Journal of Engineering Science and Technology
Vol. 16, No. 3 (2021) 2439 - 2453
© School of Engineering, Taylor's University

LAZY MULTI-LEVEL DYNAMIC TRAFFIC LOAD BALANCING PROTOCOL FOR DATA CENTER (LMDTLB)

ABIDULKARIM K. I. YASARI[1], ABDULKARIM D ABBAS[2], HAYFAA A ATEE[3,*], L.A. LATIFF[4], RUDZIDATUL A DZIYAUDDIN[4], DALAL A HAMMOOD[5]

[1]Engineering College, Al-Muthanna University, Al-Muthanna Province, Iraq
[2]Al-Maaref University College, Anbar Province, Iraq
[3]Institute of Administration Rusafa, Middle Technical University (MTU), Baghdad, Iraq
[4]UTM Razak Faculty of Technology and Informatics, Kuala Lumpur, Malaysia
[5]Electrical Engineering Technical College/ Middle Technical University (MTU), Baghdad, Iraq
*Corresponding Author: hayfaa-atee@mtu.edu.iq

Abstract

The minimization of tail latency is especially crucial in user interfacing services and fast responding apps. The literature on the datacenter load balancing protocols contains of many protocols but works discussing tail latency are scarce. This work proposes a novel variant of the Multi-Level Dynamic Traffic Load Balancing (MDTLB) protocol for a datacenter called the Lazy MDTLB or LMDTLB, which uses the concept of delaying the rerouting decision by a few packets for every flow that require path changes to provide the network with the time to ensure that a terrible path condition is not temporary. An evaluation of the state-of-the-art protocols of load balancing was conducted to determine the best performing one for curtailing tail latency involving flows of data mining, web search, and general flows. The findings confirmed that LMDTLB was the most efficient in minimizing tail latency and flow completion time (FCT).

Keywords: Datacenter, Data mining, Flow completion time, Tail latency, Traffic load.

1. Introduction

Limiting the latency of message delivery in datacenter network is crucial towards meeting the needs of applications such as web search [1], recommendation systems [2], and chatting platforms [3]. These applications share something in common, which requires quick responses and message deliveries, the failure of which results in decreased performance and subsequent financial losses [4]. For example, Google reported a 20% traffic decrease due to an extra 500 ms latency (introduced inadvertently), while Amazon said that every additional 100 ms of latency resulted in a 1% loss of revenue. Large flows cannot allocate buffers swiftly, which creates queueing delay and packet loss of short messages from buffer overflows. Packet loss is also anticipated due to black holes and silent random packet drops, all of which result in tail latency [5, 6].

The MDTLB protocol was a recently developed protocol for load-balancing [7]. It uses an adaptive approach to set the parameters and rerouting algorithms. It also improves the routing protocol called Hermes [8] by increasing the number of path levels to provide increased awareness of the path's status by dividing it into five types {very good, good, grey, bad, and very bad}. This allows for a more dynamic path evaluation via the introduction of static and dynamic thresholds. The MDTLB was compared with other state-of-the-art protocols and proved to be more effective. However, the assessment did not include the tail latency aspect.

This paper details the development of a new variant of MDTLB that includes tail latency. This was achieved by delaying the path decision until its state is confirmed. This new variant is called the lazy MDTLB, or LMDTLB.

Analysing the MDTLB from the perspective of tail latency is not extant in the literature. Therefore, comparing the performance of LMDTLB and MDTLB with other load balancing protocols such as DRB [9], Drill [10], Presto [11], CONGA [12], TLB [8], Clove [13], ECMP [14], and LetFlow [15] will be carried out.

This paper is organized in the following order: Section 2 outlines related work, Section 3 discusses the developed method, Section 4 details the experimental results and its analyses, and Section 5 concludes the paper.

2. Related Works

The literature has many works detailing the minimization and elimination of tail latency for datacenters path and load balancing.

Bai et al. [16] detailed a series of experimental testbeds and large-scale simulations in the course of determining the deep-rooted trade-offs between latency, throughput, and weighted fair sharing in multi-queue schemes by utilizing the guidelines of the exploration enabling the Explicit Congestion Notification (ECN) for multi-service multi-queue production Data Center Networks (DCNs). They reported that the performance of the proposed work is translatable to other schemes, and the proposed work resulted in a reduced performance at low loads. However, when the load was increased, the performance improved tremendously, but at very high loads, the performance decreased. Assuming that the Equal Cost Multi-Path (ECMP) is antagonistic to both service classes and flow sizes, it does not guarantee the diffusion of massive flows from the corresponding service class over diverse paths.

Geng et al. [17] tested a new reordering resilient data center network capable of sending any packet at any level of priority and paths. His design utilized the short delay of packets in a data center network alongside the underlying traffic business to eliminate the undesired effects of packet reordering almost by its entirety. It maintains this state for only a small number of flows at any given time.

HOMA is a novel transport protocol architecture for data center networks [18]. It provides high workloads of tiny messages low latency by implementing an in-network message queue with dynamic priority allocation. HOMA also supports long messages with efficient network utilization using controlled receiver downlinks at high load.

