Scaling beyond packet switch limits with multiple dataplanes

Yibo Guo
UC San Diego

William M. Mellette
inFocus Networks

Alex C. Snoeren
George Porter
UC San Diego

Abstract

Scale-out datacenter network fabrics enable network operators to translate improved link and switch speeds directly into end-host throughput. Unfortunately, limits in the underlying CMOS packet switch chip manufacturing roadmap mean that NICs, links, and switches are not getting faster fast enough to meet demand. As a result, operators have introduced alternative, *parallel* fabric designs in the core of the network that deliver N-times the bandwidth by simply forwarding traffic over any of N parallel network fabrics.

In this work, we consider extending this parallel network idea all the way to the end host. Our initial impressions found that direct application of existing path selection and forwarding techniques resulted in poor performance. Instead, we show that appropriate path selection and forwarding protocols can not only improve the performance of existing, homogeneous parallel fabrics, but enable the development of heterogeneous parallel network fabrics that can deliver even higher bandwidth, lower latency, and improved resiliency than traditional designs constructed from the same constituent components.

CCS Concepts: • **Networks** → **Network architectures**; **Data center networks**; **Network simulations**.

ACM Reference Format:
Yibo Guo, William M. Mellette, Alex C. Snoeren, and George Porter. 2022. Scaling beyond packet switch limits with multiple dataplanes. In *The 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22), December 6–9, 2022, Roma, Italy*. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3555050.3569141

1 Introduction

Bandwidth-hungry applications demand ever more from datacenter network fabrics. Starting from "data-intensive" applications like MapReduce [14] and Spark [46], moving to graph-traversal systems [13], and on to emerging large-scale machine learning training [31], capacity requirements only continue to increase. While standards bodies and device vendors continue to deploy faster NICs, links, and switches, the pace of these deployments has slowed. As a result, there are periods of time where datacenter bandwidth needs exceed the capabilities of commodity network gear. To temporarily bridge this gap, datacenter operators like Facebook [9] and LinkedIn [47] have chosen to deploy *explicitly parallel* backplanes by connecting top-of-rack (ToR) switches with multiple disjoint replicas of their fabrics (in both cases four 100-Gb/s fabrics, with LinkedIn's version shown in Figure 1), delivering higher capacity without increasing the link rates of constituent components.

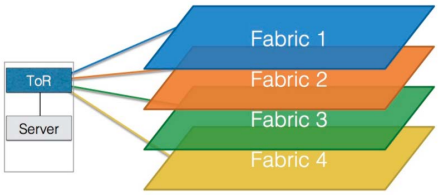


Figure 1. LinkedIn's 4-way parallel network [47]. Each fabric implements a 100-Gb/s fat tree [5] and is separate from one another except at the ToR switches.

We assume that there will continue to be an "ebb and flow" pattern between the expected bandwidth required by the operators and the delivered bandwidth provided by commodity networking equipment. To meet bandwidth demands during the "ebb" phase of this cycle, we propose a new class of *flattened* network topologies where each host—as opposed to ToR—is connected to N different disjoint network *planes*, each of which has its own set of switches and links connecting it to the other hosts in the network. We call these networks Parallel Dataplane Networks (P-Nets). Once a packet leaves the host in a P-Net and enters a given network plane, that packet cannot move to another network plane until it reaches its destination. This separation enables operators to linearly scale bandwidth by deploying multiple disjoint copies of their network—without having to insert switches or links between planes—lowering cost and energy demands.

However, the result is an explosion in the number of paths between end hosts, since not only does each network plane have multiple paths between a source and destination, but

This work is licensed under a Creative Commons Attribution International 4.0 License.

CoNEXT '22, December 6–9, 2022, Roma, Italy
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9508-3/22/12.
https://doi.org/10.1145/3555050.3569141

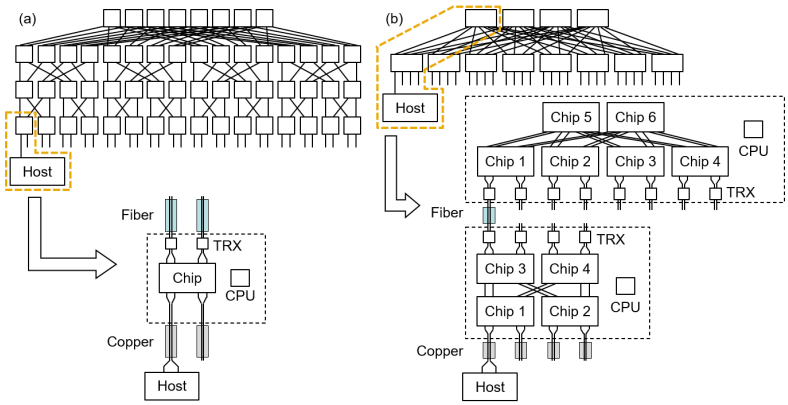


Figure 2. A (a) single-channel traditional and (b) single-channel chassis-based fat tree.

now there are multiple planes to choose from as well. Traditional fat tree networks already have multiple, equal-cost paths between hosts, and achieving full bisection bandwidth requires making effective use of them all [5]. Both operators and academics have developed a wide variety of techniques to approach the theoretical optimum in practice, with varying degrees of success. As we show in simulation, these techniques frequently struggle to achieve full performance when faced with the even larger set of choices in parallel network fabrics. Indeed, under certain conditions approaches like ECMP [1] barely leverage the added physical capacity.

