Technische Universität München
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Wolfgang Kellerer

Master‘s Thesis

Deliberate Load-Imbalancing in Data Center Networks

Author:	Kreft, Florian
Address:	Adalbert-Stifter-Str. 2
	84424 Isen
	Germany
Matriculation Number:	03639813
Supervisor:	Krämer, Patrick
Begin:	13. March 2020
End:	12. March 2021

Kurzfassung

Das Ziel beim Traffic Engineering in Rechenzentrumsnetzwerken ist es, die hohe Bisektionsbandbreite, die in Rechenzentrumsnetzwerken typischerweise verwendet wird, möglichst effizient zu nutzen. Hierfür werden Lastausgleichsverfahren benötigt. Aktuelle Techniken zur Lastverteilung basieren dabei auf einer gleichmäßigen Aufteilung von Flüssen über verfügbare Pfade zwischen zwei Endpunkten. Hierbei kanne es jedoch zur Kollision eines großen Flusses mit vielen kleinen Flüssen kommen. Dies resultiert in Paketverlusten, die Flusszeit erhöhen. Diese Thesis untersucht, ob eine ungleichmäßige Auslastung in diesen Situationen von Vorteil ist. Größere Flüsse können dann auf die weniger ausgelasteten Pfade gelegt werden und somit Paketverluste vermieden werden. Um dies zu Untersuchen, wird in dieser Masterarbeit ein verteiltes Protokoll entwickelt, im Netzwerksimulator ns-3 implementiert und in Simultationen ausgewertet. Die Ergebnisse zeigen, dass ein gezieltes Ungleichgewicht in zu einer Verringerung der Flusszeiten um bis zu 50% für FatTree Topologien der Größe k=8 bis k=16.

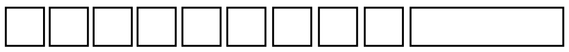
Abstract

The goal in load balancing and traffic engineering in data center networks is to use the large bisection bandwidth and path diversity of data center networks most efficiently. For this, load balancing schemes are needed. The goal of existing approaches is to keep paths approximately equally loaded. Due to the heavy tailed distribution of flow sizes and the bursty nature of flows, keeping links equally loaded can lead to collisions of large with many small flows. Collisions can result in packet loss, which increases the flow completion time. This hurts especially small flows. This thesis investigates whether load imbalancing over paths can mitigate this problem. Large flows can then be forwarded over low utilized paths, mitigating the risk of collisions. To ingestigate this hypothesis, this thesis designs a distributed load imbalancing scheme called IMBAL and implements it in the network simulator ns-3. IMBAL improves FCT up to 50% on fat-tree topologies ranging in size from k=8 to k=16.

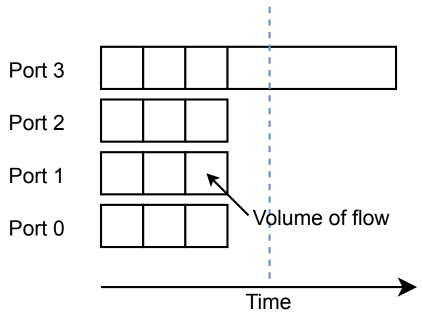
Contents

Chapter 1

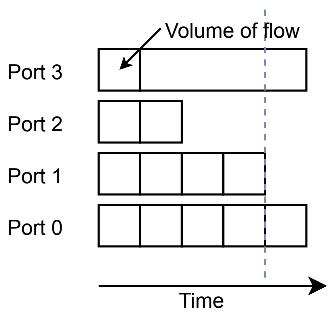
Introduction



(a) Order of arriving flows.



(b) Load Balancing.



(c) Load Imbalancing.

Figure 1.1: Imbalancing load over ports can reduce job completion times. Blue line indicates the shortest amount of time to finish all jobs.

Multi-rooted trees are a popular topology for data-center networks [ZMSG19]. They provide large bisection bandwidth and a large number of paths between hosts in the network [AFLV08]. Effectively balancing traffic over those paths is a critical ability to fully utilize path diversity and bandwidth. A common strategy to load-balancing is minimizing the maximum link utilization [KHK$^+$16, CKS14, AED$^+$14, KHG$^+$16, HTB$^+$20, ZTZ$^+$14, SSIF13, BAAZ11, AFRR$^+$10, BK19, LZW$^+$20, VPA$^+$17], i.e., keeping links approximately equally loaded. The goal of this strategy is twofold: 1) reduce congestion, and as a consequence 2) reduce flow completion time. Where the latter is the actual objective of those strategies. Minimizing the maximum link utilization is thus an intermediate optimization objective with the goal to improve FCT. The online nature of the problem, i.e., balancing flows without the knowledge of all flows, as well as the heavy tailed nature of flow and flowlet sizes [HRA+15] can lead to detrimental effects, though [FW00, Alb09]. Especially in the presence of coflows, a set of flows where the completion time of the slowest flow, the so-called makespan [Alb09], is important. Coflows arise frequently in data-center workloads [CZS14].

Consider the example in Fig. 1.1. Fig. 1.1a shows the order in which ten flows to the same destination arrive almost at the same time at a switch. The size of the rectangles in Fig. 1.1 indicates the volume of each flow: Nine flows are small and one flow is larger with four times the volume. A mix like this is not uncommon in data-center networks [AED+14]. The switch balances the flows as they arrive over its four ports. The switch does not know about flows in advance, i.e., operates in an online scenario. Minimizing the maximum link utilization with e.g., the least loaded first heuristic [Alb09] can result in the situation in Fig. 1.1b. The large flow is assigned together with three small ones to the same port. This assignment causes two problems: 1) The large flow can collide with the small flows, resulting in congestion and packet loss similar to ECMP. 2) There is no utility in finishing small flows very fast if the application depends on the completion of the large flow as well [CCBA16]. A better solution in this situation would be deliberate *imbalancing* of load. A strategy well known in the field of online scheduling [Alb09]. Imbalancing can result in the situation in Fig. 1.1c. The small flows are distributed unequally over the ports of the switch. As a result, the large flow collides with only one small flow, and the time until all flows are finished is reduced compared to Fig. 1.1b.

Recent in-network load balancing techniques [KHK+16, KHG+16, AED+14, HTB+20] miss this opportunity and can suffer from collisions and stragglers. Techniques that specifically optimize for FCT [AYS+13, CLCL18, AHLPMY+19] reduce congestion, but in the presence of coflows result in stragglers. Flow scheduling techniques that specifically target the coflow case either require advanced knowledge about coflows [ARN+18, CZS14, BKA+20, CCBA16] or complex control planes to detect coflows at runtime [CS15, ZCY+16].

The challenge in maintaining an imbalanced schedule, i.e., keeping links differently utilized, lies in the fact that to achieve this, the size of flows must be known at decisions time. Advanced knowledge of flows cannot be assumed common knowledge, though [DJK+19]. Not even the sending application might know the exact size of a transmission ahead of time. An in-network load balancer that wants to maintain an imbalanced schedule in order to reduce the risk of congestion and stragglers in co-flows must thus not rely on advanced knowledge of flow-sizes.

This thesis proposes IMBAL, an in-dataplane load balancing technique that keeps links purposefully imbalanced and does not require a-prior knowledge of flow sizes. In contrast to existing work, IMBAL optimizes for makespan, i.e., IMBAL treats all flows in the network as belonging to one large coflow and optimizes for makespan instead of FCT or maximum link utilization. IMBAL operates on the granularity of flowlets and uses a distance-vector like protocol to distribute global network state through the network. IMBAL is implemented and evaluated in the network simulator ns-3 and compared against HULA [KHK+16] and ECMP. Simulations show that IMBAL reduces congestion as well as FCT by up to 50 % on real world traffic traces.

Contributions of this thesis are:

1. This thesis proposes the imbalancing of traffic as an alternative objective to existing load balancing techniques that optimize for maximum link utilization.
2. IMBAL is designed in this thesis. IMBAL is a load imbalancer that does not require prior knowledge about flow-sizes and can be implemented on programmable network devices.
3. In the scope of this thesis, IMBAL is implemented in the network simulator ns-3.
4. The effectiveness of IMBAL is shown in large scale simulations and compared against state-of-the art in-network load balancing algorithms of similar complexity.

This thesis is organized as follows: Sec. 2 gives background on data center characteristics and challenges online scheduling, different objectives and their relation to networking. Further, Sec. 2 introduces HULA, a hop-by-hop utilization-aware load balancing architecture. Sec. 3 discusses related work. Sec. 4 gives an overview over current scheduling techniques employed in data center networks. Sec. 5 details the implementation of IMBAL. Sec. 6 compares IMBAL against HULA and ECMP in large scale simulations on real production workloads. Sec. 7 concludes this thesis.

Chapter 2

Background

This chapter contains information necessary to understand the motivation of this thesis. Section 2.1 introduces data centers regarding its typical characteristics. Section 2.2 introduces challenges in traffic engineering for data center networks. Section 2.3 defines online scheduling in general. Section 2.5 explains of HULA, which provides the basic structure and functionality for the IMBAL scheduling algorithm proposed on Section 5.2 in the context of network scheduling.

2.1 Data Center Characteristics

A data center network refers to a cluster of servers. Those servers transmit a large amount of data. To avoid single point congestion, data center design takes advantage of multiple paths between its hosts. Due to the high traffic demands, data center networks feature a large bisection bandwidth. A popular architecture are multi-rooted trees like FatTrees [AFLV08]. Figure 2.1 shows a FatTree topology. The topology consists of so-called pods and a spine. A pod, consists of ToR and aggregation switches. ToRs connect to end-hosts. aggregation switches provide connectivity within the pod. The spine interconnects the pods.

The specific traffic demands depend on the application. Figure 2.2 shows two flow size distributions for web and data mining workload [KHK$^+$16], optained empirically from production datacenters. Of note is the heavy tailed distribution of the flows: most flows are small, while a small number of large flows contribute to a substantial portion of the traffic.

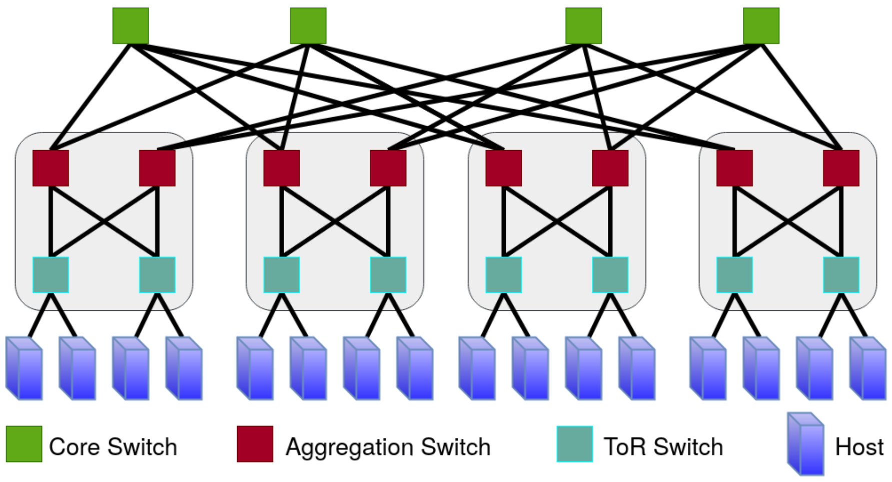


Figure 2.1: FatTree for k=4

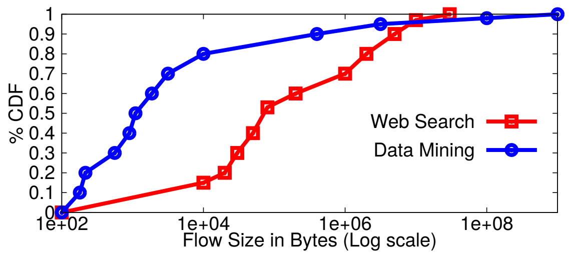


Figure 2.2: Empirical traffic distribution in data center networks [KHK+16]

2.2 Challenges in Data Center Traffic Engineering

Existing data center topologies struggle to provide enough capacity between the servers they interconnect [BAAZ11]. Since current data centers are built from high-cost hardware, the higher tiers of the tree are oversubscribed to keep the total cost low. Thus, under heavy traffic workloads, data centers are prone to congestion. Data center traffic is volatile and bursty [AED+14]. Therefore, mechanisms aiming to distribute load on the network should be higly responsive in its reaction to congestion. This is in contrast to traffic engineering techniques designed for internet service providers, which function on coarser timeframes [BAAZ11].

Network mechanisms providing should be oblivious to the transport protocol at the end-host (TCP, UDP, etc). Specifically modification of TCP should be avoided because of deployment complexity.

2.3 Online Scheduling

In online scheduling, jobs arrive one by one over time as a sequence $I = J_1, ..., J_n$. Upon arrival, each job must be scheduled to one of m machines without prior knowledge of subsequent jobs. For each job, the processing time p_i is known.

This thesis interprets load distribution on a network as a form of non-preemptive online scheduling. For this thesis, packet flows are treated as jobs which need to be distributed among paths in the network. Ports of switches are treated as machines to which arriving packets are scheduled. For more specifics on this, see Section 5.1. Scheduled jobs cannot be stopped or paused [Alb09], which is inline with networking needs: every switch needs to schedule packets as they arrive to avoid packet drops due to high queue sizes.