Wang et al. [19] designed, analysed, and evaluated Luopan, which is a novel sampling-based load balancing protocol that improved the FCT and scalability in large-scale networks. For each destination switch, a few paths were periodically sampled, and the flow cells will be directed to the least congested switch.

Katta et al. [20] overcame two limitations; the switch memory limitation and implementation in custom hardware. In the former, done at the edge switches, the congestion-tracking state cannot be maintained effectively and is less scalable for large-scale topologies. Also, because in-field modification is impossible, the HULA algorithm is implemented instead of using leaf switches track-congestion over all of the paths to the destination. This will be done for the best route via a neighboring switch to the goal. HULA was designed for coding in emerging programmable switches, such as in P4 (Programming Protocol-independent Packet Processors) [21], and executed on programmable chipsets without requiring custom hardware.

Another interesting work is the work of Huang et al. [22]. The Queueing Delay Aware Packet Spraying (QDAPS) was designed to decrease the packet reordering for the packet-level load balancer by selecting packets paths according to the packet queueing delay output buffer, while also permitting the arriving packet to be forwarded before the later packets to prevent packet reordering.

The work detailed in [4] involved the design, implementation, and evaluation of Cloudburst, a simple readily-deployable solution to achieve similar or even better results without complexities. The Cloudburst explores Forward Error Correction (FEC) over the multipath: it proactively spreads FEC-coded packets generated from messages over the multipaths, recovers them with the first few arrivals, and exploits underutilized paths to decrease tail latency.

3. Method

This section is divided into two sub-sections; sub-section 3.1 details a review of the MDTLB, while sub-section 3.2 elucidates a newer variant of MDTLB called LMDTLB.

3.1. MDTLB

MDTLB utilizes thresholds and parameters to represent the paths. It identifies the best one by validating the lower Round Trip Time (RTT) measurement and the lower the rate of the congestion mismatch. This allows for the congested route to be pinpointed as per the RTT and the explicit congestion notification (ECN). The protocol consists of an adaptive parameter-setting and rerouting algorithm.

The MDTLB algorithm for rerouting logic was designed to pinpoint the optimal path for each flow. The genesis of this is if there is a new flow, the process will involve searching for suitable (very good) paths. If the search failed to produce appropriate paths, it will search for a good (grey) path, and failing that, it will randomly select a path. In this case, the proposed approach will choose a path that has the lowest local sending rate. In the presence of old flows with bad paths, low rates, and sent size lower than the threshold (S), the approach will search for a better path. If there are none, it stops the rerouting process.

The following consideration was taken; the flow sends several data before the network condition changes. The current flow path is congested. Thus, the packet reordered for rerouting to a less crowded path is required. Multiple conditions are employed to decrease the frequency of rerouting, which delivers only when there is a gain in performance:

3.1.1. The remaining size of flow:

If the remaining size of the flow being sent is small, rerouting benefit will be limited and lower than the harm garnered from packet reordering. Therefore, the rerouting will not occur if the remaining size of the flow is less than the size threshold S.

3.1.2. The difference in sending rate:

If the sending rate of the new path is not significantly better than its older counterpart, the rerouting decision will not occur. The presence of multiple path levels helps increase the choices when selecting the best flow path.

In summary, the MDTLB approach in solving the packet reordering limits packet reordering to curtail its effect instead of using additional mechanisms to do so, which could slow the network. The FCT and tail latency improved via the implementation of this approach.

The pseudocode for judging algorithm is illustrated in Table 1. In Fig. 1, the illustration of multi-level path judging algorithm is provided.

Table 1. Judging algorithm [7].

```
1.  for each p (path) do:
2.  if fECN < $T_{ECN}$ and $T_{RTT}$ < $T_{RTT\_LS}$ then type = very good
3.      else if fECN < $T_{ECN}$ and $T_{RTT}$ < $T_{RTT\_LD}$ then type = good
4.      else if fECN > $T_{ECN}$ and $T_{RTT}$ > $T_{RTT\_HS}$ then type = bad
5.      else if fECN >$T_{ECN}$ and $T_{RTT}$ > $T_{RTT\_HD}$ then type = very bad
6.      else   type = grey
7.  if ($n_{tout}$ > 3 and no packet is ACKed) or ($f_{ret}$ > 1% and type ≠ bad or very bad), then
    type = failed
8.  end for
```

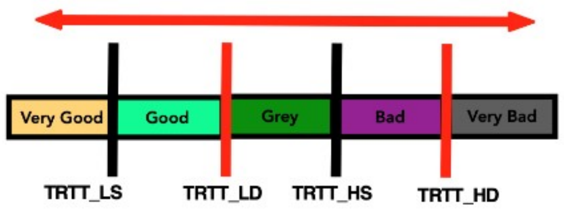


Fig. 1. Multi-level path judging algorithm [7].

The logic of rerouting algorithm is illustrated in Table 2.

