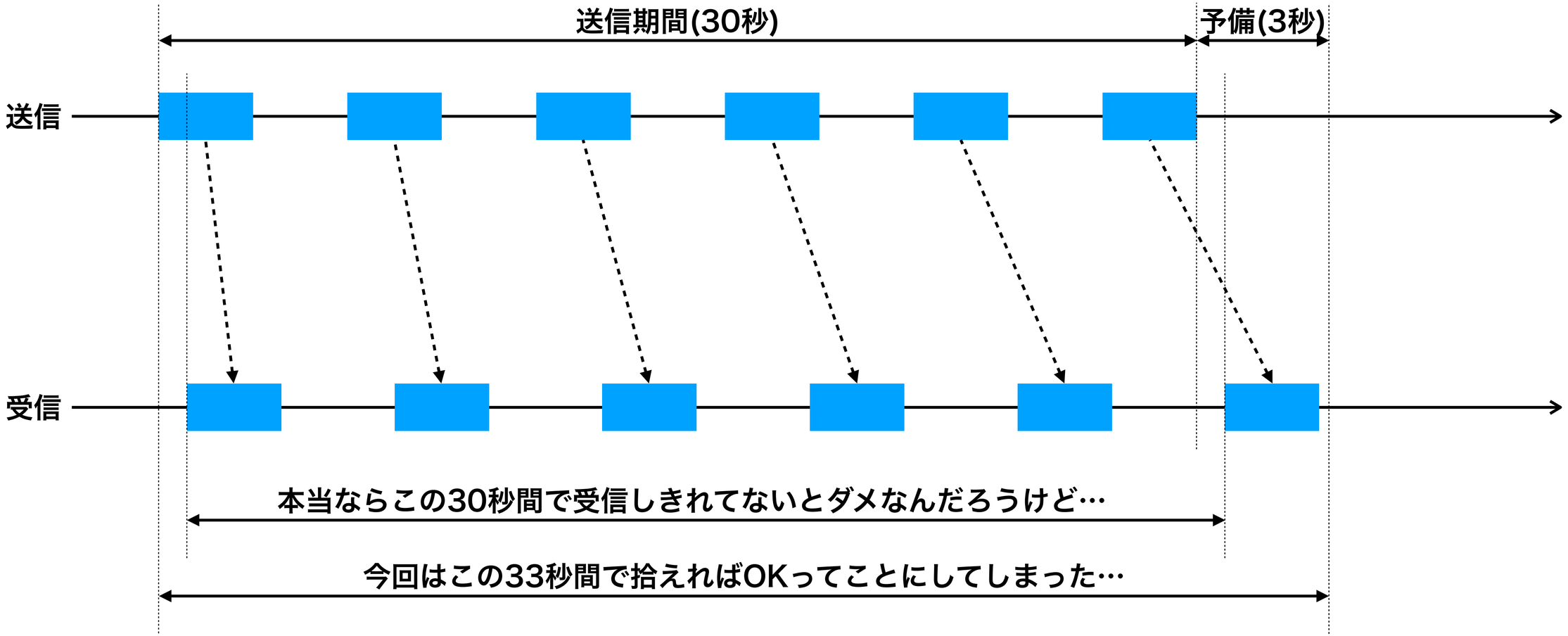
ベンチマーク結果: PacketRate 1kpps 100Flows



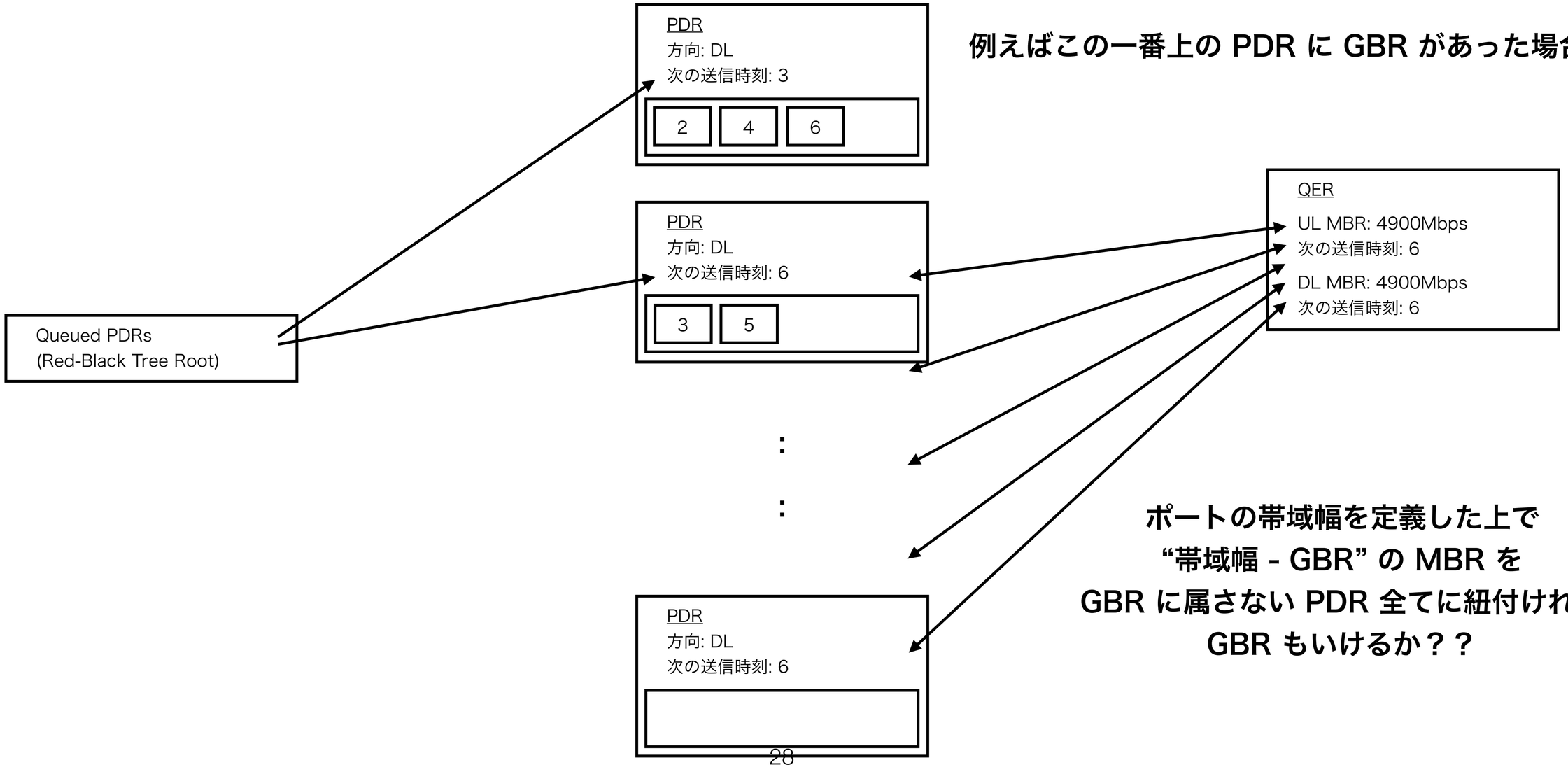
ベンチマーク結果: PacketRate 1kpps 1kFlows