2.4 Makespan minimization

The goal of makespan minimization is minimize the completion time of the last job that finishes in the schedule. For online scheduling, this assumes immediate and permanent allocation of a job to a machine [Alb09].

The optimal assignement OPT of the jobs seen in Fig. 2.3, where size is proportional to needed processing time, can be seen in Fig. 2.4.

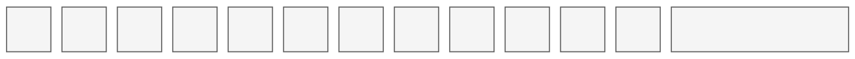


Figure 2.3: Sequence of jobs in order of arrival.

In online scheduling, the goal for makespan minimization algorithms is to get the worst case performance as close as possible to the OPT. The difference in performance is denoted by the *competitive ratio* [FW00].

Makespan minimization under the restrictions mentioned above for online scheduling was tackeld by Graham [Gra66]. His LIST algorithm operates as follows:

Let *load* of a machine be the sum of processing times of all jobs currently assigned to one machine. Then, simply assign every new arriving job to the least loaded machine. This results in a competitive ratio of $2 - \frac{1}{m}$ [Alb09].

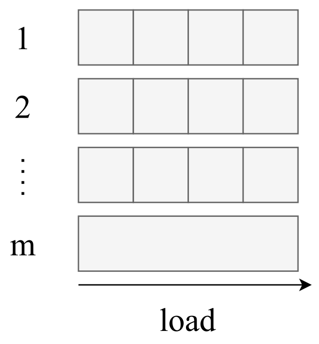


Figure 2.4: Optimal assignment of the jobs to 4 machines.

2.5 HULA

HULA [KHK+16] is an approach which aims to solve the challenges outlined in 2.2 by distributing path utilization information over the network. Each switch uses this information to make local routing decisions. Every switch maintains a list of best ports for every destination based on least path utilization.

As in HULA, information regarding the reachability and path utilization is transmitted via probes. Probes are mall packets that inform the switches in the network about the utilization of paths. Every ToR switch sends out these probes to all connected switches periodically. This way, every switch in the network receives information by way of reverse propagation.

Each switch thus learns two things: 1) which ToR switches can be reached via each port, and 2) the utilization of the least utilized path via the neighboring switch.

Availability can be learned via time-outs. If a switch did not receive a probe from a ToR over a port for some time, the switch assumes that no path to that ToR exists. This way, a failing link is recognized and traffic will be rerouted at the switch. And information about the now potential higher load on the remaining load will be propagated throughout the network.

An advantage of this approach is the abstraction of path loads to ports: every switch only stores the next best hop for each destination, keeping the forwarding state to a minimum.

2.5.1 Flowlet based forwarding

HULA employs flowlet routing. Flowlets are parts of flows which are packets with an inter-arrival-time smaller than a specified threshold in the order of the network round trip time [KHK+16]. Flowlets based routing is meant to allow for a more fine-grained load-balancing than flow based routing, while still minimizing packet re-ordering at the receiver [KKSB07]. To accomplish flowlet based routing in HULA, every switch needs to maintain a next best hop for that flowlet as well as a timestamp indicating the last arrival of a packet for every flowlet.

When a non-probe packet arrives (e.g. data or TCP control packets), HULA switches firstly check if there is an active flowlet for that packet. If there is, its output port is used for forwarding and its timestamp is updated. If there is not, new entries for both are generated with the hash of the flow as a key. This way, packets belonging to one flowlet get routed on the same output port for every switch, until the inter-arrival gap crosses the threshold. In this case, even if more packets for the same flow need to be routed, the risk of reordering is minimized.

2.5.2 Optimization objective

The forwarding decisions in a data center network can be interpreted as an online scheduling problem. For HULA, we view data flows as jobs and the admissible output ports on a switch as machines. HULA can be seen as a form of Grahams LIST algorithm: new flowlets are assigned to the least loaded output port. ”Least loaded” in this case means ”leads to a path with the least expected utilization”. The information for the least expected utilization is propagated to each switch in the form of small pakets called probes. HULA switches schedule packets on the port with the lowest path utilization based on the information provided by incoming probes over all paths. This way, routing utilizes multipath diversity and adapts to changes in traffic load, at the cost of a small overhead (probe packets distributing the information).

Chapter 3

Related Work

Hedera [AFRR$^+$10] uses a centralized flow-scheduler that re-assigns large flows to less congested paths, thus implementing a similar strategy as IMBAL. In contrast to Hedera, IMBAL is distributed and operates at millisecond granularity. MicroTE [BAAZ11] estimates traffic matrices and uses a bin packing heuristic to reduce maximum link utilization, unpredictable traffic is routed with ECMP. FastPaths [BDS$^+$21] designs a new routing scheme based on flowlets for non-clos data-center topologies. Flier [KS17] implements load balancing in direct connect data-centers in a host based approach, where host decide whether to forward packets along longer or shorter paths. SPAIN enables multi-pathing in arbitrary topologies by computing trees and mapping those to VLAN tags. Trees are computed over pre-selected paths. Planck [RSD$^+$14] implements a measurement system that enables milisecond scheduling decisions without programmable hardware.

In contrast to central schedluers[AFRR$^+$10, BAAZ11] is IMBAL distributed. IMBAL is not designed for arbitrary topologies but targets Clos Networks. Further, IMBAL targets makespan minimization.

Chiesa et al. show the conditions under which ECMP is optimal [CKS14].

Flux [DJK$^+$19] estimates flow sizes at line rate in data center networks. They show that advnaced knowledge of flow-sizes is generally not a plausible assumptions, but that due to the repritive structure of jobs in DCs flow sizes might be learned.

RCP [DM06] is a protocol that approximates processor sharing in the presence of feedback delay. The authors argue why FCT is the metric one should care for.

3.1 In-Network Load Balancing

Spotlight [ACX$^+$20] implements a distributed L4-load taking the end-host utilization into consideration. The goal is FCT minimization, the objective is the even distribution of loads over end-points. CONGA [AED$^+$14] uses per-path congestion information to forward flowlets over the currently least utilized path. HULA [KHK$^+$16] takes a similar approach and improves scalability, performance and deployability through better disemination of congestion information in the network, per-hop forwarding decisions and programmable hardware. CONTRA [HBC$^+$20] generalizes HULA to arbitrary topologies and provides a high-level language to easily implement forwarding policies such as least loaded first, WCMP, etc. [HTB$^+$20] implements adaptive WCMP, where weights are updated adaptively to the current load situation at line rate using programmable data planes. TCP-Path [AHLPMY$^+$19] uses SYN-flooding for each new flow to discover the currently fastest path in the network. The SYN-packets determine the currently fastest path between two end-hosts. FlowDyn [BK19] improves flowlet based load balancing approaches by adapting the flowlet threshould through measurements in the network, improving the trade-off between minimizing packet reordering and improving through frequent switching. PDQ [HCG12] implements a distribted algorithm that allows switches to collaboratively gather information about flows and converge to a stable allocation decision, as well as allow preemption of jobs, i.e., switchovers. PDQ is able to implement scheduling strategies such as SJF or Earliest Deadline First. FLARE [KKSB07] forwards bursts of packets, i.e., flowlets in WANs. CLOVE [KHG$^+$16] pulls load-balancing into the network hypervisor and balances flows over available paths using different source ports. Thus, hardware in the network can be left unchanged. PLB [LZW$^+$20] uses only partial probing of the network topology, thus reducing probing overhead. LocalFlow [SSIF13] optimizes maximum link utiliziation by aggregating traffic to one host on a switch and then splitting this traffic over the outgoing ports. LetFlow [VPA$^+$17] investigates in detail the advantage of flowlet switching and design a simple randomized load balancer based on those insights. WCMP [ZTZ$^+$14] adapts the sending rates of ECMP to reflect asymmetries in the topology.

IMBAL is similar in that it provides in-network based load balancing and uses probing to distribute global load information in the network. In contrast to previous in-network based load balancing schemes targets IMBAL makespan minimization and for this purposefully imbalances traffic over available links. Previous appraches, implicitly or explicitly, optimize for maximum link utilization.

3.2 Co-Flow scheduling

In Sincronia [ARN$^+$18], applications inform the network fabric about their co-flows. The network fabric uses this information to sort and schedule co-flows and the constituting flows using priority queuing in order to minimize co-flow completion time. Varys [CZS14] uses a centralized co-flow scheduler and knowledge about co-flows to pace flows in a co-flow such that every flow finishes at the same time as the slowest flow in the co-flow, using available resources more efficiently. Aalo [CS15] does not assume prior knowledge about co-flows and uses a centralized entity to estimate co-flows, and distributed entities that assign rates and priorities based on those estimations. Aalo approximates least attained service first strategy. CODA [ZCY$^+$16] uses a complex control plane to detect co-flows without prior knowledge, i.e., at runtime. pCoFlow [BKA$^+$20] maintains flow-affinity then re-priotizing coflows to take the packets in flight into account with the goal to reduce packet-reordering caused by priority changes with multiqueue switches. Karuna [CCBA16] improves pFabric by prioritizing flows in such a way that they finish just before their deadline, thus leaving room for non-deadlined flows, improving overall network utilization.

IMBAL is similar to approaches in co-flow scheduling in that IMBAL does not optimize for completion times of individual flows. IMBAL also allows some flows to take longer than strictly necessary. In contrast to co-flow scheduling approaches, IMBAL considers all flows in the network to belong to one big job, i.e., IMBAL does not have an explicit notion of co-flows. Further, IMBAL does not require any prior information on flow-size, co-flow characteristics and does not use rate limiting.

3.3 Host-based load balancing

DCTCP [AGM$^+$10] is a variant of TCP designed for data-centers that uses the Explicit Congestion Notification (ECN) to indicate congestion. Sources use the ECN bit to estimate congestion and adjust sending rate. HULL [AKE$^+$12] improves DCTCP with pacing and by leaving head-room on switches, i.e., not fully utilizes links, in order to not hold up small latency sensitive flows. pFabric [AYS$^+$13] uses strict priority queueing, where end-hosts indicate the priority of traffic in packet headers. Sending rates are lazily adjusted to avoid packet loss. PIAS [BCC$^+$14] uses MLFQ and sieving to approximate shortest job first without prior information on the volume of flows. Packets are tagged with priorities on the end-hosts based on the amount of already-send traffic. The objective of PIAS is to minimize flow-time, i.e., FCT of individual flows without notion of dependency between flows. AuTO [CLCL18] uses deep reinforcement learning to learn and adapt thresholds for sieving used in PIAS. DCMPTCP improves multipath TCP for use in the data-center environment. DCMPTCP does not use multiple flows for small flows and intra-rack traffic. DCMPTCP further synchronizes ECN between sub-flows, i.e., adapting sending rates of multiple sub-flows in step. pHost [GNK+15] decouples the network fabric from scheduling decisions by allowing end-hosts to make per-packet scheduling decisions. QJump [GSG+15] uses rate limitation and strict priority queueing to reduce tail latencies. Presto [HRA+15] splits flows into small flow-cells and extends packet-reordering capabilities at end-hosts, enabling packet spraying and thus reaping optimality condition of ECMP. Homa [MLAO18] uses receiver driven scheduling of flows, where receivers tell the senders priorities, when to send, etc. Fastpath [POB+14] implements a central scheduler that assigns to every packet in the network the path it should take and the time at which it should be transmitted. Fastpath reduces queue occupancy and reduces TCP retransmissions. Raiciu et al. [RBP+11] evaluate in depth the usability of MPTCP in DCs and propose small modifications to DC networks to improve the usability of MPTCP in DCs. ICON [RCAV19] assumes knowledge about applications and uses packet pacing at the sender to avoid packet drops due to congestion in incast scenarios. ResQueue [RV20] uses flow-size to prioritize flows, and also prioritizes re-transmitted packets, resulting in better FCTs in in-cast scenarios, where also re-transmitted packets get frequently dropped due to collisions. ALB [SWFX19] uses latency information to assign flows to paths. Flows are assigned to paths by choosing among a specific set of ports that result in the desired allocation with ECMP. SPLB [XYZ+20] scoutes for each packet the best possible path to maintain packet order. The reserve a part of the bandwidth for each link for stand-in packets that scout for the actual data packets. DeTail [ZDM+12] uses cross-layer information to reduce long tail FCT.

In constrast to host based methods is IMBAL solely located in the network. Further, IMBAL does not use priority queueing or rate limiting. IMBAL targets makespan minimization, while most of the previous work in this area focuses on shortest job first, not taking the co-flow nature into account. IMBAL is orthogonal to work that targets incast scenarios. IMBAL can be combined with those approaches. Further, IMBAL could be implemented with approacehs such as CLOVE.

Chapter 4

Scheduling in datacenter networks

This Chapter summarizes currently explored scheduling methods in data center networks.

4.1 Shortest Job First

To optimimize FCT, traffic engineering algorithms in data center networks approximate shortest job first algorithms (SJF), that are known to be optimal for the flow-time objective. Examples are AuTO [CLCL18], PIAS [BCC$^+$14] and others.