Table 2. Rerouting algorithm [7].

```
1.  for each packet do:
2.  Assume the packet flow is $f$ and its path is p
3.  if $f$ is new or $f$.if$_{tout}$ == true or p$_{type}$ == failed then
4.  {p'} = all very good paths
5.    if {p'} $\neq \phi$ then
6.        P* = Argmin$_{p \in \{p'\}}$(p.r$_p$)
7.  */The very good path with the smallest sending rate will be selected*/
8.    else
9.      {p"} = all good paths
10.     if {p"} $\neq \phi$ then
11.         P* = Argmin$_{p \in \{p''\}}$(p.r$_p$)
12. */The good path with the smallest sending rate will be selected*/
13.   else
14.                 {p'''} = all grey paths
15.                 if {p'''} $\neq \phi$ then
16.                     p* = Argmin$_{p \in \{p'''\}}$(p.r$_p$)
17.                 else
18.                     p* = random selection for path with no failure
19.  else if p:type == very bad then
20.           if $f.ss$ < S and $f.rf$ < R then
21.               {P'} = all very good paths are better than p
22.               / $\forall$p'$\in$ {P'} , we have p.tRTT - p'.tRTT > $\Delta_{RTT}$ and p.fECN -
    p'.fECN >$\Delta_{ECN}$/*
23.               if {p'} $\neq \phi$ then
24.                   P* = Argmin$_{p \in \{p'\}}$(p.r$_p$)
25. else
26.     {p"} = all good paths are better than p
27.     if {p"} $\neq \phi$ then
28.         P* = Argmin$_{p \in \{p''\}}$(p.r$_p$)
29. */The good path with the smallest sending rate will be selected*/
30.             else
31.                 {p'''} = all grey paths are better than p
32.                 if {p'''} $\neq \phi$ then
33.                     p* = Argmin $_{p \in \{p'''\}}$(p.r$_p$)
34.                 else
35.                     p = p     /* Don't reroute/*
36.     return p*      /* The new routing path */
37. end for
```

This protocol has not been evaluated from the perspective of tail latency. MDTLB has been compared with: DRB, Drill, Presto, CONGA, TLB, Clove, ECMP, and LetFlow schemes. Three types of flow are considered: data mining, web search, and general flows. The evaluation is conducted by generating two metrics: the first metric is the normalized FCT and the second is the throughput for each type of the flows using one of the mentioned states of the art protocols.

3.2. LMDTLB

This section presents the developed method for achieving another variant of MDTLB with a lower tail latency called Lazy Multi-Level Dynamic Traffic Load Balancing LMDTLB protocol. The flowchart in Fig. 2, illustrates the process.