We show that extracting the full, physical capacity of massively parallel P-Nets in practice requires explicitly striping traffic across the multiple planes in a strategic fashion. In particular, by using MPTCP [43] to multiplex flows across a bounded set of shortest paths, we are able to deliver performance similar to a traditional, single-plane scale-out network with upgraded link speeds. Not only can this additional sophistication unlock the extra capacity of additional, parallel network planes, but we observe that it also makes it possible to consider a further evolution in fabric topologies: parallel networks where the various forwarding planes are not simply identical copies of each other—as is inherent in a fat-tree-based approach—but instead, provide explicitly diverse physical connectivity. We refer to these networks as *heterogeneous* parallel networks. We further discover that such heterogeneity results in path-length variations across network planes, and by routing traffic over network planes with shorter paths to a given destination, we can reduce latency and bandwidth consumption, thereby also improving flow throughput. Even though such heterogeneous networks may be more complex to build, we find that recent optical interconnect technologies [3, 18] can dramatically simplify these operations and also lower power usage. Indeed, the gap between bandwidth demands and the delivered bandwidth of switch hardware can serve as an opportunity to explore alternative network architectures and protocols focused on improving application performance and efficiency.

In this paper, we carry out a study of parallel homogeneous fat trees as well as heterogeneous expander-based architectures. We explore issues including routing, forwarding, and transport protocol performance across different degrees of parallelism. The primary contributions of this work are: (1) a study of P-Net, a class of networks that achieves high bandwidth by using multiple network planes consisting of lower-speed, but also lower-cost-and-power commodity switches and cables, (2) examples of sub-optimal performance when adopting naive approaches to path selection and forwarding, and (3) a quantitative analysis of micro- and macro-application performance using both synthetic traffic and real datacenter flow traces, showing the benefits that heterogeneity can bring to parallel networks.

2 Motivation and background

The design of datacenter networks has been shaped by a combination of network bandwidth requirements from end hosts and the availability of high-speed merchant silicon packet switch chips. A driving factor in their design is ensuring that improvements in packet switching speed can be translated into increased end-host bandwidth. In this section, we describe this evolution and focus on recent scaling limits as motivation for deploying P-Nets.

2.1 The limitations of scale-out networks

Originally credited to Charles Clos [12], the observation that large switch fabrics can be composed of relatively small and inexpensive switches became relevant in datacenter network architecture with the advent of merchant silicon switch chips [5]. The structure of a folded-Clos network can be characterized by the number of tiers of switch chips that it requires, and how the chips are packaged into boxes. These design choices then dictate the number of hops a packet must traverse. Each additional tier incurs cost, power, latency, and cabling complexity, making it desirable to use the largest-radix commodity switches available.

Architecture	Tiers	Hops	Chips	Boxes	Links
Serial (scale-out)	4	7	3,584	3,584	24.6 k
Serial chassis	2	7	3,584	192	8.2 k
Parallel 8×	2	3	1,536	192	8.2 k

Table 1. Component counts for the two serial fat tree architectures shown in Figure 2 and parallel fat tree architecture shown in Figure 4. All networks have the same bisection bandwidth, with links in the 8× parallel networks optimized for deployment (section 6.1).

Figure 2(a) shows a small-scale illustration of a traditional folded-Clos or fat tree topology built from 4-port switches, with 32 end hosts and four tiers. As a more realistic—but difficult to illustrate—example, the components required to build an 8,192-end-host network out of 16-port switches are listed in the first row of Table 1. While small compared to today's largest networks, we use an 8,192-end-host exemplar network here because it allows for an "apples to apples" comparison between designs. As shown in the first row of Table 1, traditional fat tree designs have several shortcomings, including the deployment and maintenance overhead of many (often long, fiber-optic) links and the replication of packaging and ancillary hardware including CPUs, fans, power supplies, etc.

2.2 The adoption of chassis switches

The disadvantages of traditional fat-tree designs led several industrial players to design and build chassis-based fat trees [41] in which multiple switch chips are integrated into a common box, known as a chassis, and connected using energy- and cost-efficient copper backplane traces. By increasing the radix and thus the density of switching capacity, this architecture requires fewer optical transceivers and long fiber runs which reduces total hardware, power, and deployment costs. Figure 2(b) illustrates a 32-host fat tree built with a chassis architecture, including how switch chips are connected inside aggregation and spine chassis. This chassis-based architecture has been a critical factor in enabling large-scale datacenter deployments, where the costs of a traditional fat tree would be infeasible [10, 36].

The second row of Table 1 shows the components required to build an 8,192-node network out of 128-port chassis. Spine chassis use 24 16-port chips in a 3-stage internal Clos. Aggregation chassis do not need to be non-blocking, and are built with 16 16-port chips in a 2-stage topology. Despite the blocking aggregation chassis, the network as a whole retains the non-blocking property of Clos topologies, a fact leveraged in production networks [36]. Only two tiers of switch chassis are required, reducing the cabling by 1/3. While the number of switch chips remains constant, the number of discrete switch boxes is reduced by an order of magnitude.

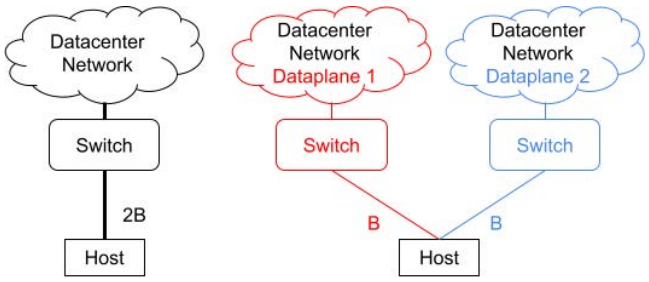


Figure 3. A "serial" network (left) uses high-bandwidth links throughout the topology, but requires high-cost and high-power chassis switches. A P-Net (right) uses lower-bandwidth links, allowing each switch to be implemented with a single lower-cost and lower-power switch chip.