The problem with SJF optimization: Long flows get starved. Jobs can consist of coflows, meaning a group of flows which are only considered finished on completion of the last flow. In a scenario like this, it is not relevant if all short flows are finished but a long one is still pending. So, only focusing on FCT of individual flows can be suboptimal[CZS14].

4.2 Max Link utilization

Another frequent objective is the minimization of the maximum link utilization. Examples are HULA and CONGA [AED$^+$14] that implement a greedy algorithm of admitting flows to the currently least loaded path. Other algorithms such as ECMP, Presto and WCMP implicitly optimize for this, other schemes sucha as MicroTE [BAAZ11] directly optimize for it with an LP.

On an already loaded switch, equally loaded paths are the worst case for LIST algorithm if a large flow needs to be scheduled. This also leads do collissions of flows.

4.3 Makespan Minimization

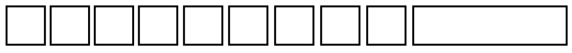
For makespan minimization mentioned in Section 2.4, there is not yet direct correspondence for data center networks. The closest is co-flow optimization. Coflow scheduling tries to optimize makespan for a specific set of flows, namely those of one application. The difficulty lies in knowing these coflows, their sizes, and all other coflows etc. It is impossible to know all of this in advance. Learning it at runtime is difficult and requires complex control planes. A possible solution is to treat all flows as one huge coflow.

Chapter 5

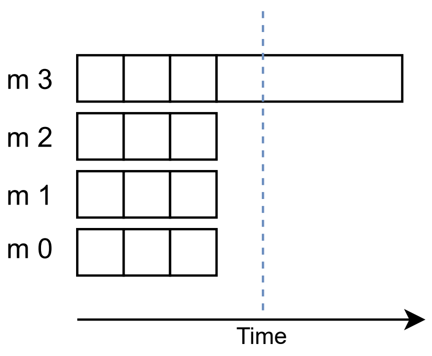
Design and Implementation

This Chapter describes the design and implementation of IMBAL. Section 5.1 describes the MR algorithm, which provides the scheduling scheme used in IMBAL. Section 5.2 explains the general structure of IMBAL, while Sections 5.4 and 5.3 go more in depth on its scheduling and probing mechanisms. Finally, Section 5.6 explores possible realization problems due to computational complexity.

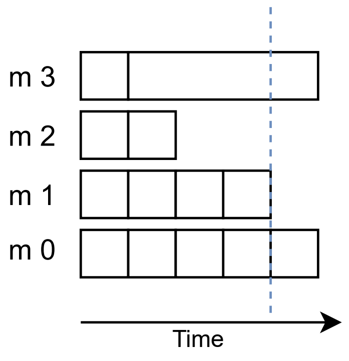
5.1 IMBAL scheduling



(a) Order of arriving jobs.



(b) Load Balancing.



(c) Load Imbalancing.

Figure 5.1: Imbalancing load over ports can reduce job completion times. Blue line indicates the shortest amount of time to finish all jobs

MR is an optimisation approach for online scheduling as described in 2.3. The MR algorithm [Alb09] is a scheduling algorithm which aims to improve on the Grahams List algorithm (see Section 2.3) for scheduling. The List algorithm schedules jobs on the least loaded machine, which is the scheduling approach employed by HULA. In contrast, MR aims to maintain an imbalanced schedule, which improves the competitive ratio for online scheduling in comparison to LIST. This improves worst case performance for online job scheduling. An example for a scenario were an imbalanced schedule is beneficial can be seen in Figure 5.1: Shown are several jobs (5.1a) arriving almost at the same time, but chronologically from left to right. In this example, there are 4 possible machines to assign the new flows to. As described in 2.3, the job sizes of future jobs are unknown. Therefore, the theoretical optimal assignment of jobs to minimize makespan, as shown by the blue line, is impossible to achieve reliably. The assignment of jobs to the least loaded machines can lead to a the subotpimal distribution as shown in 5.1b. If the scheduling mechanism attempts to keep an imbalanced schedule, the assignment could look like 5.1c. Here, the worst case has been avoided be leaving one machine lightly loaded, allowing the large job at the end to be processed faster.

Algorithm 1 shows the job assignment process of MR. MR operates on a list of machines M that are sorted ascending based on their load, a list of already scheduled jobs J, and produces a scheduling decision for a newly arriving job j_t. The job j_t is either scheduled on the least loaded machine, or on the machine with the $(k+1)$th load. MR first calculates the average load λ on all machines in Line 4 of Algorithm 1. MR then compares the load of machine $2k+1$ to the average load to decide if the schedule is imbalanced. If the corresponding condition in Line 5 is true, the schedule is imbalanced and the job is scheduled to the least loaded machine. If the condition is false, then the schedule is not imbalanced, and MR considers to assign job j_t to the machine with the $k+1$st smallest load. To decide this, MR calculates the average load of all jobs including j_t in Line 8, and the load l_{k+1} on machine $M[k+1]$ if j_t would be assigned to it in Line 9. MR then compares l_{k+1} to L in Line 10. If the condition is true, then the job is placed on machine $k+1$. If the condition is false, then the assignment is deemed risky and the job is placed on the least loaded machine [FW00].

Related to data center traffic engineering, this algorithm could be used on switches to schedule flows while taking unknown future flows into account. If the size of flows is treated as jobsize, a mix of jobs as shown in 5.1 is not uncommon in data-center networks [AED$^+$14].

Due to the imbalance in the schedule, IMBAL can assign larger flows to less utilized links, thus avoiding collisions of flows that occur with the minimization of the maximum link utilization. IMBAL achieves this at the cost of potentially

```
Data: Machines sorted based on load
      $M \leftarrow [m_1 \ldots, m_n \mid \text{load}(m_i) \leq \text{load}(m_{i+1})]$; Already scheduled Jobs
      $J \leftarrow [j_1, \ldots, j_h]$; Job $j_t$ to schedule
Result: Machine to schedule $j_t$ on.
1  $c \leftarrow 1 + \sqrt{\frac{1+\ln 2}{2}}$;
2  $k \leftarrow \left\lfloor n \cdot \frac{2(c-1)^2-1}{c} \right\rfloor + 1$;
3  $\alpha \leftarrow \frac{2c-3}{2(c-1)}$;
4  $\lambda \leftarrow \frac{1}{k} \sum_{i=1}^{k-1} \text{load}(M[i])$;
5  if $\lambda \leq \alpha \, \text{load}(M[2k+1])$ then
6      return $M[0]$;
7  else
8      $L \leftarrow \frac{1}{n} \left( \sum_{i=1}^{h-1} \text{load}(J[i]) + j_t \right)$;
9      $l_{k+1} \leftarrow \text{load}(M[k+1]) + \text{load}(j_t)$;
10     if $l_{k+1} \leq cL$ then
11         return $M[k+1]$;
12     else
13         return $M[0]$;
14     end
15 end
```

Algorithm 1: Pseudocode for MR algorithm.

delaying shorter flows compared to the flow time objective. However, this drawback is offset in part by the existence of coflows that benefit from the assignment behavior.

Due to the reliance of IMBAL on the load of a job, IMBAL cannot readily be used as scheduling algorithm for flows, since the volume of flows is generally not available. Further, IMBAL is designed for individual machines. The utilization of a port on a switch might not correctly reflect the utilization of the path. A lowly utilized port can be the beginning of a path with a heavily congested link. Section 5.2 will explain how IMBAL can be used as load-imbalancer despite those restrictions.

5.2 IMBAL design

HULA as described in [KHK+16] provides the basis for the IMBAL extensions described in 5.1.

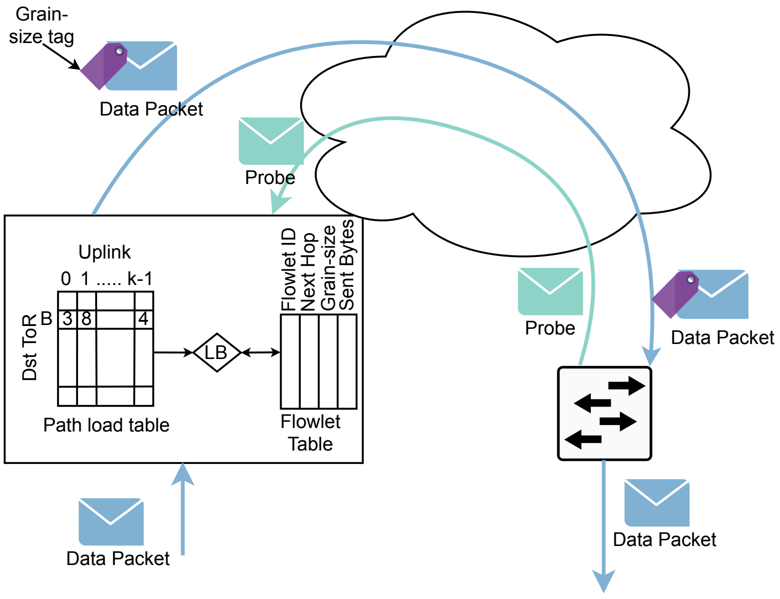


Figure 5.2: System diagram of IMBAL

Figure 5.2 shows the system diagram of IMBAL. Every switch in IMBAL has a path load table, a flowlet table and the logic that makes forwarding decisions. The rows of the path load table are indexed by ToR IDs. The columns of the path load table correspond to the neighbors of a switch. Each entry in the path load table contains the load of the least loaded path with which the destination ToR can be reached via the corresponding neighbor. The necessary load information is distributed in the network by means of special probe packets, similar to other in-dataplane load balancing schemes [KHK$^+$16, HBC$^+$20, LZW$^+$20].

The flowlet table contains for each flowlet the next hop and the flowlet grain size. The grain size of a flowlet is used to estimate the volume. The estimate is based on the already sent data. The ToR switch tags the corresponding data packets with the current grain size. The destination ToR removes the grain-size tag again.

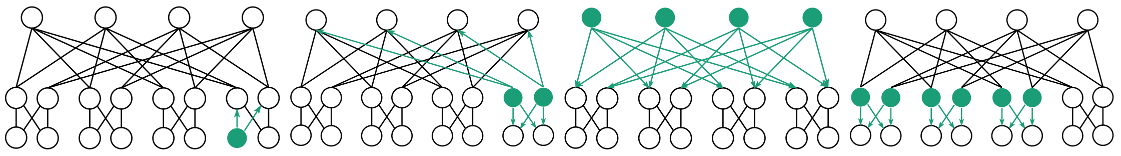
IMBAL adopts the following properties from HULA:

1. **Non-clairvoyance.** To make IMBAL easily and broadly deployable, IMBAL should not explicitly rely on separate data-sources beyond what is observable on switches. That is, IMBAL should not rely on complex control planes or estimation techniques to identify traffic characteristics but on information provided by arriving data packets and probes as in HULA.
2. **In-network.** IMBAL should not make changes to end-hosts necessary. That is, IMBAL should not require custom operating systems or kernel modules on end-hosts in order to operate.

Furthermore, in contrast to HULA, the goal of IMBAL is **minimizing makespan**, i.e., optimize for the completion time of a set of flows instead of the FCT of individual flows. To stay in alignment with the above properties, this is performed on a per switch basis.

5.3 Probing

Information regarding the reachability is transmitted via small packets to all ToR switches that need to be reached. In contrast to HULA, which uses port and path utilization as its metrics, MR operates on load. The probes inform switches about the load in the network, where load is the sum of bytes scheduled to ports. The sum of bytes is a rough estimator of processing time, which is used in the MR algorithm. Since this is unknown in the case of live network balancing, the used metric will be flow-size. In fact, flow-size is proportional to processing time. This decision is based on the assumption that every flow is sent with the complete availiable ToR uplink data rate. Another possibility for load would be the rate every flow needs, which is not availiable on arrival of the first packet of a new flow. But MR needs the load of a new job at assignment of this first packet. To have the



(a) ToR - Step 1. (b) ToR - Step 2. (c) ToR - Step 3. (d) ToR - Step 4.

Figure 5.3: Propagation of probes for one source ToR.

information regarding needed rate per flow would need additional configuration on the hosts, and is as such not further explored in this thesis. The investigation of rate as substitute for load is left for future work.

Each switch thus learns two things: 1) which ToR switches can be reached via each port, and 2) the load of the least loaded path via the neighboring switch.

Availability can be learned via time-outs. If a switch did not receive a probe from a ToR over a port for some time, the switch assumes that no path to that ToR exists. This way, a failing link is recognized and traffic will be rerouted at the switch. And information about the now potential higher load on the remaining load will be propagated throughout the network.

Information saved in the probe header is:

- **torID** (32bits): Origin ToR of the probe. Switches match this to their possible destination ToRs, for which this probe is carrying path utilization in the opposite direction.
- **minPathLoad** (32bit): minimum path load of the path the probe took if the packet were to travel in opposite direction. For this thesis, load corresponds to flow size.

The probe replication mechanism is shown in Figure 5.3. Probes are broadcasted to all neighbouring switches, except probes coming from upstream switches only to downstream switches. This avoids loops and therefore unneccessary overhead of circulating probes. To further reduce overhead, there is a timeout preventing replication of probes if there was a probe replicated in a definable threshold.