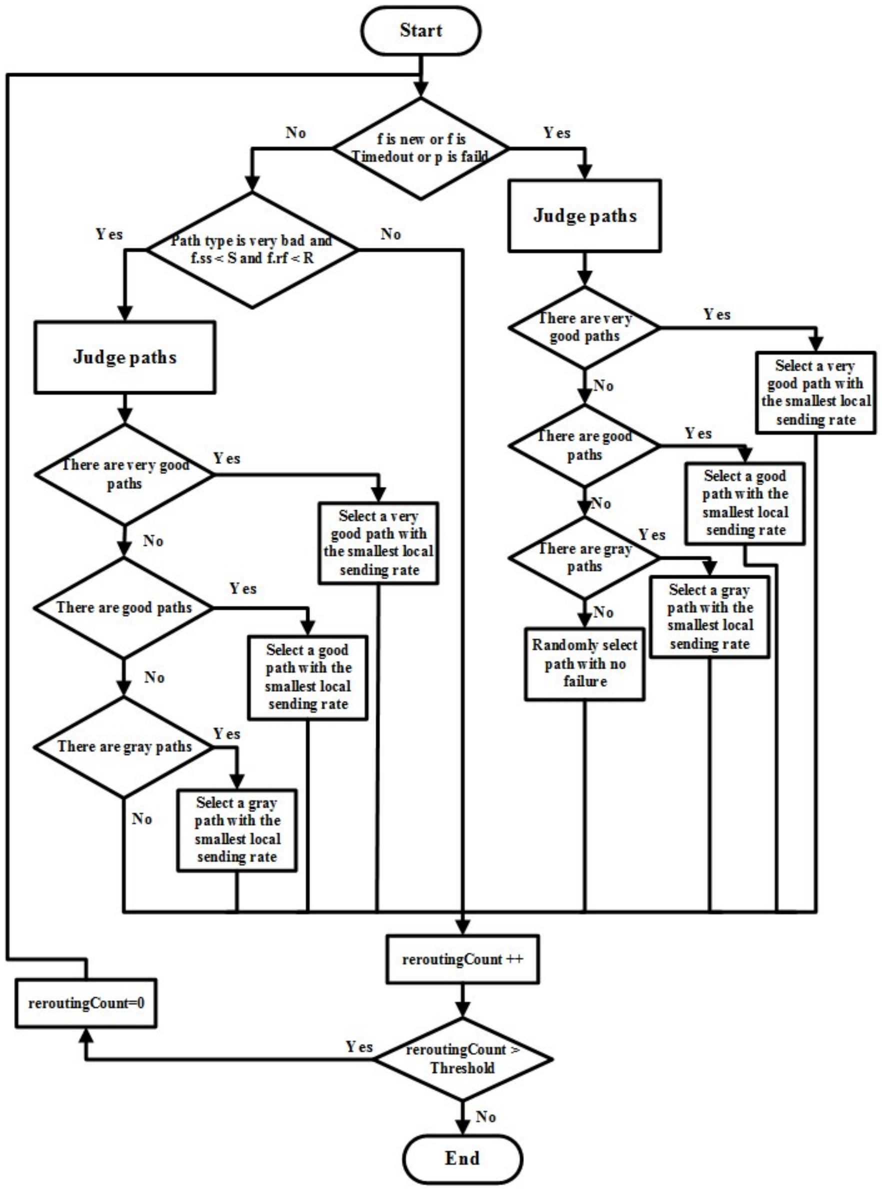

Start
f is new or f is Timedout or p is faild
No
Yes
Judge paths
Path type is very bad and f.ss < S and f.rf < R
Yes
No
Judge paths
There are very good paths
Yes
No
Select a very good path with the smallest local sending rate
There are good paths
Yes
No
Select a good path with the smallest local sending rate
There are gray paths
Yes
No
Select a gray path with the smallest local sending rate
There are very good paths
Yes
No
Select a very good path with the smallest local sending rate
There are good paths
Yes
No
Select a good path with the smallest local sending rate
There are gray paths
Yes
No
Select a gray path with the smallest local sending rate
Randomly select path with no failure
reroutingCount ++
reroutingCount=0
reroutingCount > Threshold
Yes
No
End

Fig. 2. LMDTLB rerouting logic.

The concept of this protocol is to delay the rerouting decision a few packets (routing Threshold) for every flow when it needs to change its path in order to give the network more time to assure that the very bad path condition is temporary, that can help in reducing the rerouting of flows, and hence will reduce the packet reordering effect and improve tail latency. The logic of rerouting in LMDTLB protocol is demonstrated in Table 3.

Table 3. Rerouting in LMDTLB protocol.

```
1.  for each packet do:
2.  Assume the packet flow is $f$ and its path is p
3.  if $f$ is new or $f.if_{tout}$ == true or $p_{type}$ == failed then
4.  {p'} = all very good paths
5.    if {p'} $\neq \phi$ then
6.       P* = $Argmin_{p∈\{p'\}}(p.r_p)$
7.  */The very good path with the smallest sending rate will be selected*/
8.    else
9.      {p"} = all good paths
10.     if {p"} $\neq \phi$ then
11.        P* = $Argmin_{p∈\{p''\}}(p.r_p)$
12. */The good path with the smallest sending rate will be selected*/
13.        else
14.                   {p'"} = all grey paths
15.                   if {p'"} $\neq \phi$ then
16.                      p* = $Argmin_{p∈\{p'''\}}(p.r_p)$
17. */The grey path with the smallest sending rate will be selected*/
18.                   else
19.                      p* = random selection for path with no failure
20.  else if p: type == very bad then
21.            if $f.ss$ < S and $f.rf$ < R then
22.                reroutingCount++
23.                if reroutingCount > R then
24.                  reroutingCount = 0
25.               {P'} = all very good paths are better than p
26.             /* ∀p'∈ {P'} , we have p.tRTT - p'.tRTT > $\Delta_{RTT}$ and p.fECN
    - p'.fECN >$\Delta_{ECN}$/*
27.                 if {p'} $\neq \phi$ then
28.                   P* = $Argmin_{p∈\{p'\}}(p.r_p)$
29.   */The very good path with the smallest sending rate will be selected*/
30.  else
31.     {p"} = all good paths notably better than p
32.     if {p"} $\neq \phi$ then
33.        P* = $Argmin_{p∈\{p''\}}(p.r_p)$
34. */The good path with the smallest sending rate will be selected*/
35.     else
36.            {p'"} = all grey paths are better than p
37.             if {p'"} $\neq \phi$ then
38.                 p* = Argmin $_{p∈\{p'''\}}(p.r_p)$
39. */The grey path with the smallest sending rate will be selected*/
40.            else
41.                 p* = p           */ Do not reroute */
42.     return p*                    */The new routing path */
43. end for
```

4. Experimental Results and Analysis

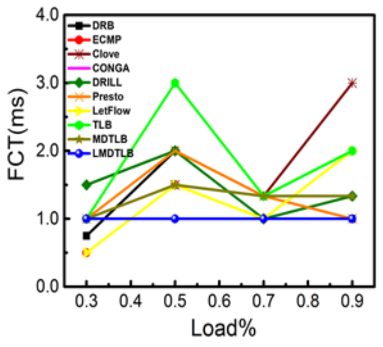
LMDTLB was implemented in Network Simulator 3 (NS-3). The simulation environment was configured using the selected parameters. The LMDTLB tested under a symmetric 4 x 4 leaf-spine topology, which includes 64 hosts, with connection links of 10 Gbps. The simulation conducted as a 2:1 oversubscription at the leaf level as like the characteristic data center deployments. Three types of flows are used to conduct the simulation. The first type is data mining flow presented by VL2 with flow cumulative distribution function (VL2_CDF) [23]. The second type of flows is the web-search flow, and it is presented by data center TCP algorithm with flow cumulative distribution function (DCTCP_CDF). Lastly, the third flow type is the general flow which is derived from the first two flows as in following: (VL2_CDF + DCTCP) / 2. The simulation parameters of LMDTLB are shown in Table 4.

Table 4. Simulation parameters of implemented load balancing routing protocols.