2.3 Scaling beyond current chip limits

While the chassis architecture affords cost and power savings over a traditional scale-out fat tree, it has its own limitations. First, each chassis comes with a high power density, which presents scaling challenges as link speeds increase. (E.g., Facebook had to re-design their switches to fit in a hard 1750 W per-rack power limit [39].) Further, because the chassis architecture is simply a re-packaging of the traditional fat tree, it is ultimately subject to the same underlying scaling limitations in terms of switch chips and hops.

To circumvent these limitations, industrial datacenter network operators have recently begun to introduce parallelism to the core of their network designs, exposing the underlying per-port bandwidth to ToR switches rather than using chassis switches to implement a higher-bandwidth single link abstraction [9, 47]. In existing designs, hosts are connected to ToRs, which are then connected to multiple, independent network planes. StarDust [50] proposes a slightly different design that also opted for lower per-port bandwidth and higher radix, but instead form a single network in the core and rely on cell-based forwarding and simpler switch hardware design to scale the network. In this paper, we take this approach a step further, and consider what happens when end hosts are connected to each of the planes directly—i.e., ToRs are a member of only one plane.

3 Parallel Dataplane Networks

The basic concept of Parallel Dataplane Networks (or P-Nets) is relatively straightforward: rather than building a single "serial" high-link-speed network, the network is architected as multiple, disjoint lower-link-speed forwarding planes, or *dataplanes* for short, as illustrated logically in Figure 3. P-Nets extend parallelism all the way to end hosts, where each host has connections to N dataplanes via N ToR switches, albeit at $1/N\times$ the bandwidth per connection. The important distinction between a serial network and a parallel network is this: each dataplane runs like a traditional serial network, but packets cannot cross dataplanes, as they are logically separate. This leaves the end host to decide which dataplane(s)

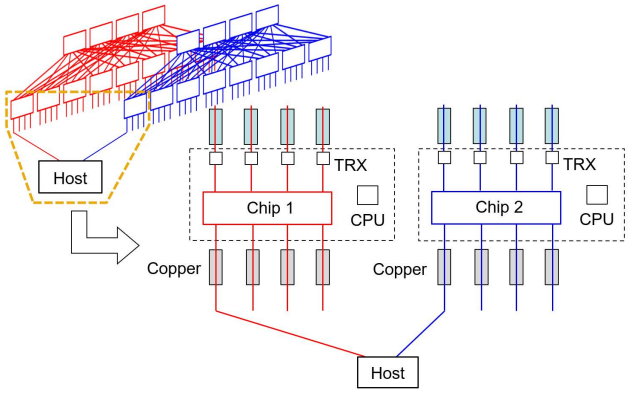


Figure 4. A two-way parallel fat tree/homogeneous P-Net

to send its traffic through, and once a packet leaves an end host and enters a particular dataplane, it stays within the dataplane until reaching the destination host.

Even though each dataplane only provides a fraction of the total uplink bandwidth, end hosts can still make use of the full bandwidth by employing multipath solutions like MPTCP [43]. In this section, we present the design of P-Nets, show the difference between traditional serial networks and our parallel networks, and explain how to adapt end hosts and applications to make full use of these parallel dataplanes.

3.1 Homogeneous P-Nets/ Parallel fat trees

Perhaps the simplest class of P-Nets are those that retain the same connectivity and physical structure among the dataplanes. We refer to these as homogeneous P-Nets, and the most straightforward example is a parallel fat tree. Figure 4 shows a 32-host parallel fat tree with two dataplanes; each host connects to both the red and blue dataplanes as shown. This fabric is constructed using the same switch chips as the networks in Figure 2, but in a different configuration. Note that in scale-out and chassis designs, although each switch chip has eight inputs, or internal ports, they only act as four ports, because every two internal ports are grouped into one high-speed port. This comes at the cost of lower radix, and chassis switches compensate for this by using two tiers and *four* chips per switch to maintain the radix of eight. The two-way parallel fat tree shown in Figure 4, on the other hand, breaks these grouped ports out to support the same number of hosts, and compensates for per-host bandwidth by connecting both planes to each host.

The design of a parallel fat tree results in a fabric with the same switching capacity as multistage chassis networks, but with fewer switch chips and lower power density. The parallel topology also reduces the number of hops as it maintains the switch chips' native radix by not grouping ports together and does not require multiple tiers of chips as in chassis switches. The final row of Table 1 compares the component usage of an 8× parallel fat tree to traditional serial and chassis designs, all with an equivalent number of end hosts and bisection bandwidth. Comparing the parallel fat tree to the chassis-based design, we see that P-Net enables significant hardware and power savings, while also reducing the number of hops. Note that a naive implementation of a parallel fat tree can increase the number of physical cables, but by leveraging the homogeneity of the network, we can group multiple low-bandwidth links using cable bundles to significantly reduce complexity (details in section 6).

3.2 Heterogeneous P-Nets

While parallel fat trees present an intuitive picture of parallel networks, P-Nets are not restricted to simply replicating identical copies of each dataplane. A prominent candidate for heterogeneous P-Nets is expander-graph-based networks. Due to their random [38] or pseudorandom [42] construction, we can create different instantiations for each dataplane, opening up new options for forwarding traffic.