5.4 Packet scheduling

The scheduling algorithm of IMBAL is similar in structure as HULA [1].

[1]For the HULA pseudocode, see Appendix A.1

To implement the MR algorithm described in 5.1, every switch needs to keep track of several state variables and static Information regarding MR.

Static information needed on every switch:

- `MR constants` $c = 1 + \sqrt{1 + \ln(2)/2}$
  $\alpha = \frac{2c-3}{2(c-1)} \approx 0.46$, as described in Section 5.1.
- `imbalM` corresponds to m in 1: number of machines. In this case: possible output ports for packets. This has the potential to be variable (e.g. failing links), but that possibiliy was not explored in this thesis.
- `imbalK` corresponds to $k = \lfloor \frac{2(c-1)^2-1}{c} m \rfloor + 1 \approx \lfloor 0.36m \rfloor + 1$. Number of ports IMBAL keeps lightly loaded.
- `KEEP_ALIVE_THRESH` Timeout after which an entry in `pathLoad` is obsolete. Could be used to dynamically alter `imbalM` if link failures occur.
- FLOWLET_TIMEOUT time after which a packet with the same flowHash is considered a new flowlet.

Variable state stored on every switch:

- `txLoad[port]` outgoing load per port. Sum of flowsize currently assigned to this port.
- `pathLoad[ToR][port]` 2 dimensional array holding minimum path load for every ToR on every port (this is needed for Imbal calculations). Ports need to be ordered by current load for every port.
- `imbalLambda[ToR]` Corresponds to λ as described in 1. Average load on the imbalK least loaded paths $\lambda = \frac{1}{k} \sum_{i=1}^{k} l_i$
- `updateTime[ToR][port]` array holding last update time for every `pathLoad` entry.
- `flowletTime[flowHash]` array of last time a packet of a specific flow arrived.
- `flowletHop[flowHash]` best hop for flow at the time of flowlet start.
- `flowletSize[flowHash]` tracks how many flowsize is already processed to be able to subtract the rest if needed.

```
1 probe p arrives at switch on port rxPort:
2 if $p.bestPathLoad < txLoad[rxPort]$ then
3     maxLoad = txLoad[rxPort]
4 else
5     maxLoad = p.bestPathLoad
6 end
7 pathLoad[p.torId][rxPort] = maxLoad
8 updateTime[p.torId][rxPort] = currentTime
9 p.bestPathLoad = $min_{m \in \mathcal{M}}(pathLoad[p.torId][m])$
```

Algorithm 2: IMBAL probe processing

If the received packet on a IMBAL switch is a probe, it is handled fundamentally different than other types of traffic such as data packets. Probe processing is described in Algorithm 2.

In Line 3, the switch checks if the load the switch assigned to port `rxPort` is larger than the load that is reported in the probe. If this is the case, then the switch updates the load in the probe with the load on its port (see). If the assigned load is smaller than the load reported in the probe, the switch retrieves the minimum load it has stored for the ToR ID in the probe on that port. All `pathLoads` are saved, so no comparison needs to be made (see line 7), since the calculation of the next best hop needs information regarding all paths anyway. If load reported in the probe is smaller than the minimum path load stored on the switch, then the switch updates its minimum path load for the ToR the probe originated on and the port. If the minimum path load is smaller than the load in the probe, then the switch updates the probe with this value. checks if last hop has higher load than the previous path of probe (similar to HULA with pathUtil of probe and local utilization).

Line 8: last update time is used to detect if a path is no longer available.

Line 9: Update probe path load to best possible load from this switch.

After this, multicast the probe as described in Section 5.3.

Algorithm 3 variable	MR variable
$imbalM$	m
$imbalLambda$	λ
$imbalL$	L
$imbalK$	k
$pathLoadDst[i]$	l_i
$newLoad$	p_t

Table 5.1: Variables used in the Algorithm 3 corresponding to the MR variables in Section 5.1

The IMBAL packet scheduling process is described in Algorithm 3. It aims to combine the flowlet routing of HULA with the scheduling techniques in MR.

```
if $currentTime - flowletTime[flowHash] > FLOWLET\_TOUT$ then
    pathLoadDst = pathLoad[p.dstToR]
    imbalLambda = 0
    for $i = 1; i < k; i++$ do
        imbalLambda += pathLoadDst[i]
    end
    imbalLambda = imbalLambda / imbalK
    imbalL = 0
    for $i = 1; i < m; i++$ do
        imbalL += pathLoadDst[i]
    end
    imbalL = (imbalL + newLoad) / imbalM
    if $imbalLambda > (alpha * pathLoadDst[2imbalK + 1]$ and
     $pathLoadDst[imbalK + 1] + newLoad < c * imbalL)$ then
        bestHop = port of the imbalK+1 smallest entry of pathLoadDst
    else
        bestHop = port where pathLoadDst is minimum
    end
    if $flowletTime[flowHash] = 0$ then
        txLoad[bestHop] += newLoad
    else
        txLoad[flowletHop[flowhash]] -= (newLoad - lastFlowletSize[flowHash])
        txLoad[bestHop] += newLoad - flowletSize[flowHash]
        lastFlowletSize[flowHash] = flowletSize[flowHash]
        flowletSize[flowHash] = 0
    end
    flowletHop[flowHash] = bestHop
end
p.nextHop = flowletHop[flowHash]
flowletTime[flowHash] = currentTime
```

Algorithm 3: IMBAL forwarding

Line 2: calculate current *imbalLambda* as described in 5.4. Needs ordered

$pathLoadDst[i]$ for every destination $dstToRId$. This calculation is dependent on the destination Port.

Line 7: $imbalL$ is average load of the machines. MR simply adds all jobs up until this point. Here, we add all accumulated loads on the paths and add the new job.

Line 11 $imbalL$ denotes the average load on the machines after the current load is assigned therefore add the $newLoad$ here (e.g. $flowSize$ from oracle).

Line 13: Is schedule imbalanced? The calculation assumes $pathLoad$ is ordered.

Line 18: Update local load. If a flow already exists, already processed part has to be subtracted from previously assigned port. For this we need to also keep track of the $flowletSize$ on the switch, if this flow gets reassigned again to keep the values consistent.

Line 27: Set next hop of packet to next hop for its flowlet and start/refresh timer for flowlet.

5.5 Flow-size estimation

In order to keep an imbalanced schedule, IMBAL requires the load, i.e., size of a flowlet as described in Section 5.2. As this information is typically not availiable when a flowlet has to be routed, some sort of prediction is needed. IMBAL estimates the flow-size based on the already send volume, similar to [BCC$^+$14].

Flowlets are assigned to one of b bins. These bins are tags for ranges of already transmitted data for this flow. The predicted flowlet size is the right edge of the current bin. The example in Fig. 5.4 shows a flowlet which has just crossed the b_1 threshold, and is as such assumed to have size b_2. Each ToR switch keeps track of the current bin for every flowlet originating from hosts connected to the ToR. The bin a flowlet is currently assigned (also called the grain size) is added to packets of the flowlet. Intermediate switches can the use this tag to estimate the remaining flowsize as $b_{i+1} - b_i$, which is then used to make forwarding decisions. If a switch registers a bin change, the corrsponding flowlet gets reassigned: the load corresponding to its former bin size gets removed from its assigned port, and

IMBAL is used to newly assign the flowlet with the right edge of the new bin as estimated load.

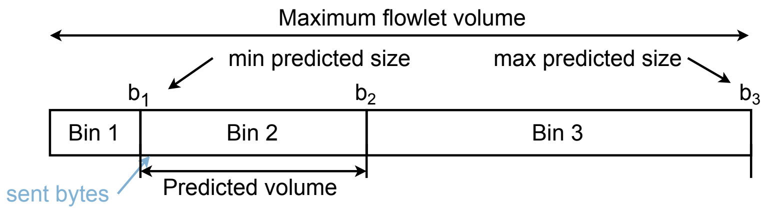


Figure 5.4: Sieving Bins

Sensible bin sizes are subject to configuration, but at least an accurate estimate of the maximum flowlet size is needed. The maximum flowlet size can be used as a maximum estimated flow-size, meaning the right edge of the highest bin. Since flowlets get reassigned on bin change, this could lead to unwanted packet reordering, which was not accounted for in this thesis. In our experiments, a small number of bins with exponentially growing bin sizes are used, see Section 6.4.

5.6 Complexity differences between HULA and IMBAL

Chapter 5.3 in [KHK+16] states the needed operations of HULA in P4. In addition to the staless operations such as probe header adjustments based on link utilization, HULA needs to keep track and modify several stateful lists:

- Minimum path utilizations, $size \propto numberofdestinationToRs$
- The next best hop per destination,
- Last update time per destination
- Next hop per flowlet
- Last time a flowlet was routed

These lists need to be compared to header fields and replaced if the algorithm meets criteria described in section 2.5.

IMBAL handles probes in a similar way, but instead of only saving the minimum path utilization per destination, this algorithm needs to keep track of the load per destination per possible output port, so the size of this matrix is $\propto (\#destination\,ToRs) * (\#output\,ports)$.

The computational cost is also higher: at the moment of assigning a new flowlet, IMBAL needs the list of loads for its particular destination node per output port sorted by value. In the current implementation this sorting is done directly before routing a new flowlet, but on an actual realization on switches, sorting after each relevant probe should be considered. This has the potential to decrease the routing time for data packets.

Additionaly, the routing decision requires several computations and temporary values, as described in 5.1.

Chapter 6

Evaluation

This Chapter evaluates the IMBAL ns-3 implementation in comparison to HULA and ECMP. In Section 6.1, relevant research questions for this thesis are introduced. Section 6.2 describes the chosen simulation scenarios, while Section 6.3 and Section 6.4 describe the results for clairvoyant IMBAL and IMBAL with sieving respectively.

6.1 Research Questions

With the evaluation this thesis wants to investigate the following questions:

1. How does IMBAL, i.e., makespan minimization compare to HULA, i.e., load balancing?
2. How does IMBAL behave in larger network topologies?
3. What is the difference between clairvoyant and non-clairvoyant scheduling, i.e., between sieving and perfect flow-size information?

To evaluate those questions, we use the flow completion time (FCT). This is the time between the first packet of a flow is send, and the last packet of a flow is received, as performance metric. In addition to evaluating the average FCT as is done in previous work [KHK+16], we also investigate the distribution of FCT based on flow-size.

6.2 Simulation Scenario

This Section describes the chosen evaluation scenarios regardin chosen parameters (Section 6.2.1) topology (Section 6.2.2) and traffic (Section 6.2.3).

6.2.1 Routing algorithms and parameters

HULA is a close alternative to IMBAL and targets the minimization of the maximum link utilization as described in Section 2.5. HULA forwards flowlets along the currently least utilized path. As HULA provides the basic mechanisms of traffic routing for IMBAL, all evaluation is done in comparison to a reimplementation of HULA.

Free parameters in our simulation are the flowlet gap, i.e., the time between two packets of the same flow that separates two flowlets, the probe frequency, and the number of bins as well as the distribution of the bins over flow sizes. Following previous work [KHK$^+$16], the order of the flowlet-gap is set to be in the order of the network RTT to minimize packet re-ordering at the receiver. For this thesis, a value of $10\,ms$ is used in our experiments. We set the probe frequency for HULA and IMBAL to $5\,ms$, as to not overwhelm the network with overhead traffic. To keep the evaluation results betwen IMBAL and HULA comparable, probe frequency and flowlet gap were chosen the same for all scenarios.

Where ECMP is used, it is flow-level ECMP, i.e., the next hop of a flow is determined by a hash over the five tuple source IP, destination IP, source port, destination port and protocol.

To showcase the full potential of load imbalancing, IMBAL is evaluated additionally in a clairvoyant setting. In the clairvoyant setting, an oracle provides the size of each flow during arrival. Thus, sieving for the flow-size is not necessary. Aside from the flow-size no additional information is provided out-of band.

6.2.2 Topology

Since load imbalancing only works for more than three machines, the smallest topology used in this analysis is the $k = 8$ FatTree. Figure 6.1 shows one pod of a k=8 FatTree. Traffic coming from a host (red) arriving on its ToR switch (green) can be routed to one of 4 output upstream output ports if the destination is not connected to, which all have a valid route to all other hosts in this topology. FatTree sizes in this evaluation go up to $k \in \{8, 10, 12, 14, 16\}$, meant to show scalability of IMBAL. For this thesis, all the scenarios have a link capacity of 10 Mbps. This is meant to keep simulation times to a manageable level.

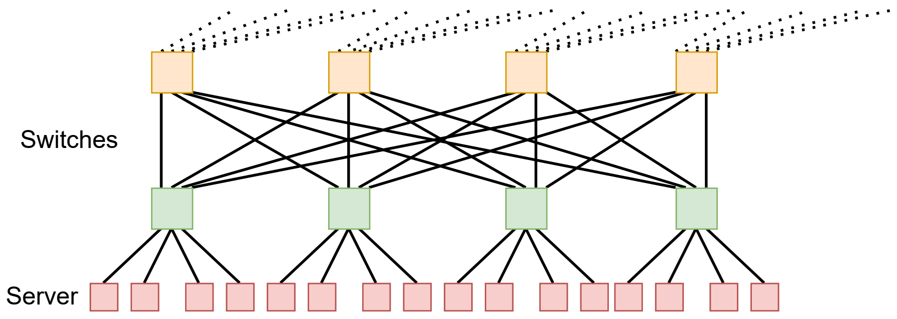


Figure 6.1: One Pod of a k=8 FatTree

6.2.3 Traffic

This thesis uses an incast scenario, where one pod receives traffic from other pods. Flows arrive based on a poisson arrival process. Communication pairs are sampled uniformly at random. The arrival process is tuned to achieve a specific expected load in the receiving pod. As traffic distribution, flowsizes were sampled from production traces. Figure 6.2 shows the CDF of the used distribution [AED+14], denoted by the blue line. Of interest is the heavy tailed nature of the workload as described in 2.1.