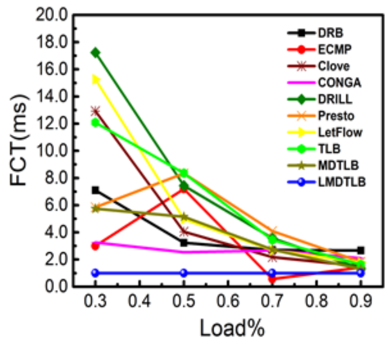
Parameters	Values
runMode	LMDTLB, MDTLB, TLB (Hermes), Conga, Presto, DRB, ECMP, Clove, DRILL, LetFlow
transportProt (Transport protocol**)**	Tcp
enableLargeDupAck	False
enableLargeSynRetries	False
enableFastReConnection	False
enableLargeDataRetries	False
For each leaf serverCount	8
spineCount	4
leafCount	4
linkCount	1
spineLeafCapacity	10 Gbps
leafServerCapacity	10 Gbps
linkLatency (one hop link latency)	10 μs
cdfFileName (CDF cumulative distribution functions)	data mining: VL2_CDF, web search: DCTCP_CDF, general flows: (VL2-CDF + DCTCP_CDF)/2
load	[0.3, 0.5, 0.7, 0.9]
LMDTLBMinRTTS (Min static threshold)	40 μs
LMDTLBHighRTTS (Max static threshold)	180 μs
Initial LMDTLBMinRTTD (Initial min dynamic threshold)	40 μs
Initial LMDTLBHighRTTD (Initial max dynamic threshold)	180 μs
LMDTLBBetterPathRTT (RTT judging threshold to check if the path is better than another)	1 μs

LMDTLBT1 (Time interval to update the path condition)	100 µs
LMDTLBECNPortionLow	0.1
LMDTLBProbingEnable	True
LMDTLBProbingInterval	50 µs
LMDTLBSmooth	True
LMDTLBRerouting	True
LMDTLBS	64000 bytes
LMDTLBReverseACK	True
quantifyRTTBase	10
LMDTLBFlowletTimeout	5 ms
Simulation Time	0.25 s

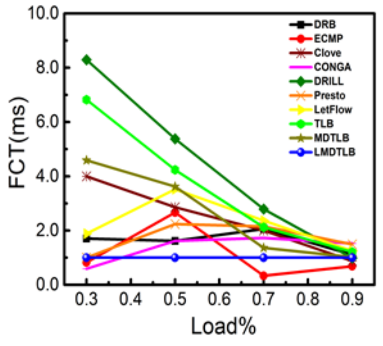
In Fig. 3, the FCT was generated for each of the protocols. LMDTLB had the lowest FCT comparing with the other protocols. This is shown more obvious for the web-search and general flows as it is observed in Figs. 3(d) to (k) and still recognized in data mining flows in Figs. 3(a) to (c). It is observed that in most experiments, FCT is lower for LMDTLB algorithm comparing with the benchmarks. In almost all types of flows and with load lower than 0.9 FCT is better with high percentage which can reach up to 350% in some cases. It is also observed that for very high flows there are similar values of FCT for almost all protocols. Some differences exist because of the random behaviour.



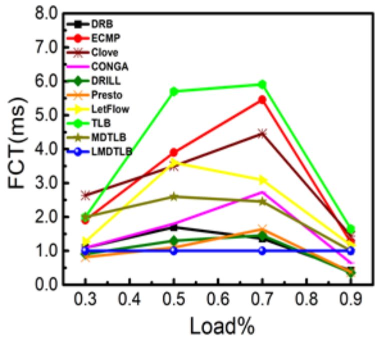
(a) FCT for data mining short flows.



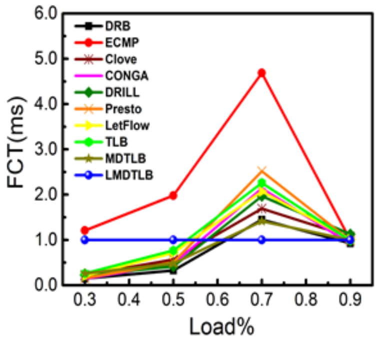
(b) FCT for data mining long flows.



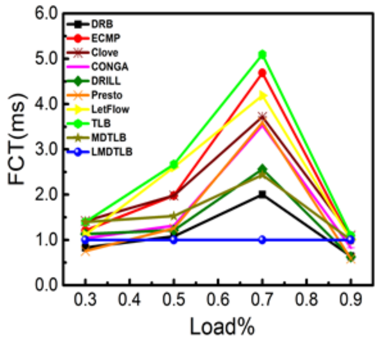
(c) Overall FCT for data mining of both short and long flows.



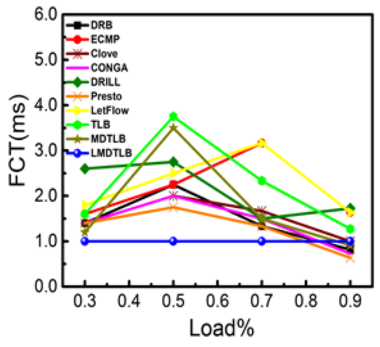
(d) FCT for web-search short flows.