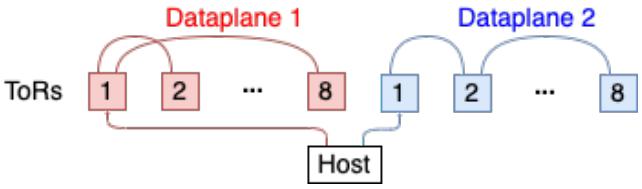
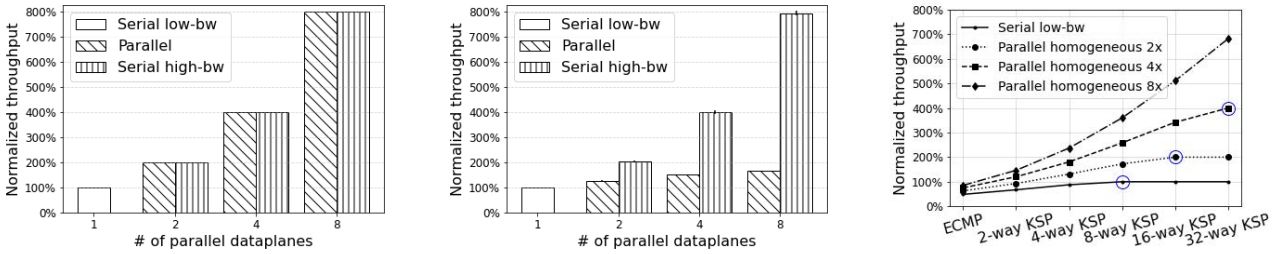


Figure 5. A two-way parallel heterogeneous expander

Figure 5 shows a P-Net built with two expander-based dataplanes, where each dataplane instantiates a different expander network among the ToR switches. Serial expander graphs are known to have short path lengths [38], but applying P-Net to expanders yields additional benefits because the probability of having a short path between a particular pair of racks grows with each additional data plane realization. Hence, a heterogeneous P-Net has the potential for significant latency improvement over a homogeneous P-Net. We evaluate these benefits in section 5 and address the practical issues of such heterogeneous topologies in section 6.

3.3 Implications on switches

Today's network switches can be configured to provide either fewer ports at higher speed (e.g. 32 ports at 400 Gb/s per port) or more ports at lower speed (e.g. 128 ports at 100 Gb/s per port) [8, 32]. Serial network designs like the scale-out and chassis-based fat trees shown in Figure 2 configure switches for lower radix, requiring more tiers of switches leading to high cost, power, and hop count. P-Nets, on the other hand, configure the constituent switches with a higher radix, allowing a significant reduction in the number of switching tiers and commensurate savings in cost, power, and hop count. P-Nets' multiple planes require only a linear multiple of switch chips, whereas traditional and chassis-based fat trees need even more due to additional tiers and denser switches. This allows P-Nets to scale similarly to [50], as both approaches opt for higher radix of the underlying chip.



(a) All-to-all throughput, ECMP **(b)** Permutation throughput, ECMP **(c)** Single-path vs multi-path

Figure 6. fat tree ideal throughput with ECMP (a and b), and performance scaling using multipath (c). Circled points indicate the multipath level needed to saturate P-Net. Throughput normalized against serial low-bandwidth.

3.4 Implications on end hosts

Multiple uplinks to ToRs While P-Nets can, in principle, have an arbitrary number of parallel dataplanes, in practice end hosts must have a sufficient number of uplinks to connect to all dataplanes. This can be achieved with multi-channel Ethernet links (details in section 6.1), multi-port NICs [4, 26] or multiple NICs per server. For practicality, we limit the number of dataplanes to $\leq$ 8, similar to how today's high-speed (e.g. 400G) ports aggregate up to 8 lower-speed ports. An even higher level of parallelism exacerbates the deployment complexity and often necessitates a generational improvement in the underlying silicon speed.

Utilizing multiple links in OSes At the OS level, we expose multiple dataplanes to end hosts at the IP layer following reasons: 1) this is the default way of accessing multiple ports on Linux; 2) by using different IPs per dataplane, we can use existing end-host routing solutions like DARD and Fastpass [33, 44] on each dataplane; 3) applications can decide which dataplane(s) to send their traffic through by using the appropriate IP address(es); 4) we can reuse existing multipath transport like MPTCP [43] to seamlessly adapt deployed applications. End hosts can also quickly detect individual dataplane failures via link status and avoid using the broken dataplane(s), allowing graceful performance degradation.

End-host routing solutions provide OS direct access to routing information and can facilitate better flow placement decisions in P-Net. This approach also avoids the limited memory constraint on commodity switches in order to support routing over multiple dataplanes.

Application interaction By default, round-robin is used for load balancing, but as we show in section 5, applications with special needs like low latency and high throughput can choose to use low-hop-count dataplane and to send traffic across multiple dataplanes to achieve their respective goals.

In practice, end hosts are aware of the topologies of all dataplanes in P-Net, and thus can provide pseudo/proxy interfaces like "low-latency" single-shortest-path and "high-throughput" multipath interfaces. Applications/ flows can use special tags like traffic classes to choose how to take advantage of the multiple dataplanes in P-Net.

4 Forwarding traffic over multiple dataplanes

The performance of a network depends not just on its topology, but also on the way that traffic is forwarded over that topology. There have been numerous studies of path selection algorithms for different topologies, and in this section, we discuss how to adapt these approaches to P-Net.

To start with, we first considered adapting ECMP [1]. In this case, each end host selects, for each flow, one of the N parallel dataplanes using a hashing algorithm. We simulated all-to-all and permutation traffic using ECMP on parallel fat trees (we defer the details to section 5.1.1), and Figure 6a and 6b show the achieved throughput plotted against that of a serial fat tree. We found that even though dense traffic patterns like all-to-all can fully saturate up to 8× parallel fat tree networks, sparser traffic patterns like permutation achieve minimal performance improvement when adding more dataplanes. Thus we concluded that naive ECMP-based routing is not adequate to fully exploit the combined capacity of these parallel dataplanes.