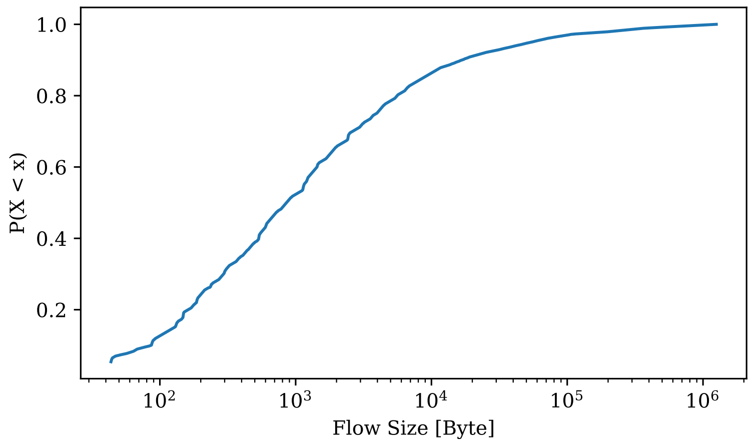


Figure 6.2: CONGA enterprise workload

Initial simulations were conducted with UDP as transport protocol. As this means no retransmission of lost packets or other traffic control present in TCP, this scenario provides a basic overview over the potential capabilities of IMBAL.

For TCP, every flow establishes a seperate TCP connection between one of the hosts in the destination pod and a random source host in another pod.

6.3 A priori Flow Knowledge

As described in Section 5.2, initial simulations were conducted with clairvoyance over flow sizes for IMBAL. In practice, the availability of this information is problematic [DJK+19]. In general, advance knowledge of flow sizes is not a plausible assumption. Nevertheless, assuming a priori knowledge over flow sizes provides an overview over potential performance improvements.

6.3.1 UDP traffic

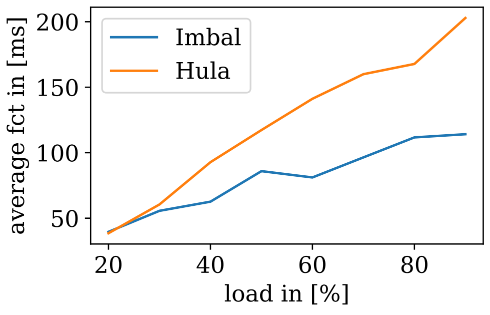
Figure 6.3 shows the performance of IMBAL compared to HULA for a k=8 FatTree. Subfigure 6.3a shows the average flow completion time for loads between 20% and 90%. While the average flow completion time is about the same for IMBAL and HULA for 20% network load, if the network is loaded more than 30%, differences start to show: on average, for this scenario, IMBAL has roughly two times faster average flow completion time than HULA.

Subfigure 6.3b shows the corresponding error rate as a percentage of packages lost. While at low loads, IMBAL and HULA are about equal. But at loads higher than 70%, HULA lost more than twice as much packets.

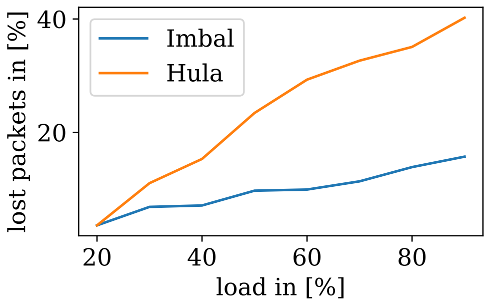
Both of the above results point to a better congestion avoidance of IMBAL in this scenario.

Figure 6.4 shows violin plots over all flow completion times for a specific load level. The average flow completion time is marked with a star. The median flow completion time with a bar. Of note is the roughly equal median flow completion time except in the highest loaded scenario. Especially for loads higher than 50% the number of flows with a higher flow completion time than 200ms surpasses the number of flows with IMBAL scheduling.

To show the scalablity of IMBAL, Figure 6.5 shows the same metrics as Figure 6.3 but for a larger network size of k=12. Subfigure 6.5a shows the average flow completion time for loads between 20% and 90%. While the average flow completion time is about the same for IMBAL and HULA for up to 40% network load, if the



(a) 1a



(b) 1b

Figure 6.3: Average flow completion times (a) and error rates (b) at different loads for UDP and a k=8 FatTree with 30 seconds Simulation time

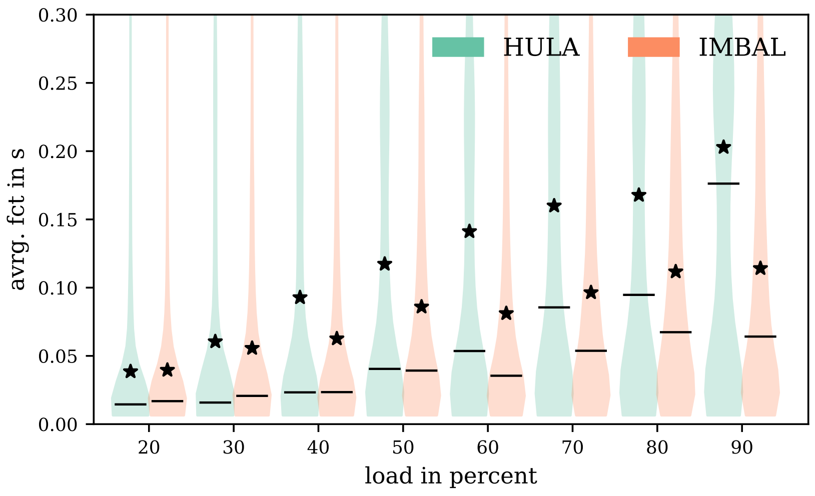
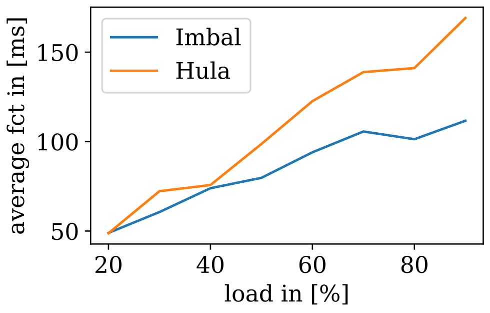


Figure 6.4: Flow completion at different loads for a k=8 FatTree with 30s simulation time

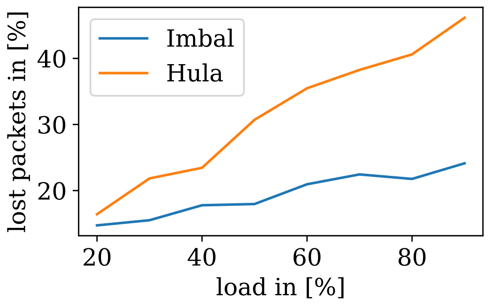
network is loaded more than 50%, IMBAL flow completion times are 50% smaller on average compared to HULA.

Subfigure 6.5b shows the corresponding error rate as a percentage of lost packages. While at low loads, IMBAL and HULA are about equal. But at loads higher than 70%, HULA lost more than twice as much packets.

Another metric to evaluate, especially for UDP, is packet loss distribution per flow depending on flow size. Figure 6.7 shows a heatmap for this metric. The x-axis shows the number of sent packets per flow, while the y-axis shows the amount of packets received for each of those flows. A lighter color indicates that a larger



(a) Average flow completion time



(b) Percentage of lost Packets

Figure 6.5: Average flow completion times (a) and error rates (b) at different loads for a k=12 FatTree with 30 seconds Simulation time

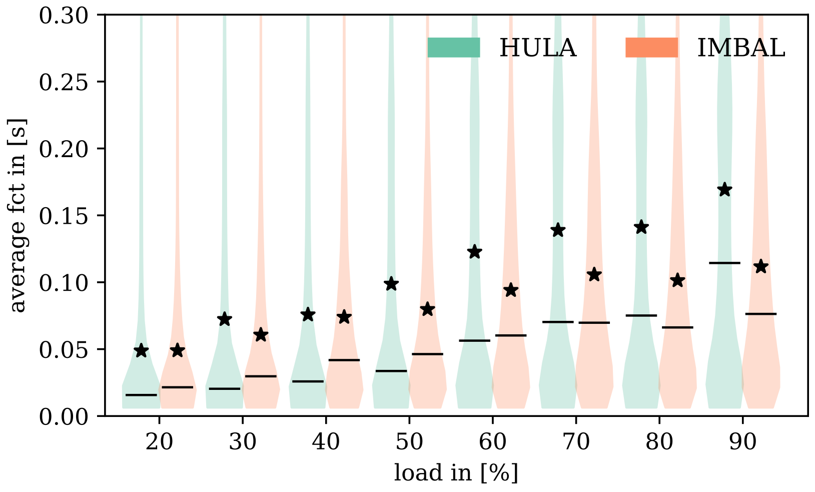


Figure 6.6: Flow completion at different loads for a k=12 FatTree with 30s simulation time

number of flows fall that belong to the corresponding bin. If no packets would be dropped, the plot would show a diagonal line. In case of packet loss, the corresponding entry is translated downwards. Figure 6.7 reflects the heavy tailed flow size distribution. The bright color at the lower left indicate a large number of small flows. The bins corresponding to larger flows are darker, indicating fewer flows. Figure shows specifically lost packets for higher flows, as the resolution of the graph is not high enough to show meaningful data in the bottom left corner. For HULA, quite a few packets in the flow size range of 50 to 250 packets get lost, while IMBAL distribution is closer to the case of no losses. Even more apparent becomes the discrepancy for the larger flows. While IMBAL is able to route more than 75% percent of packets per flow in most cases, HULA has packet drops and very few completely transmitted flows for this heavily loaded scenario, specifically regarding large flows.

The data in Figure 6.7 therefore shows the potential advantages of keeping an imbalanced schedule to accomodate for these large flows.

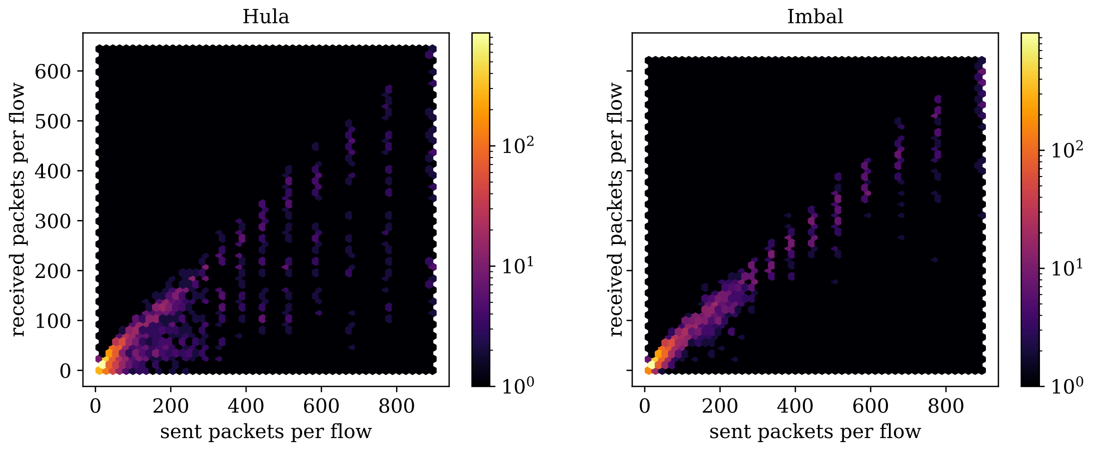


Figure 6.7: Heat map showing relationship between sent packets to received packets, for a k=16 FatTree with UDP web traffic at network load 90%

6.3.2 TCP traffic

The dominant transport protocol in data center networks is TCP [AGM+10]. Thus, the behavior of IMBAL with TCP is important.

Figure 6.8 shows FCTs in the form of a violin plot. The stars denote the average flow completin time, while the bars denote the median flow completion time. The results are similar to the results obtained with UDP: While the FCTs are about the same in scenarios up to a load of 30% with even a slightly higher median, IMBAL achieves lower FCTs than HULA by approximately 50% with a load of 50% or higher.