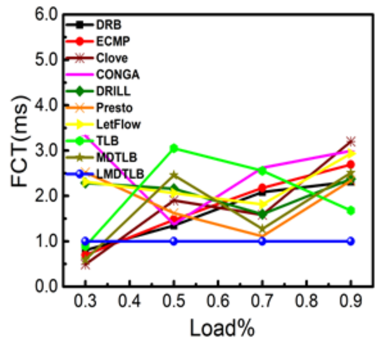
(e) FCT for web-search long flows.



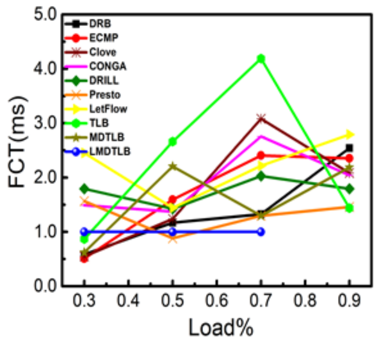
(f) Overall FCT for web-search of both short and long flows.



(g) FCT for short general flows.



(h) FCT for long general flows.



(k) Overall FCT for both short and long general flows.

Fig. 3. FCT generated for each of the protocols.

Figures 4 to 6 were generated for analysing the throughput of LMDTLB, MDTLB, and other protocols. The throughput generated for data mining flows is shown in Fig. 4, for web-research flows in Fig. 5, and for general flows in Fig. 6. Figures show that both MDTLB and LMDTLB are among the highest average throughput protocols. Furthermore, the LMDTLB achieves higher average throughput compared to MDTLB for web search and general flows as it is shown in Fig. 5. However, in Fig. 6, has achieved the highest average throughput for general flows among all protocols. This proves the superiority of LMDTLB over MDTLB. In Fig. 4, it can be observed that MDTLB has achieved higher average throughput but with very small amount than LMDTLB, at the same time, both of them are competitive with other protocols in terms of average throughput for data mining flows.

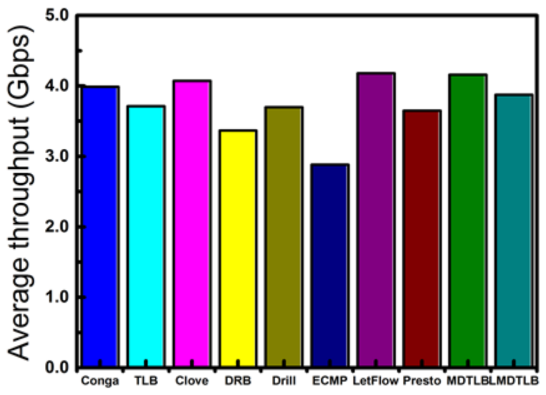


Fig. 4. Average throughput for the data mining flows.

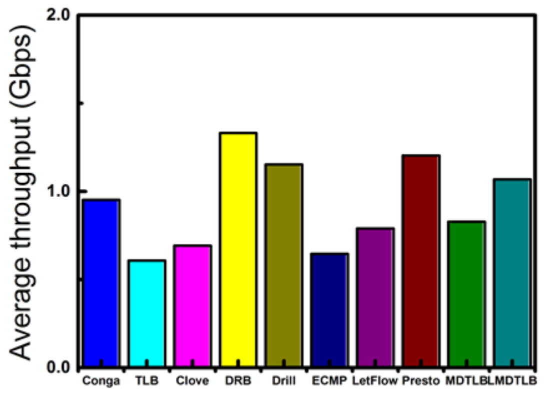


Fig. 5. Average throughput for the web-research flows.

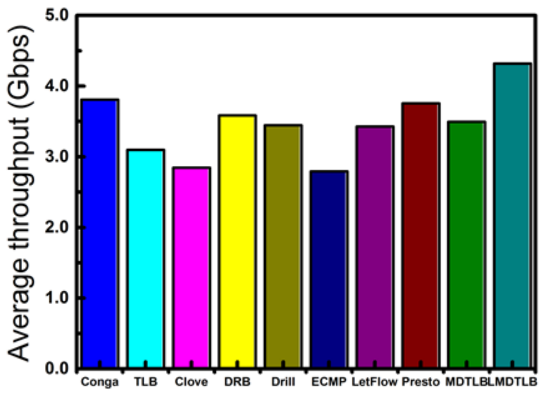
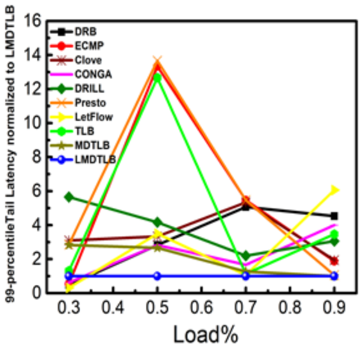
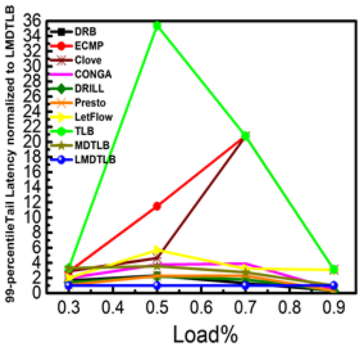


Fig. 6. Average throughput for the general flows.