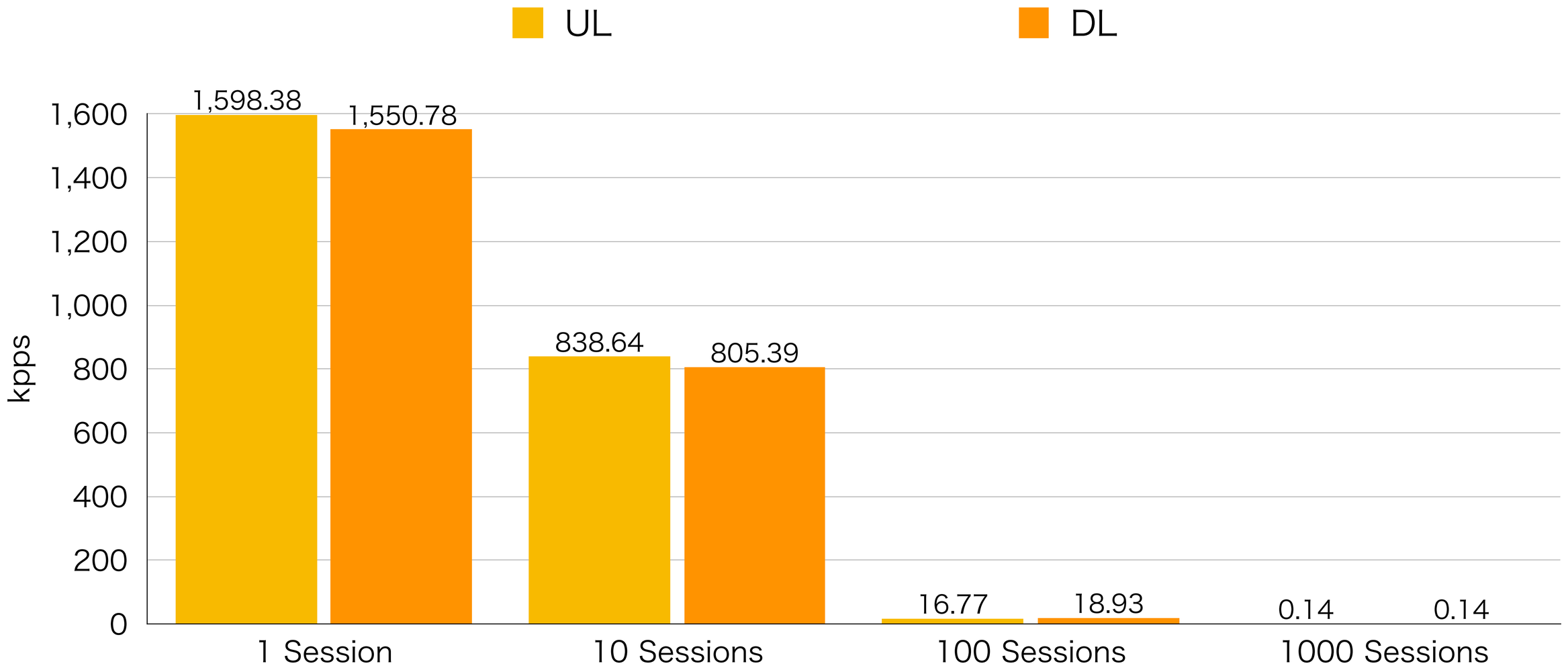
なんで若干多め？


送信期間(30秒)
予備(3秒)
送信
受信
本当ならこの30秒間で受信しきれてないとダメなんだろうけど…
今回はこの33秒間で拾えればOKってことにしてしまった…