Figure 6.9 shows average FCTs (6.9a) for a k=12 FatTree Simulation as well as the its errorate 6.9b. The average FCT of IMBAL and HULA are close to each other up to a load of 40%. For higher loads, IMBAL has a up to 60% smaller FCT compared to HULA. This confirms the results of the UDP simulations. The behavior of lost packets is similar as well. Figure 6.9b shows the amount of lost packets. IMBAL has a 30% smaller error rate compared to HULA for load levels

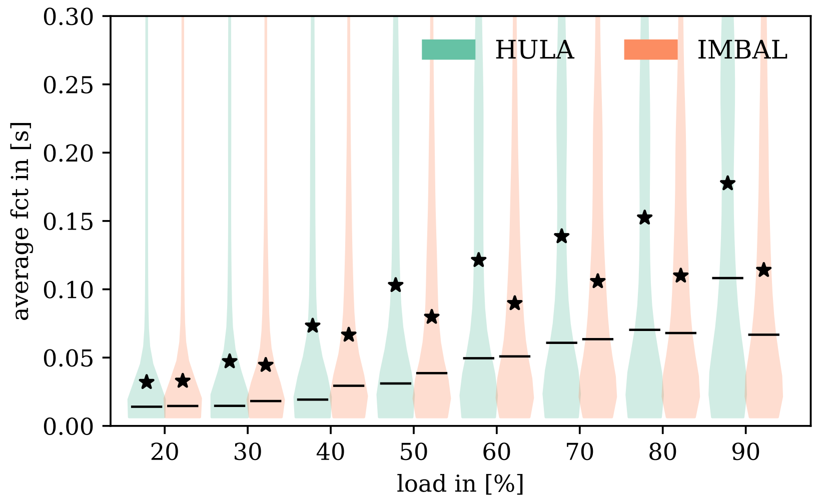
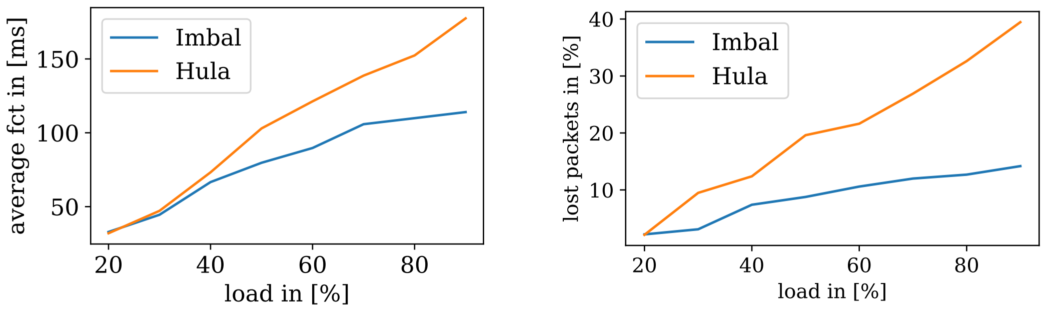


Figure 6.8: TCP Flow completion at different loads for a k=8 FatTree with 30s simulation time

up to 40%. The gap keeps increasing with load, reaching almost 400% difference between IMBAL and HULA. Since TCP re-transmit packets, significant packet loss indicates increased flow completion times.



(a) FCT

(b) Errorate

Figure 6.9: Average flow completion times (a) and errorrates (b) at different loads for a k=12 FatTree with 30 seconds Simulation time

Figure 6.10 shows a violinplot of the individual FCTs for the k=12 Fat-Tree. In this scenario, HULA has lower average completion times, denoted by the stars, up to a load of 40%. Furthermore, the median flow completion time remains faster or equal compared to IMBAL. This behavior can be explained with the different scheduling strategies. In the lowly loaded scenarios, HULA profits from utilizing each port equally. IMBAL keeps some ports more loaded than others. Due to the low utilization, IMBAL cannot benefit, though. That is the utilization is small enough that collisions are infrequent for HULA. This changes starting at higher loads. Collisions become more frequent and IMBAL can benefit from its strategy.

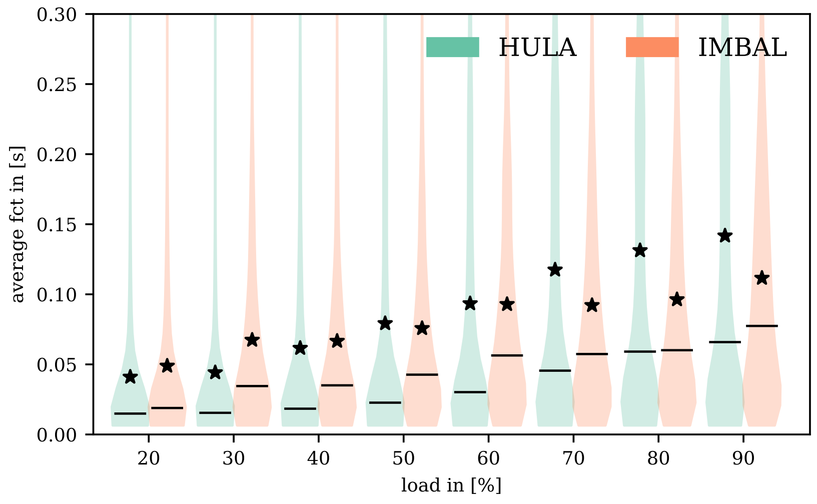


Figure 6.10: TCP Flow completion at different loads for a k=12 FatTree with 30s simulation time

Figure 6.11 shows the relationship between sent and received packets per flow for a very high load scenario of 90%. Ideally, all sent packets would be received. This means, a diagonal from bottom left to top right would be ideal. IMBAL exhibits a much tighter spread towards this ideal. HULA drops far more packets across flowsizes observable at this hexbin grainsize.

Fig. 6.12 shows the send vs. received number of packets for a load of 70 %. The y-Axis of Fig. 6.12 holds the number of received packets. The x-Axis shows the sent number of packets. Fig. 6.12a shows that in case of HULA the sender sends more packets than received by the receiver. In contrast, IMBAL results in an almost perfect diagonal. IMBAL has fewer packet re-transmissions because of packet loss compared to HULA. This is expected. Since HULA keeps paths approximately equally balanced, a larger flow can easily collide with multiple small flows, resulting in congestion and packet loss. In contrast, IMBAL keeps some ports lowly utilized on purpose. Once a large flow arrives, IMBAL can then place this flow on the lowly utilized port, reducing the potential of collisions and congestion in the process.

Figure 6.14 shows the average queue size distribution for destination pod downstream queues, for a k=8 FatTree with TCP web traffic at network load 50% for clairvoyant IMBAL and HULA. IMBAL is less likely to have queuesizes larger than 10 over time. At the same time, queusizes smaller than 10 are slightly more

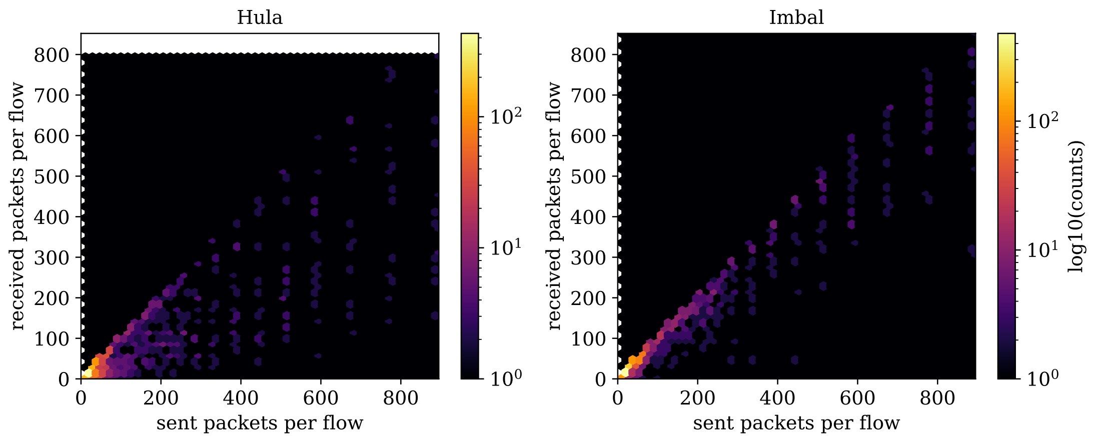


Figure 6.11: Heat map showing relationship between sent packets to received packets, for a k=12 FatTree with TCP web traffic at network load 90%

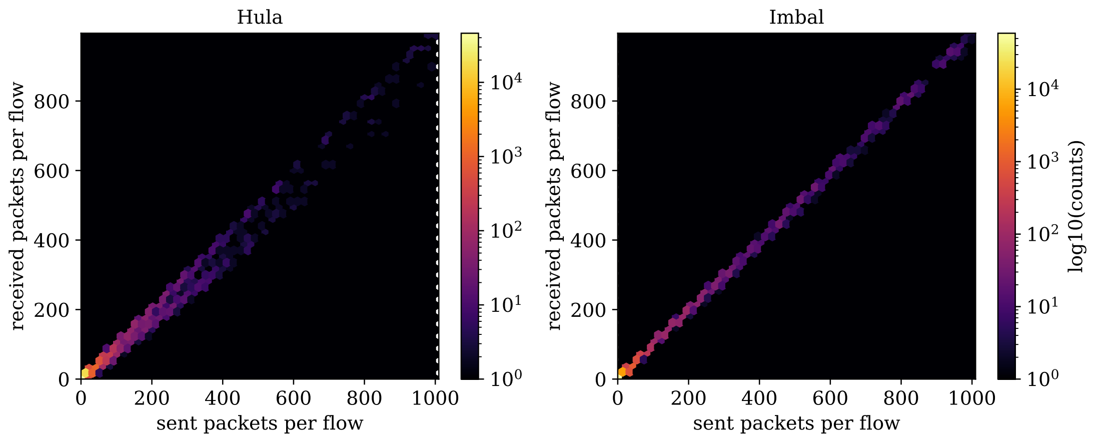


Figure 6.12: Heat map showing relationship between sent packets to received packets, for a k=10 FatTree with TCP web traffic at network load 70%

likely. Since for our scenario, maximum queue size of the queues on the switch was 100, additional arriving packets need to be dropped. This leads to the need for retransmission in the case of TCP, and increases FCT as well as load on the network. Since IMBAL is less likely to assign flows in a way which causes high queuesizes, it is more likely to avoid theses problems, which is in line with the other evaluation results.

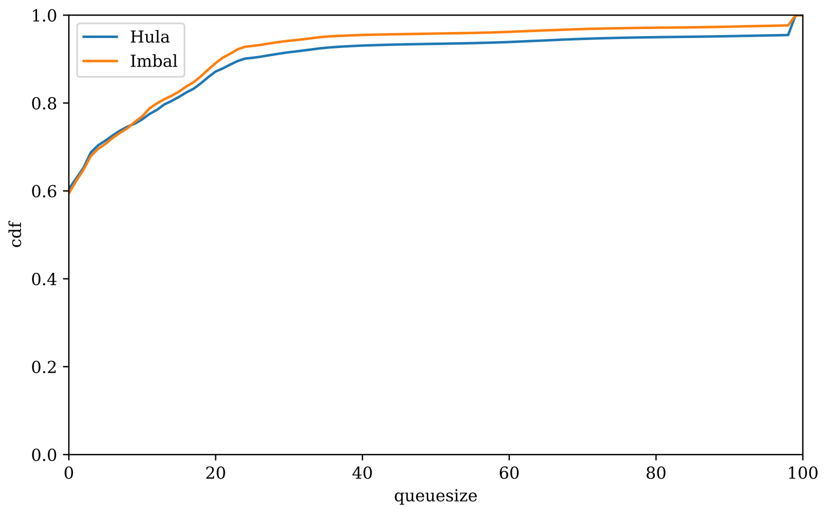


Figure 6.13: CDF showing the queue size distribution over time for destination pod downstream queues, for a k=8 FatTree with TCP web traffic at network load 50%

6.4 Sieving

As stated in Section 5.2 For binning in sieving, we use two different types of bins: and equal-width binning on the distribution of flow-sizes (in plots designated as Siev1) and exponential bin sizes (in plots designated as Siev2). For exponential only two numbers are required, a minimum and maximum value. For equal-width binning the empirical flow-size distribution has to be known, which in practice requires more data gathering effort. With equla-width binning, a histogram is constructed where each bin contains the same number of samples, i.e., has the same height, but different widths. We use 5 bins if not stated otherwise.

Figure 6.14 shows the amount of unfinished flows for TCP traffic. For this scenario, network traffic has been generated for 60 seconds, and flows which have not finished for 3 seconds after that are treated as unfinished. Clairvoyant IMBAL first starts to not finish flows at 30% load, and manages to keep unfinished flows an order of magnitude below HULA. Both sieving bin approaches have less unfinished flows on average than HULA.