As single-path ECMP-based routing cannot fully utilize parallel dataplanes in P-Net, we next considered multipath routing and transport. Specifically, we looked at MPTCP [43] combined with K shortest paths (KSP) [2, 45] routing. The Jellyfish [38] work found that this provides a large improvement over ECMP + TCP in utilizing an increased number of paths. Figure 6c shows the throughput for the same permutation traffic in Figure 6b for parallel and serial fat trees, but this time using MPTCP and KSP, for various values of K. With MPTCP and KSP, sparse permutation traffic can now fully utilize the combined bandwidth of the parallel dataplanes. In fact, more parallel dataplanes demand higher multipath levels to saturate the network: 8-way multipath can fully utilize serial networks, but 2-dataplane P-Nets need 16-way multipath and 4-dataplane P-Nets need 32-way multipath.

Furthermore, we note that KSP is commonly employed in expander networks [38, 42], which are prime candidates of heterogeneous P-Nets, and Singla et al. have shown its good performance on serial expander networks [38]. Thus, we propose MPTCP + KSP as a promising approach for both

homogeneous and heterogeneous P-Nets. We analyze its performance in heterogeneous P-Nets in section 5, and propose a method for extracting the best performance for both short and long flows given MPTCP's limited ramp-up and convergence time.

5 Performance evaluation

Now that we have discussed how to build P-Nets, let's quantitatively evaluate how P-Nets perform relative to traditional single-dataplane networks. To evaluate the performance of P-Nets, we used a combination of synthetic workloads and real data center traffic distributions. The synthetic workloads consist of bulk and short flows, which represent the typical elephants and mice datacenter traffic, as well as application-like traffic patterns with multiple requests/responses. Real traffic traces include webserver/cache/Hadoop [35], datamining [22] and web search [6].

Depending on the traffic types, we use either linear programming (LP) solver [29] to measure the throughput, which allows us to scale much larger than packet simulators at high speed (100/400G), or packet simulator *htsim* [23] to capture latencies and flow completion times. Except otherwise noted, we repeat each experiment at least five times (each time with a newly instantiated topology) and plotted the standard deviations as error bars. We found little variation in most cases.

In our evaluations, we considered the following types of networks:

- **Serial low-bandwidth** network, which is a single network composed of (relatively) low capacity links. In our evaluation, we use 100Gb/s links (i.e. 1×100G).
- **Parallel homogeneous** network, or homogeneous P-Nets, which consists of N parallel dataplanes each with the same topology (e.g. fat tree). The links in each data plane run at 100Gb/s.
- **Parallel heterogeneous** network, or heterogeneous P-Nets, in which each of the N dataplanes implements a different topology (e.g. expander graph).
- **Serial high-bandwidth** network, which represents an ideal (but cost- and power-prohibitive network) with links running at $N \times 100$Gb/s.

Our goal here is to compare traditional serial networks with their parallel (homogeneous/heterogeneous) versions; specifically, we want to see whether P-Net can achieve similar or close performance of an ideal serial high-bandwidth network, and what other benefits/drawbacks does adding parallelism to the network bring. We do not attempt to directly compare fat trees with expanders.

Note that although we choose 100G networks as our baseline, most of the *throughput* evaluation results can apply to networks of arbitrary speed (25G, 40G, or more than 100G) and their parallel or high-bandwidth equivalent, since the benefits of parallelism come from using multiple networks planes, i.e. the degree of parallelism, not the actual speed.

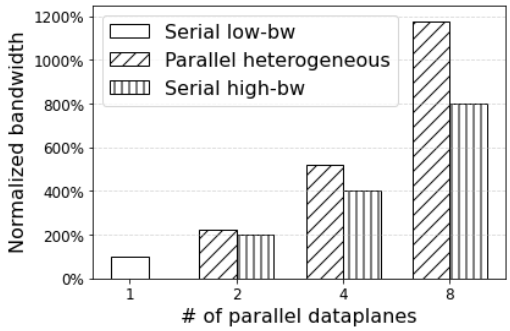


Figure 7. Ideal throughput on Jellyfish with *rack-level* all-to-all traffic.

5.1 Microbenchmark

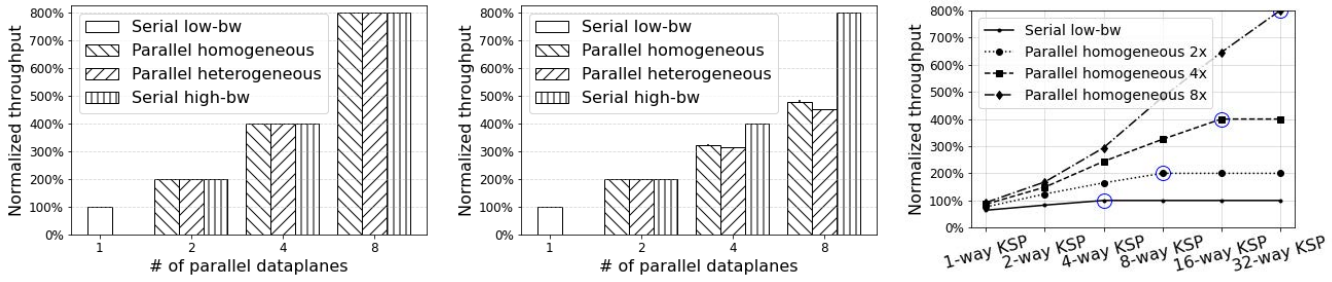
5.1.1 Bulk traffic throughput. One of the key questions we'd like to answer is whether parallel networks can achieve similar throughput as a serial high-bandwidth network. In this section, we compare the total throughput of flows under all-to-all and permutation traffic matrices.