お？ GBR もいけるか？

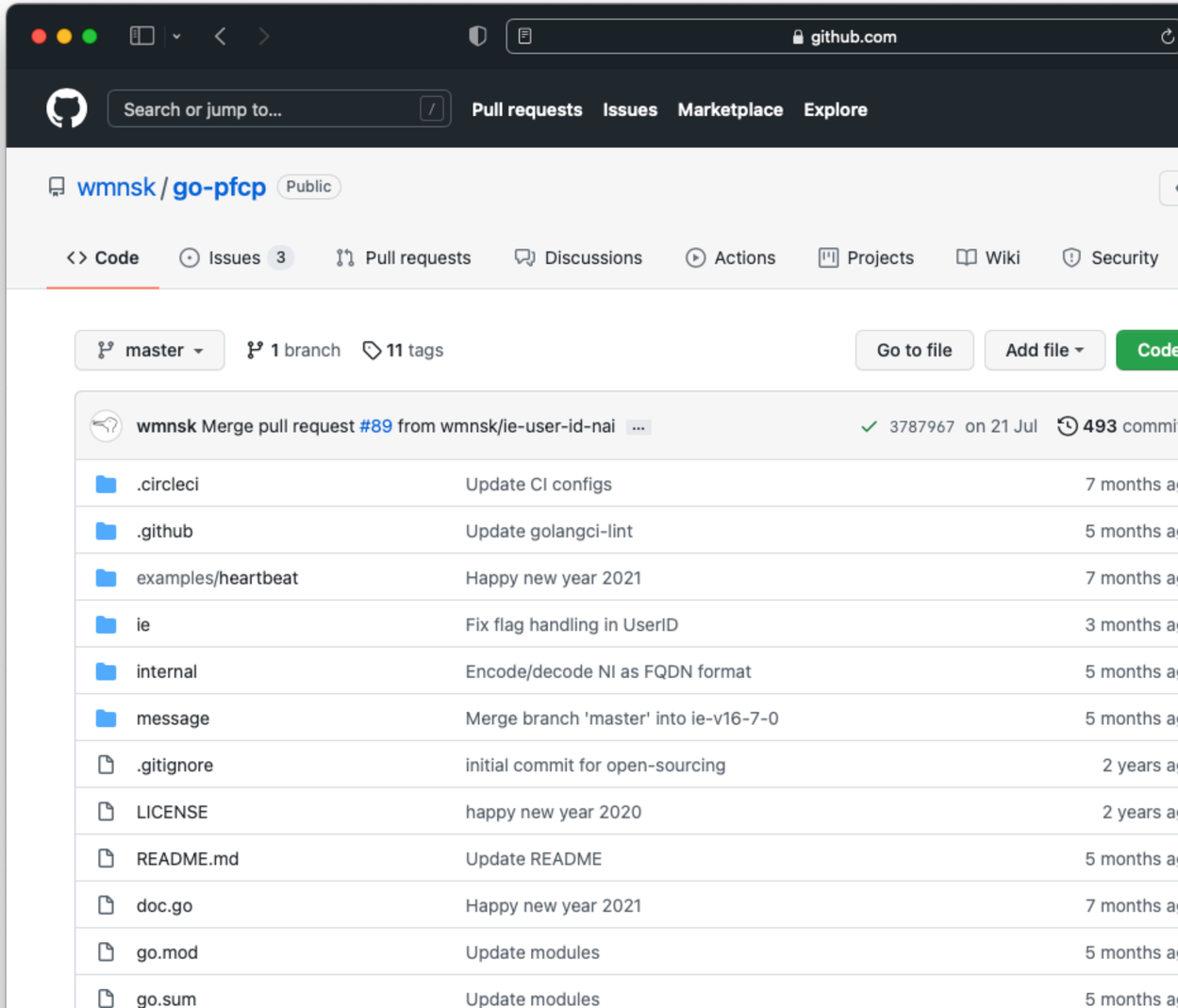

PDR
方向: DL
次の送信時刻: 3
2
4
6
PDR
方向: DL
次の送信時刻: 6
3
5
PDR
方向: DL
次の送信時刻: 6
Queued PDRs
(Red-Black Tree Root)
QER
UL MBR: 4900Mbps
次の送信時刻: 6
DL MBR: 4900Mbps
次の送信時刻: 6
例えばこの一番上の PDR に GBR があった場合…
ポートの帯域幅を定義した上で
“帯域幅 - GBR” の MBR を
GBR に属さない PDR 全てに紐付ければ
GBR もいけるか？？

ベンチマーク結果: GBR 1Mbps 1Flow udp64b



まとめ

- あとは PFCP を実装すれば QER が切れてそこそこ性能も出る UPF ができるかも？
  - PFCP は @wmnskdmms さんの go-pfcp があればいける！！！
- あーあと SDF Filter もやんないとか…
- ARP や IP ルーティングは L3 スイッチの VRRP MAC アドレス向けときゃとりあえずはいいか…？
  - でも ARP の返事くらいは実装した方が良いのかな…？
- つか nff-go の改造は取り込んでもらえたりすんのかな…



おまけ: BAR

- いる？？？


tools.ietf.org
IETF TOOLS
IETF Home
About Tools
Tools:
diffs spell
xml2rfc idnits
id2xml
tracker src
News
Get Passwd
IETF-112:
Rooms
Agenda
iCal maker
Documents
RFCs
Doc fetch:
Wikis:
IESG IRTF
Dev RSOC
Chairs Edu
Tools BOFs
NomCom
Areas
WGs:
concluded...
6lo*
6man
6tisch
Ace
Acme
Add
Alto
Anima
Asap
Asdf
Aqm Status Pages
Tsv Area: Zaheduzzaman Sarker, Martin Duke | 2013-Sep-
Active Queue Management and Packet Scheduling (Concluded WG)
Drafts | Agendas | Minutes | Wiki | Training | Charters | Jabber Room,Logs | List Archive | List search:
Current official charter page
2013... charter-aqm-2013-09-27