Figure 6.15 shows the results of varying the bin count for exponential bin sizes to HULA and ECMP normalized to clairvoyant IMBAL performance regarding average flow completion time. Of note is the strongly smaller flow completion time of HULA and IMBAL in comparison to ECMP. These results highlight the importance of correct binning. Fig. 6.15 indicates a soft spot for 6-7 bins. In

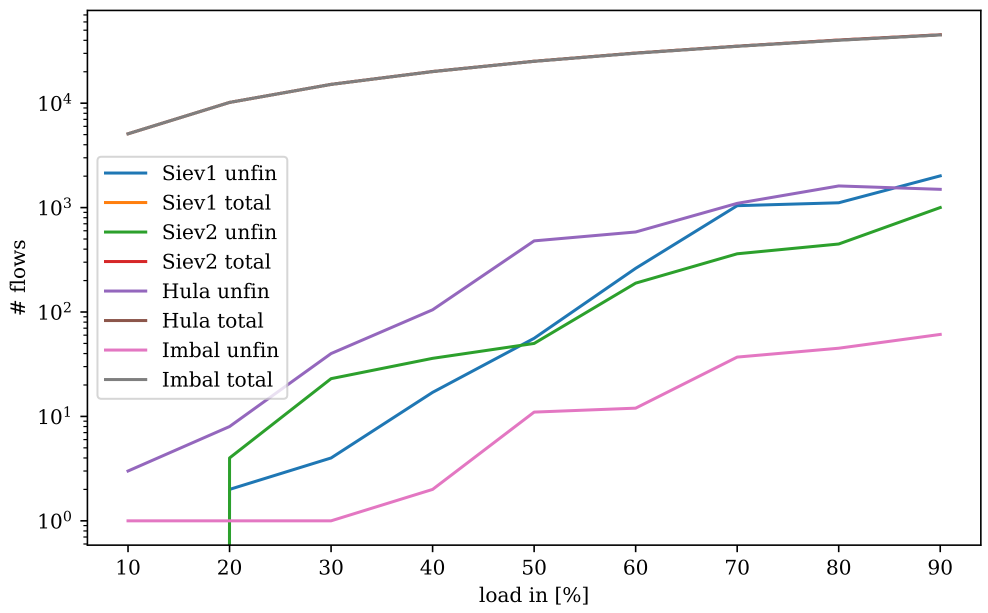


Figure 6.14: Unfinished flows depending on network load 70%

those scenarios, IMBAL with sieving performs almost as well as IMBAL with flow-size information. For more or fewer bins, the performance deteriorates. If very few bins are used, the average flow completion time surpasses HULA, since the estimation of the flow size gets more inaccurate. On very high bin counts, the current bin for each flow changes more often. Since this triggers a reassignment of the flow in IMBAL, the advantages of flowlet routing could be undermined. 5 bins lead to the smallest flow completion time, and are used for the following simulation.

Figure 6.16 shows a similar pattern for increasing topology size, while the relative performances differences increase.

Figure 6.17 introduces a new performance metric: job slowdown. The job slowdown here is defined as $FCT/flowsize$ in seconds per byte, which is shown on the x axis. The plot shows the cumulative distribution of this slowdown. Of note is the lower cap of slowdown at 0.12 for both clairvoyant as well as IMBAL with sieving compared to HULA. Clairvoyant IMBAL has a the lowest slowdown across the board, while the slowdowns of IMBAL with sieving are between HULA and clairvoyant IMBAL. ECMP is included to show the high slowdown discrepancy in the scenario compared to the aforementions algorithms.

Fig. 6.18 shows the FCT of flows for a load of 60 % on the x-Axis and the flow-size on the y-axis. Fig. 6.18c shows that HULA has a diffuse pattern. HULA has

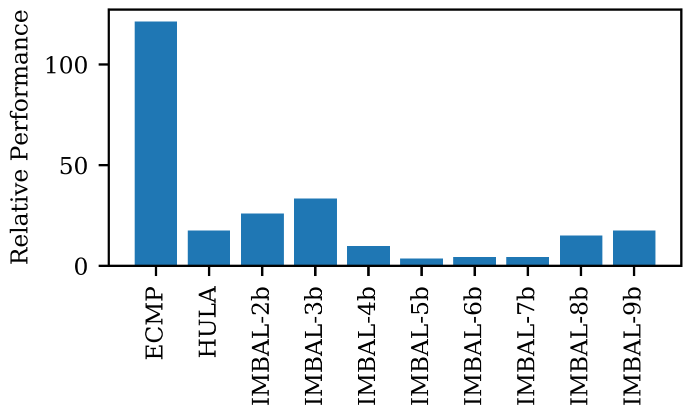


Figure 6.15: Relative performance of different sieving bin counts for k=12 FatTree

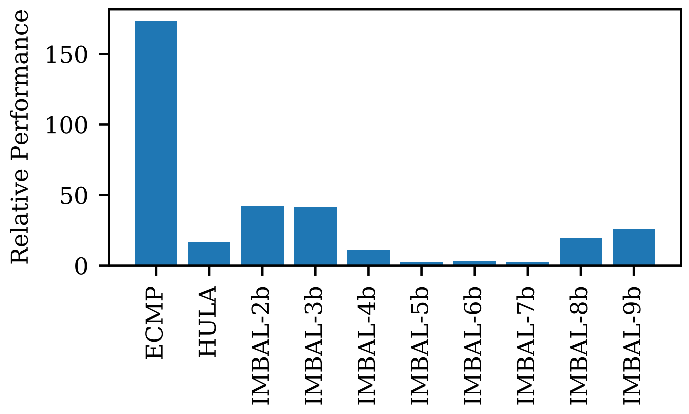


Figure 6.16: Relative performance of different sieving bin counts for k=16 FatTree

many small flows that have a small FCT, indicated by the bright color in the lower left of Fig. 6.18c. For larger flows, the FCT of HULA in Fig. 6.18c is hard to predict. For flows up to 0.5 MBytes follow a roughly linear trend. The FCTs for a specific flow size vary.

In contrast, the FCT for clairvoyant IMBAL behaves more stable. For IMBAL, the FCT also increases with volume. In contrast to HULA, clairvoyant IMBAL results in a smoother behavior. In contrast to HULA, clairvoyant IMBAL has higher FCT for smaller flows compared to HULA.

The two sieving methods mix the behavior from HULA and clarivoyant IMBAL. The sieving methods also have more smaller flows with small FCT, indicated by the bright color in the bottom left corner of Figs.6.18a,b. The sieving methods do

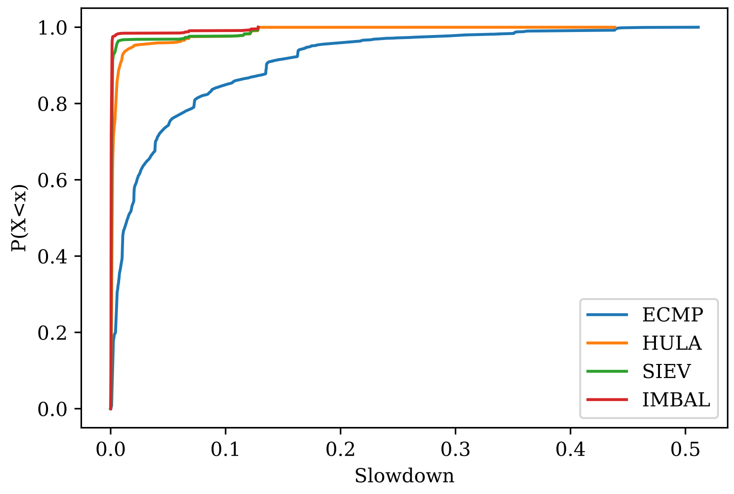


Figure 6.17: Slowdown for k=8 FatTree

show the diffuse behavior of HULA for larger flows, but also the more predictable behavior of clairvoyant IMBAL.

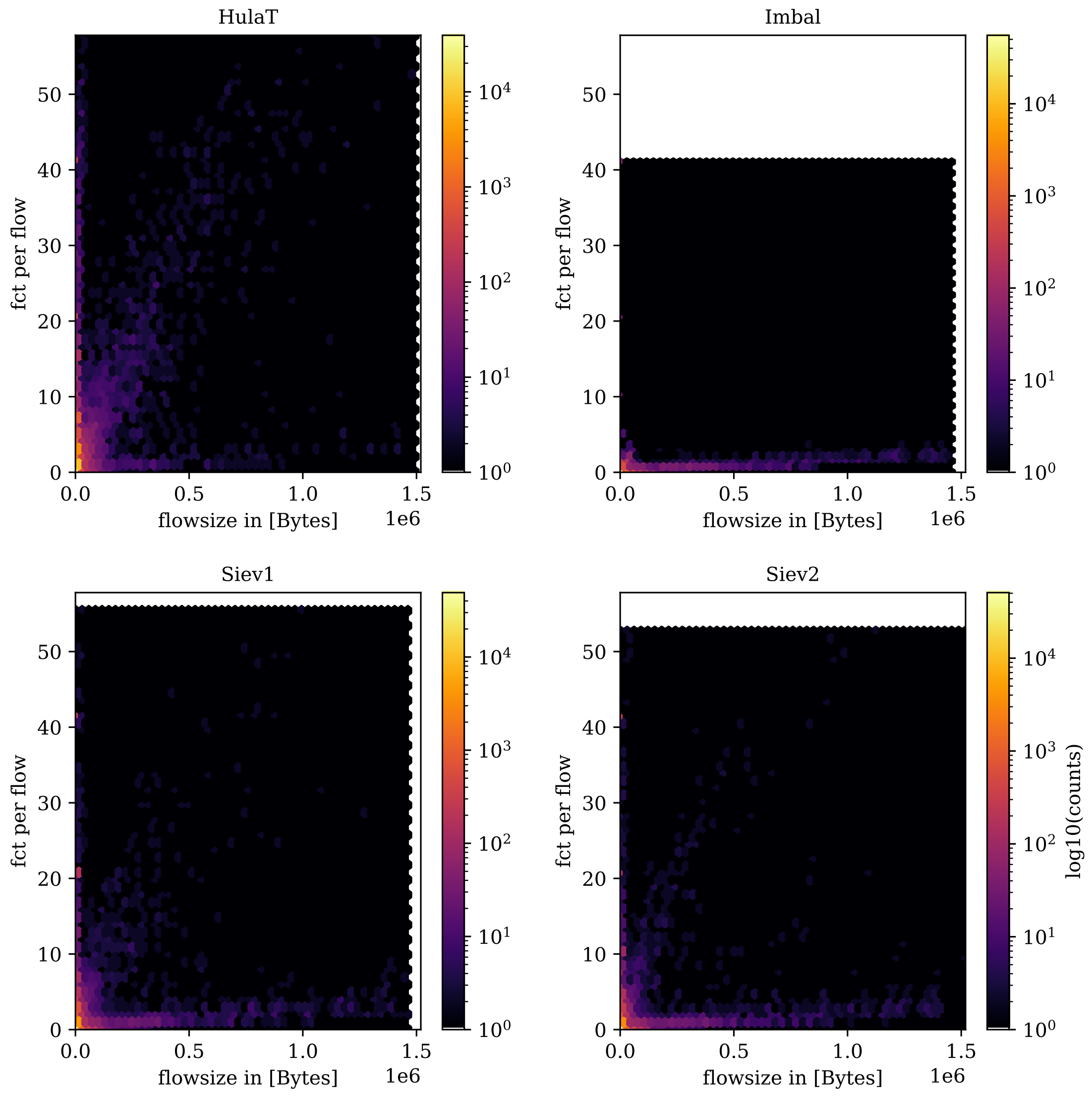


Figure 6.18: FCT depending on flow size for a load of 60 %.

Chapter 7

Conclusions and Outlook

7.1 Conclusions

This thesis proposes IMBAL, an in-dataplane load balancing technique that keeps links purposefully imbalanced and does not require a-prior knowledge of flow sizes. In contrast to existing work, IMBAL optimizes for makespan, i.e., IMBAL treats all flows in the network as belonging to one large coflow and minimizies. IMBAL operates on the granularity of flowlets and uses a distance-vector like protocol to distribute global network state through the network. IMBAL is implemented and evaluated in the network simulator ns-3 and compared against HULA [KHK^{+}16] and ECMP. The evaluation in Section 6 shows that IMBAL is a viable alternative to HULA, outperforming HULA on the evaluated scenarios. IMBAL improves FCT up to 50% on fat-tree topologies ranging in size from k=8 to k=16. Further, IMBAL results in fewer packet losses, and a smaller job slowdown. The evaluation shows that IMBAL with sieving also has the potential to outperform HULA. The evaluation further shows that detailed knowledge of flow size information is helpful to improve the performance of IMBAL.

Contributions of this thesis are:

1. This thesis proposes the imbalancing of traffic as an alternative objective to existing load balancing techniques that optimize for maximum link utilization.
2. For this thesis, IMBAL has been designed, a load imbalancer that does not require advanced knowledge of flow-sizes and can be implemented with recent programmable network devices.

3. For this thesis, IMBAL has been implemented in the network simulator ns-3.
4. The effectiveness of IMBAL is shown in large scale simulations and compare IMBAL against state-of-the art in-network load balancing algorithms of similar complexity.

7.2 Future Work

This thesis opens up interesting directions for future work.

In terms of evaluation, a look into higher datarate simulations is necessary, since current linkrates far surpass the 10Mbps used in the evaluation. Other interesting evaluations could include different traffic characteristics, as the value of imbalancing in scheduling depends on the shape of flow sizes and their arrival pattern.. An example where imbalancing would be suboptimal is uniform traffic, since no large future jobs need to be accounted for. Additionally, persistent TCP connections are typical for data center networks [KHK$^+$16], and would be an interesting test scenario.

As for flow size estimation with sieving, two main points not explored in this thesis seem worth investigating: 1) how to optimally choose the bin size and 2) investigation of the potential detrimental impact of increased packet reordering due to reassignment of flowlets to different paths.