We first measure the ideal throughput under no path constraint, which represents the total capacity of the network core. For this purpose, we run rack-level all-to-all traffic on a 128-rack fat tree and an equivalent Jellyfish (built with the same networking equipment, as described in [38]) using the LP solver. We plot the throughput on Jellyfish networks in Figure 7. Throughputs are normalized against serial low-bandwidth networks.

Because parallel heterogeneous networks using Jellyfish-like topologies have randomness[1] across the dataplanes, there may exist a shorter path for a given source/destination pair on another dataplane. Thus, each flow may consume less network capacity and thus, such heterogeneous parallel networks end up having even higher in-network capacity than their single-dataplane equivalent. Figure 7 shows that parallel Jellyfish networks can have up to 60% higher throughput than their serial high-bandwidth equivalent. Homogeneous P-Nets on both Jellyfish and fat tree topologies have the same throughput as serial high-bandwidth and are thus omitted.

Next, we take into account routing by simulating the ideal throughput with computed routes. This means we constraint the flows in LP solver to use the routes computed by ECMP or KSP. We ran all-to-all and permutation traffic on 1024-host fat trees and equivalent Jellyfish networks, and calculated the achieved throughput. The results are shown in Figure 6a and 6b for fat trees and Figure 8a and 8b for Jellyfish networks, respectively. We note that in both cases, dense traffic patterns like all-to-all can fully saturate all the parallel dataplanes, but for sparse traffic patterns like permutation traffic, standard ECMP for fat tree and default 8-way KSP for Jellyfish (which has been shown to have good performance in serial expander networks) cannot fully utilize the increased bandwidth from

[1]This is the major difference and advantage of parallel heterogeneous expander networks over simply-larger-radix serial expander networks.



(a) All-to-all throughput, 8-way KSP **(b)** Permutation throughput, 8-way KSP **(c)** Multipath performance scaling

Figure 8. Jellyfish ideal throughput with 8-way KSP (a and b), and performance scaling using multipath (c). Circled points indicate the multipath level needed to saturate P-Net. Throughput normalized against serial low-bandwidth.

multi-dataplane parallel networks. In parallel fat trees, ECMP improves the throughput minimally even with 8× as many dataplanes; and in parallel Jellyfish, the default 8-way KSP can only achieve about 60% of the total bandwidth.

Consequently, we quantitively evaluated the impact of multipath in P-Nets by varying the degree of multipath and measuring the throughput. We adjusted the routing parameter, ranging from single-path ECMP to K-way shortest paths for $K = 2, 4, 8...$ up to 32. The results are shown in Figure 6c for parallel fat tree networks and Figure 8c for parallel Jellyfish networks. As we can see, increasing the degree of multipath dramatically improves the achieved throughput, especially in the case of parallel networks. Furthermore, as highlighted in circles, more dataplanes demand a proportionally higher degree of multipath to fully utilize the combined bandwidth, i.e. P-Nets with N dataplanes need N times as many subflows. This is consistent with the intuition of using KSP in expander networks, which says more subflows are needed to saturate the increased number of uplinks/paths in expanders. In P-Net, the same reasoning applies: more dataplanes means more paths available; thus more subflows are needed to use the combined bandwidth of all dataplanes.

5.1.2 Short flow completion time. To capture the behavior of short flows, we set up a random permutation traffic pattern in a 686-host Jellyfish network using our packet simulator "htsim" [23], and measured the flow completion time as we vary the flow sizes from $100kB$ to 1GB. For this evaluation, we use P-Nets with four dataplanes.

Figure 9 compares the flow completion times (FCTs) of the four different types of networks. We also varied the degree of multipath and found that single-path routing generally gives the lowest FCTs in both serial networks whereas 4-way KSP gives the lowest FCTs in both parallel networks. Thus we plotted these results using these best settings for the respective networks.

We note that for smaller flow sizes up to 10MB, parallel networks have surprising advantages over serial networks and even outperforms serial high-bandwidth network. Upon deeper investigation, we found that these flows are small

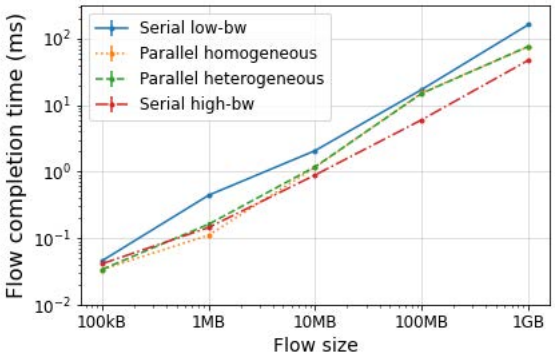


Figure 9. Small flow FCT with varying flow sizes.

enough that they can finish before hitting full queues and causing retransmits, i.e. before reaching TCP steady state. In such cases, flows in parallel networks have the advantage of using more paths in TCP slow start. However, one should consider the various factors at play at such small scales, which include slow start behavior, the retransmit timeout (we tuned to 10ms as suggested in DCTCP [6]), network load or switch queue utilization, etc. Also, as shown in [15, 16, 49], MPTCP can often hurt short flows, which happens here in serial networks. Thus, one should take care in using MPTCP for these really small flows (including sub-100kB flows), and their exact behavior in datacenters could demand further tuning and studies.

As flows grow larger to 100MB, they start to reach steady state and TCP/MPTCP was able to probe the bandwidth available across the dataplanes. Here, P-Nets have a smaller advantage over baseline serial low-bandwidth networks because MPTCP is slow to probe the optimal subflow bandwidth allocation at smaller time scales (only 100MB on a 4×100G network). Larger flows like 1GB on the other hand can realize more speedup over serial low-bandwidth because MPTCP can probe bandwidth better.