charter-aqm-2015-06-15 Δ
charter-aqm-2015-07-23 Δ
2015... charter-aqm-2015-09-10 Δ
charter-aqm-2016-04-06 Δ
charter-aqm-2016-04-07 Δ
2016... charter-aqm-2016-11-01 Δ
2016-11-01 charter
Active Queue Management and Packet Scheduling (aqm)
Charter
Current Status: Active
Chairs:
Richard Scheffenegger <rs.ietf@gmx.at>
Wesley Eddy <wes@mti-systems.com>
Transport Area Directors:
Spencer Dawkins <spencerdawkins.ietf@gmail.com>
Mirja Kühlewind <ietf@kuehlewind.net>
Transport Area Advisor:
Mirja Kühlewind <ietf@kuehlewind.net>
Mailing Lists:
General Discussion: aqm@ietf.org
To Subscribe: https://www.ietf.org/mailman/listinfo/aqm
Archive: https://mailarchive.ietf.org/arch/browse/aqm/
Description of Working Group:
Internet routers, lower-layer switches, end-host operating
systems, device drivers, and many types of additional
middleboxes include memory buffers in which they implement
queues to hold packets that require processing or otherwise
need to wait for forwarding to the next hop.
The queues are intended to absorb bursts of traffic that may
naturally occur, and avoid unneccessary losses. However, queues
also cause latency and jitter in the eventual arrival times of
packets. This can create issues and complications for interactive