Another aspect to look into is the feasibility of IMBAL in hardware. HULA has been implemented in P4 [B$^+$14], a programming language targeting programmable data planes. Because of the increased computational needs discussed in Section 5.6, feasibility of IMBAL on commodity hardware needs to be explored if it is to be implemented.

List of Figures

List of Tables

Appendix A

A.1 Hula Pseudocode

- probe header fields:
  - **pathUtil** util of best path for packet traveling in opposite direction of probe
  - **torId** origin id of probe, probe is carrying information for this ToR as destination
- variables stored on switch:
  - **txUtil**[**port**] outgoing utilization of port (estimator is used)
  - **minPathUtil**[**ToR**] array holding best path util for every ToR updateTime[ToR] array holding last arrival time of a probe from every ToR
  - **flowletTime**[**flowHash**] array of last time a packet of flow arrived
  - **flowletHop**[**flowHash**] best hop for flow at th time of flowlet start
- static variables on switch:

  KEEP_ALIVE_THRESH time after which information of ToR is considered obsolete and the first new arriving probe will be registered as best **FLOWLET_TIMEOUT** time after which a packet with the same flowHash is considered a new flowlet

```
packet p arrives at switch on port rxPort:
/* is packet a probe (packet has hulaprobe header)?          */
if $p = HulaProbe$ then
    /* check if last hop has higher util than tho previous path
       of probe                                               */
    if $p.pathUtil < txUtil[rxPort]$ then
        p.pathUtil = txUtil[rxPort]
    end
    /* if probe path is better or last update for the probe
       origin is obsolete                                     */
    if $p.pathUtil < minPathUtil[p.torId]$ or
     $currentTime - updateTime[p.torId] >$
     $KEEP\_ALIVE\_THRESHOLD$ then
        /* update values for ToR to the values of the probe    */
        minPathUtil[p.torId] = p.pathUtil
        bestHop[p.torId] = rxPort of probe
        updateTime[p.torId] = currentTime
    end
    /* update probe path util to best possible path from this
       switch to its ToR                                      */
    p.pathUtil = minPathUtil[p.torId]
else
    /* DataPackets or packets of other protocols              */
    /* if packet does not belong to an active flowlet (flow of
       this packet unknown or timed out)                      */
    if $currentTime–flowletTime[flowHash] > FLOWLET\_TOUT$ then
        /* assign flowlet to current best next hop             */
        flowletHop[flowHash] = bestHop[p.dstTor]
    end
    /* set next hop of packet to next hop for its flowlet     */
    p.nextHop = flowletHop[flowHash]
    /* start/refresh timer for flowlet                        */
    flowletTime[flowHash] = currentTime
end
update txTtil after packets have been sent
```

Algorithm 4: HULA

Appendix B

Notation und Abkürzungen

HULA	Hop-by-hop Utilization-aware Load balancing Architecture
SJF	Shortest Job First
CONGA	Congestion Aware Balancing
ECMP	Equal-Cost Multi-Path
FCT	Flow Completion Time
PCT	Packet Completion Time
ToR	Top of the Rack
TTL	Time to Live
ns-3	network simulator version 3

Bibliography

[ACX$^+$20] Ashkan Aghdai, Cing-Yu Chu, Yang Xu, David Dai, Jun Xu, and Jonathan Chao. Spotlight: Scalable Transport Layer Load Balancing for Data Center Networks. *IEEE TCC*, pages 1–1, 2020.

[AED$^+$14] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. *SIGCOMM Comput. Commun. Rev.*, 44(4):503–514, August 2014.

[AFLV08] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, August 2008.

[AFRR$^+$10] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI 10*, pages 281–296, San Jose, CA, USA, 2010. ACM.

[AGM$^+$10] Mohammad Alizadeh, Albert Greenberg, Dave Maltz, Jitu Padhye, Parveen Pate, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *SIGCOMM '10*. ACM, September 2010. Citation Kez: alizadeh2010.

[AHLPMY$^+$19] J. Alvarez-Horcajo, D. Lopez-Pajares, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco. Improving Multipath Routing of TCP Flows by Network Exploration. *IEEE Access*, 7:13608–13621, 2019.

[AKE$^+$12] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center. In *NSDI 12*, pages 253–266, San Jose, CA, 2012. USENIX.

[Alb09] Susanne Albers. Online Scheduling. In *Introduction to Scheduling*, pages 57–84. Chapman and Hall/CRC Press, 2009.

[ARN$^+$18] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. Sincronia: Near-optimal Network Design for Coflows. In *SIGCOMM '18*, pages 16–29, Budapest, Hungary, 2018. ACM.

[AYS$^+$13] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pFabric: Minimal Near-optimal Datacenter Transport. *SIGCOMM Comput. Commun. Rev.*, 43(4):435–446, August 2013.

[B$^+$14] Pat Bosshart et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Comput. Commun. Rev.*, 2014.

[BAAZ11] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *CoNEXT '11*, pages 8:1–8:12, Tokyo, Japan, 2011. ACM.

[BCC$^+$14] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Weicheng Sun. PIAS: Practical Information-Agnostic Flow Scheduling for Data Center Networks. In *HotNets '13*, pages 25:1–25:7, New York, NY, USA, 2014. ACM.

[BDS$^+$21] Maciej Besta, Jens Domke, Marcel Schneider, Marek Konieczny, Salvatore Di Girolamo, Timo Schneider, Ankit Singla, and Torsten Hoefler. High-Performance Routing With Multipathing and Path Diversity in Ethernet and HPC Networks. *IEEE TPDS*, 32(4):943–959, 2021.

[BK19] C. H. Benet and A. J. Kassler. FlowDyn: Towards a Dynamic Flowlet Gap Detection using Programmable Data Planes. In *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, pages 1–7, November 2019.

[BKA$^+$20] Cristian Hernandez Benet, Andreas J. Kassler, Gianni Antichi, Theophilus A. Benson, and Gergely Pongrácz. Providing In-network Support to Coflow Scheduling. *CoRR*, abs/2007.02624, 2020.

[CCBA16] Li Chen, Kai Chen, Wei Bai, and Mohammad Alizadeh. Scheduling Mix-flows in Commodity Datacenters with Karuna. In *SIGCOMM '16*, pages 174–187, Florianopolis, Brazil, 2016. ACM.

[CKS14] M. Chiesa, G. Kindler, and M. Schapira. Traffic engineering with Equal-Cost-Multipath: An algorithmic perspective. In *INFOCOM 2014*, pages 1590–1598, Toronto, Canada, April 2014. IEEE.

[CLCL18] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. AuTO: Scaling Deep Reinforcement Learning for Datacenter-scale Automatic Traffic Optimization. In *SIGCOMM '18*, pages 191–205, Budapest, Hungary, 2018. ACM.

[CS15] Mosharaf Chowdhury and Ion Stoica. Efficient Coflow Scheduling Without Prior Knowledge. In *SIGCOMM '15*, pages 393–406, London, United Kingdom, 2015. ACM.

[CZS14] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient Coflow Scheduling with Varys. In *SIGCOMM '14*, pages 443–454, Chicago, IL, USA, August 2014. ACM.

[DJK$^+$19] Vojislav Dukić, Sangeetha Abdu Jyothi, Bojan Karlas, Muhsen Owaida, Ce Zhang, and Ankit Singla. Is advance knowledge of flow sizes a plausible assumption? In *NSDI 19*, pages 565–580, Boston, MA, February 2019. USENIX Association.

[DM06] Nandita Dukkipati and Nick McKeown. Why Flow-completion Time is the Right Metric for Congestion Control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, January 2006.

[FW00] Rudolf Fleischer and Michaela Wahl. Online Scheduling Revisited. In Mike S. Paterson, editor, *Algorithms - ESA 2000*, pages 202–210, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[GNK$^+$15] Peter X. Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. pHost: Distributed Near-optimal Datacenter Transport over Commodity Network Fabric. In *CoNEXT '15*, pages 1:1–1:12, Heidelberg, Germany, 2015. ACM.

[Gra66] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[GSG$^+$15] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon

Crowcroft. Queues Don't Matter When You Can JUMP Them! In *NSDI 15*, pages 1–14, Oakland, CA, 2015. USENIX Association.

[HBC$^+$20] Kuo-Feng Hsu, Ryan Beckett, Ang Chen, Jennifer Rexford, and David Walker. Contra: A Programmable System for Performance-aware Routing. In *NSDI 20*, pages 701–721, Santa Clara, CA, February 2020. USENIX Association.

[HCG12] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. Finishing Flows Quickly with Preemptive Scheduling. In *SIGCOMM '12*, pages 127–138, Helsinki, Finland, 2012. ACM.

[HRA$^+$15] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based Load Balancing for Fast Datacenter Networks. In *SIGCOMM '15*, pages 465–478, London, United Kingdom, 2015. ACM.

[HTB$^+$20] Kuo-Feng Hsu, Praveen Tammana, Ryan Beckett, Ang Chen, Jennifer Rexford, and David Walker. Adaptive Weighted Traffic Splitting in Programmable Data Planes. In *SOSR '20*, SOSR '20, pages 103–109, San Jose, CA, USA, 2020. Association for Computing Machinery.

[KHG$^+$16] Naga Katta, Mukesh Hira, Aditi Ghag, Changhoon Kim, Isaac Keslassy, and Jennifer Rexford. CLOVE: How I Learned to Stop Worrying About the Core and Love the Edge. In *HotNets '16*, pages 155–161, Atlanta, GA, USA, 2016. ACM.

[KHK$^+$16] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. HULA: Scalable Load Balancing Using Programmable Data Planes. In *SOSR '16*, pages 10:1–10:12, Santa Clara, CA, USA, 2016. ACM.

[KKSB07] Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. Dynamic Load Balancing Without Packet Reordering. *SIGCOMM Comput. Commun. Rev.*, 37(2):51–62, March 2007.

[KS17] Abdul Kabbani and Milad Sharif. Flier: Flow-level congestion-aware routing for direct-connect data centers. In *INFOCOM 2017*, pages 1–9, Atlanta, GA, USA, May 2017. IEEE.

[LZW$^+$20] Kefei Liu, Jiao Zhang, Dehui Wei, Kai Zhang, and Tao Huang. PLB: Adaptive Partial Congestion-aware Load Balancing for Datacenter Networks. In *GLOBECOM 2020*, pages 1–6, Virtual Event, Taiwan, 2020. IEEE.

[MLAO18] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John K. Ousterhout. Homa: a receiver-driven low-latency transport protocol using network priorities. In Sergey Gorinsky and János Tapolcai, editors, *SIGCOMM '18*, pages 221–235, Budapest, Hungary, 2018. ACM.

[POB$^+$14] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A Centralized "Zero-queue" Datacenter Network. *SIGCOMM Comput. Commun. Rev.*, 44(4):307–318, August 2014.

[RBP$^+$11] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *SIGCOMM '11*, pages 266–277, Toronto, ON, Canada, 2011. ACM.

[RCAV19] Hamed Rezaei, Muhammad Usama Chaudhry, Hamidreza Almasi, and Balajee Vamanan. ICON: Incast Congestion Control using Packet Pacing in Datacenter Networks. In *COMSNETS*, pages 125–132, Bengaluru, India, January 2019. IEEE.

[RSD$^+$14] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. Planck: Millisecond-scale Monitoring and Control for Commodity Networks. In *SIGCOMM '14*, pages 407–418, Chicago, IL, USA, 2014. ACM.

[RV20] Hamed Rezaei and Balajee Vamanan. ResQueue: A Smarter Datacenter Flow Scheduler. In *WWW '20*, pages 2599–2605, Taipei, Taiwan, 2020. Association for Computing Machinery.

[SSIF13] Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael Freedman. Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing. In *CoNEXT '13*, pages 151–162, Santa Barbara, CA, USA, December 2013. ACM.

[SWFX19] Qingyu Shi, Fang Wang, Dan Feng, and Weibin Xie. Adaptive load balancing based on accurate congestion feedback for asymmetric topologies. *Comput. Netw.*, 157:133–145, 2019.

[VPA$^+$17] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *NSDI 17*, pages 407–420, Boston, MA, USA, March 2017. USENIX Association.

[XYZ$^+$20] Cong Xu, Tingqiu Yuan, Haibo Zhang, Tao Huang, and Feilong Tang. Dual Channel Per-packet Load Balancing for Datacenters. In *INFOCOM WKSHPS*, pages 157–164, Toronto, ON, Canada, 2020.

[ZCY$^+$16] Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, and Yanhui Geng. CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark. In *SIGCOMM '16*, pages 160–173, Florianopolis, Brazil, 2016. ACM.

[ZDM$^+$12] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *SIGCOMM '12*, pages 139–150, Helsinki, Finland, 2012. ACM. Citaiton Key: zats2012.

[ZMSG19] Mingyang Zhang, Radhika Niranjan Mysore, Sucha Supittayapornpong, and Ramesh Govindan. Understanding Lifecycle Management Complexity of Datacenter Topologies. In *NSDI 19*, pages 235–254, Boston, MA, USA, February 2019. USENIX Association.

[ZTZ$^+$14] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. In *EuroSys '14*, EuroSys '14, pages 5:1–5:14, New York, NY, USA, 2014. ACM.