To summarize, we note that in steady state: relatively small flows achieve less improvement in parallel networks, whereas larger flows start to behave like bulk flows and can

benefit from using more paths in P-Nets. Thus we empirically choose 100MB as a threshold for small versus large flows. Flows smaller than or equal to 100MB benefit less from multipath using MPTCP and thus should use single-path routing; flows larger than or equal to 1GB can significantly improve their performance by using multipath and thus should do so.

5.2 Simulated workloads

5.2.1 Small RPCs. As discussed in the previous section, MPTCP could hurt the performance of small flows. This is especially true for small RPCs that can be a few packets or even less than an MTU-sized packet. For these small packets, we choose to run single-path ECMP. Because of the random nature of expander graphs and potentially shorter paths in another dataplane for any given source and destination, there could be another way to improve RPC latency in such networks, i.e. by using shorter paths available in heterogeneous P-Nets.

To verify our hypothesis, we set up a packet simulation using a 686-host Jellyfish network and a ping-pong type RPC application. Again we use P-Nets with four dataplanes. Each host will send an RPC request of MTU size (1500B) to a random destination server, wait for the response and measure the end-to-end request completion time over 1000 rounds.

Figure 10 shows the request completion time distribution in CDF and Table 2 summarizes the statistics.

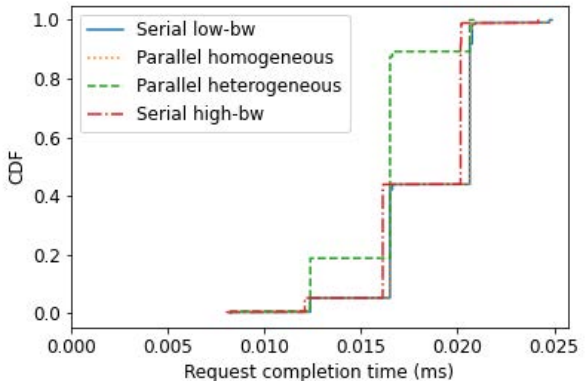


Figure 10. 1500B RPC request completion time, single-path routing. Parallel homogeneous mostly overlaps with serial low-bw.

The stepping curves come from the different hop count distributions in these networks. As we can see, serial networks and parallel homogeneous networks have identical hop count distribution as they use the same topology, whereas parallel heterogeneous networks use a mix of four different instantiations of Jellyfish. And as we suspected, for any given source and destination, there may exist a shorter path on the three additional dataplanes, compared with the other three types of networks. Thus parallel heterogeneous networks have the unique advantage in terms of lower hop count, which translates to lower latency and RPC completion time.

Network setup	Median	Average	99%-tile
Serial low-bw	100%	100%	100%
Parallel homogeneous	100%	99.2%	100%
Parallel heterogeneous	80.1%	86.6%	90.4%
Serial high-bw	98.1%	97.9%	97.4%

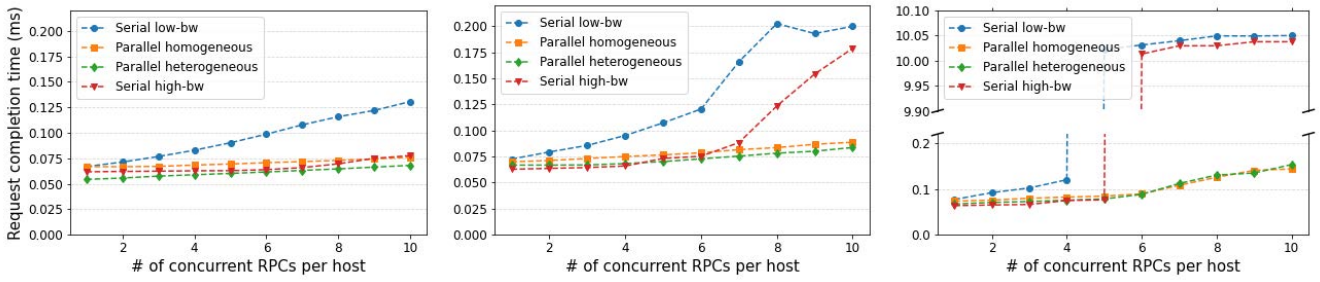
Table 2. 1500B RPC request completion time statistics, using serial low-bw as baseline

The minor difference between serial low-bandwidth and parallel homogeneous comes from the fact that parallel homogeneous networks have more paths, thus reducing the chance of flow collision given the same number of flows.

The serial high-bandwidth network has slightly less serialization delay per hop, as its links run at 400G instead of 100G in the other three networks. However, at 100G, MTU-sized packets only take 1500B/100Gb/s = 120ns; at 400G, it's only 1/4 of that. Thus serial high-bandwidth networks can only reduce the latency by 90ns per hop. Such improvement in serialization delay is relatively small compared with the propagation delays in modern datacenters. Assuming 200m per switch hop in the core, each hop will introduce a whole microsecond, which is 11× the serialization delay improvement in serial high-bandwidth networks. Thus overall, parallel heterogeneous networks can achieve much lower latency for small RPCs by using these shorter paths. As links become faster, the advantage of high-bandwidth networks diminishes, but propagation delay is fixed by the law of physics; thus the reduction in path length will further improve the end-to-end latency for small RPCs in high-speed networks.

In addition to single RPC evaluations, we also experimented with multiple concurrent RPCs. Since P-Net has more paths, we suspect that it is possible to handle more small RPC requests with less queuing behind other requests than in serial networks.