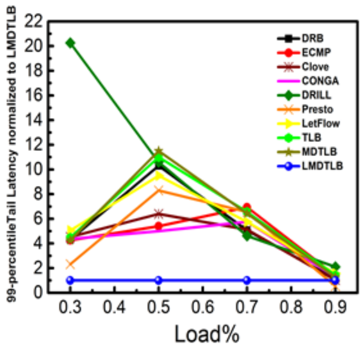
In Figs 7(a) to (c) the 99-th percentile tail latency normalized to the tail latency of LMDTLB is used for better results visualization. The 99-th percentile is derived from [24]. Obviously, LMDTLB has achieved lower tail latency comparing with other protocols for all flow types. In the category of data mining flows, it is observed that LMDTLB protocol has featured with obvious lower tail latency than other protocols especially at the load of 0.5 as shown in Fig. 7(a). Figure 7(b) shows that LMDTLB is still better than other protocols especially at the loads of 0.5 and 0.7 there are big differences with TLB, ECMP, and Clove protocols for web-search flows. Figure 7(c) confirms that LMDTLB is the best in terms of tail latency among all protocols for general flows. It also clearly observed that at 0.9 high workload there are similar values of tail latency for almost all protocols, with some differences because of random behaviour.



(a) Data mining flow.



(b) Web-search flow.



(c) General flows.

Fig. 7. The 99th percentile tail latency normalized to LMDTLB.

5. Conclusion and Summary

In this article, the LMDTLB was proposed to delay the rerouting decision a few packets for every flow when it needs to change its path in order to give the network more time to assure that the very bad path condition is not temporary, that can help in reducing the rerouting of flows which help in the reduction of the packet reordering effect and improve the tail latency. The tail latency was analysed for LMDTLB and other load balancing routing protocols. Results have shown that LMDTLB has achieved lower tail latency comparing with other state of the art load balancing routing protocols for both data mining and general flow. Furthermore, LMDTLB has achieved lower tail latency than five out of eight load balancing protocols for web search. We try to conclude which protocol has achieved better performance in the aspect of limiting tail latency for three types of flows: data mining, web search, and general flows. The finding is that LMDTLB was the best in terms of achieving lower tail latency and FCT. The future work is to convert the system to an automatic tuning system based on adaptive neuro-fuzzy inference system. Another future work is to evaluate the system based on per-flow granularity.

Nomenclatures

f	Flow
$f.rf$	Flow rerouting threshold
$f._{ss}$	Flow sent size
f_{ret}	Fraction of path’s retransmission events
if_{tout}	Set if the flow experiences a timeout
$n_{f,tout}$	Number of times out that the flow experiences
$n_{p,tout}$	No. of path’s timeout events
p	Path
$p.rp$	Path with the smallest sending rate
$P*$	Optimum path
R	Rerouting threshold
r_s	Sending rate of the flow
S	Size Threshold, byte
S_s	Size sent from the flow, byte
T_{ECN}	Threshold for Fraction of ECN
T_{RTT_HD}	Dynamic Threshold for high RTT
T_{RTT_HS}	Static Threshold for high RTT
T_{RTT_LD}	Dynamic Threshold for low RTT
T_{RTT_LS}	Static Threshold for low RTT
type	Path condition, Very Good, Good, Grey, Bad, Very Bad

Greek Symbols

Δ_{ECN}	Threshold for notably better ECN fraction
Δ_{RTT}	Threshold for notably better RTT

Abbreviations

CONGA	Cluster-Overlap Newman Girvan Algorithm
DCN	Data Center Network
DRB	Digit-Reversal Bouncing
ECMP	Equal Cost Multi Path
ECN	Explicit Congestion Notification
FCT	Flow Completion Time
FEC	Forward Error Correction
LMDTLB	Multi-Level Dynamic Traffic Load Balancing

MDTLB	Lazy Multi-Level Dynamic Traffic Load Balancing
P4	Programming Protocol-independent Packet Processors
QDAPS	Queueing Delay Aware Packet Spraying
RTT	Round Trip Time
TLB	Translation lookaside buffer

References:

1. Hu, H.; Wang, Y.; Yang, L.; Komlev, P.; Huang, L.; Chen, X.S.; Huang, J.; Wu, Y.; Merchant, M.; and Sacheti, A. (2018). Web-scale responsive visual search at bing. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* London, United Kingdom, 359-367.
2. Wang, Z.; Liao, J.; Cao, Q.; Qi, H.; and Wang, Z. (2014). Friendbook: a semantic-based friend recommendation system for social networks. *IEEE Transactions on Mobile Computing*, 14(3), 538-551.
3. Aggarwal, S.; Mahajan, N.; Kaushal, S.; and Kumar, H. (2018). Load balancing and clustering scheme for real-time VoIP applications. *Advances in Computer Communication and Computational Sciences.* Singapore: Springer, 451-461.
4. Chen, L.; Yi, B.; Chen, K.; Kim, C.; Zheng, K.; and Liu, F. (2018). Cutting tail latency in commodity datacenters with cloudburst. *Proceeding of the 26th IEEE International Conference on Network Protocol (ICNP).* Cambridge, UK, 99-109.
5. Xiao, K. (2018). *Congestion control and resource allocation in emerging wireless networks*. Alabama: Auburn University.
6. Al-Saadi, R.; Armitage, G.; But, J.; and Branch, P. (2019). A survey of delay-based and hybrid TCP congestion control algorithms. *IEEE Communications Surveys & Tutorials*, 21(4), 3609-3638.
7. Memon, S.; Huang, J.; Saajid, H.; Bux, N.K.; Saleem, A.; and Aljeroudi, Y. (2019). Novel multi-level dynamic traffic load-balancing protocol for data center. *Symmetry*, 11(2), 1-22.
8. Zhang, H.; Zhang, J.; Bai, W.; Chen, K.; and Chowdhury, M. (2017). Resilient datacenter load balancing in the wild. *Proceedings of the conference of the ACM Special Interest Group on Data Communication.* Los Angeles, USA, 253-266.
9. Cao, J.; Xia, R.; Yang, P.; Guo, C.; Lu, G.; Yuan, L.; Zheng, Y.; Wu, H.; Xiong, Y.; and Maltz, D. (2013). Per-packet load-balanced, low-latency routing for clos-based data center networks. *Proceedings Of the CoNEXT '13*: *Conference on emerging Networking Experiments and Technologies*. California, USA, 49-60.
10. Ghorbani, S.; Yang, Z.; Godfrey, P.B.; Ganjali, Y.; and Firoozshahian, A. (2017). DRILL: Micro load balancing for low-latency data center networks. *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. Los Angeles, CA, USA, 225-238.
11. He, K.; Rozner, E.; Agarwal, K.; Felter, W.; Carter, J.; and Akella, A. (2015). Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review*, 45(4), 465-478.

12. Muller, M.P.; Richardson, S.E.; McGeer, A.; Dresser, L.; Raboud, J.; Mazzulli, T.; Loeb, M.; and Louie, M. (2006). Early diagnosis of SARS: lessons from the Toronto SARS outbreak. *European Journal of Clinical Microbiology and Infectious Diseases*, 25(4), 230-237.
13. Katta, N.; Ghag, A.; Hira, M.; Keslassy, I.; Bergman, A.; Kim, C.; and Rexford, J. (2017). Clove: Congestion-aware load balancing at the virtual edge. *Proceedings of the 13th International Conference on emerging Networking Experiments and Technologies*. Seoul/Incheon, South Korea, 323-335.
14. Handigol, N.; Flajslik, M.; Seetharaman, S.; McKeown, N.; and Johari, R. (2010). Aster* x: Load-balancing as a network primitive. *Proceedings of the 9th GENI Engineering Conference*. Washington, D.C, USA, 1-2.
15. Vanini, E.; Pan, R.; Alizadeh, M.; Taheri, P.; and Edsall, T. (2017). Let it flow: Resilient asymmetric load balancing with flowlet switching. *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)* . Boston, MA, USA, 407-420.
16. Bai, W.; Chen, L.; Chen, K.; and Wu, H. (2016). Enabling ECN in multi-service multi-queue data centers. *Proceedings of the 13th USENIX Symposium on Networked Systyms Design and Implementation (NSDI '16)*. Santa Clara, CA, USA, 537-549.
17. Geng, Y.; Jeyakumar, V.; Kabbani, A.; and Alizadeh, M. (2016). Juggler: a practical reordering resilient network stack for datacenters. *Proceedings of the Eleventh European Conference on Computer Systems (ACM)*. London, United Kingdom, 1-16.
18. Montazeri, B.; Li, Y.; Alizadeh, M.; and Ousterhout, J. (2018). Homa: A receiver-driven low-latency transport protocol using network priorities. *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. Budapest, Hungary, 221-235.
19. Wang, P.; Trimponias, G.; Xu, H.; and Geng, Y. (2019). Luopan: sampling-based load balancing in data center networks. *IEEE Transactions on parallel and distributed systems*, 30(1), 133-145.
20. Katta, N.; Hira, M.; Kim, C.; Sivaraman, A.; and Rexford, J. (2016). Hula: Scalable load balancing using programmable data planes. I. *Proceedings of the Symposium on SDN Research (SOSR '16)*. Santa Clara, CA, USA, 1-12.
21. Bosshart, P.; Daly, D.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; and Walker, D. (2013). Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 44(3), 88-95.
22. Huang, J.; Lv, W.; Li, W.; Wang, J.; and He, T. (2018). QDAPS: Queueing delay aware packet spraying for load balancing in data center. *Proceedings of the 26th International Conference on Network Protocols (ICNP)*. Cambridge, UK, 66-76.
23. Greenberg, A.; Hamilton, J.R.; Jain, N.; Kandula, S.; Kim, C.; Lahiri, P.; Maltz, D.A.; Patel, P.; and Sengupta, S. (2011). VL2: A scalable and flexible data center network. *Communications of the ACM*, 54(3), 87-93.
24. Xu, H.; and Li, B. (2014). RepFlow: Minimizing flow completion times with replicated flows in data centers. *Proceedings of the IEEE Conference on Computer Communications* (*IEEE INFOCOM)*. Toronto, ON, Canada, 1581-1589.