In this experiment, we use the same network setup, but with 100kB-sized requests, and we vary the number of concurrent RPCs per host from 1 to 10. Figure 11 shows the median, 90th and 99th percentile of RPC request completion time. As we increase the number of concurrent RPCs, the request completion times also increase. Serial low-bandwidth suffers the most, as there are both limited bandwidth to drain these packets in the queue and a limited number of paths to avoid path collision that leads to queue buildup. Serial high-bandwidth networks reduce this problem only slightly by draining the queue faster. Parallel networks, on the other hand, have the advantage of 4× the number of paths, allowing concurrent RPC requests to spread across these separate links and queues. This results in both 1) less queue buildup and only a mild increase in request completion time, and 2) less chance to run into full queues, packet drops, and retransmits, as shown in Figure 11c.



(a) Median **(b)** 90%-tile **(c)** 99%-tile

Figure 11. Concurrent RPC request completion time. Note the broken axis in (c).

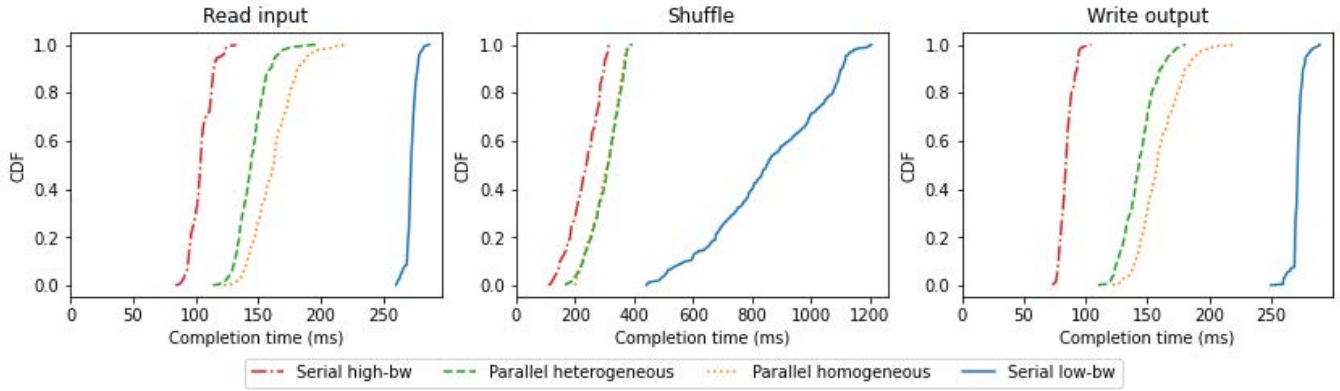
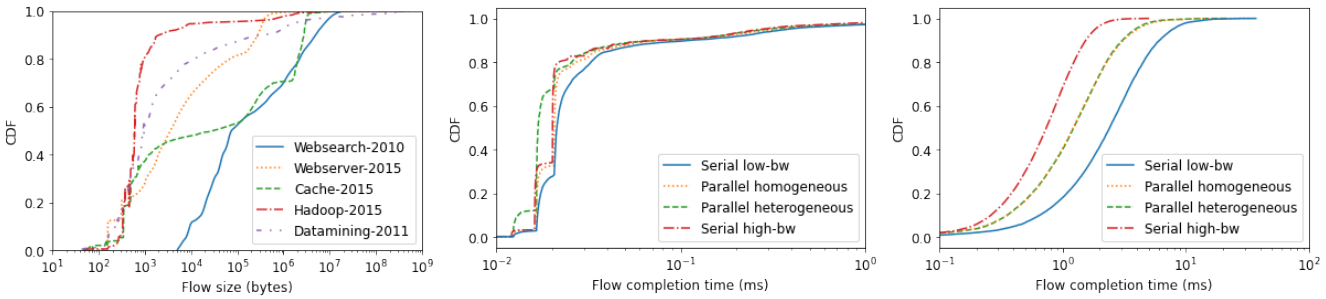


Figure 12. Simulated Hadoop-like workload per-worker completion time in each stage, single-path routing.

5.2.2 Shuffle workloads. Inspired by Hadoop-type large-scale data analytics jobs, we simulated similar workloads involving loading data from remote racks, shuffling data among workers , and writing output data to replicas.

We simulated the Hadoop traffic of a sorting application in a 250-host cluster, in which we distribute 100G data to 32 mappers and 32 reducers. Each mapper loads data in blocks of 128MB, spreads the entries into 32 buckets, and sends each bucket to one reducer to merge and sort all entries in that bucket. We assume the entries are randomly distributed; thus, the shuffle stage consists of 32×32 flows of the same size, one per (mapper, reducer) combination. After a reducer completes sorting, it will write to a replica in a random rack. We configured our mappers and reducers to read/write 4 concurrent blocks at a time to avoid sending too many flows at once. For the read input and write output stages, each flow corresponds to one block, which is 128MB. For the shuffle stage, each flow gets $1/(32 \times 32)$ of the total 100GB of data, which is roughly 100MB. Since these are relatively short flows in our 100/400G network setup as we showed earlier, we choose to run single-path routing for these flows.

Figure 12 shows the distribution of observed network request completion times per worker, which is the total time it takes each mapper/reducer to load, shuffle, or write all the data it handles. We measure this at each mapper for the read input and shuffle stages and at each reducer for the write output stage. In stage 1 read input, as there are fewer hosts sending traffic, flows in parallel networks have less chance of colliding with other flows; thus they observe reduced overall worker completion time. Furthermore, flows in parallel heterogeneous networks can utilize shorter paths in sparse traffic settings and achieve lower flow completion times. In stage 2 shuffle, the serial low-bandwidth network suffers more from condensed traffic and has a long tail. The dense traffic allows flows to take close-to-full advantage of parallel networks and achieve closer to ideal serial high-bandwidth network performance. However, parallel heterogeneous networks do not exhibit additional advantages over their parallel homogeneous counterparts. This is because dense traffic like all-to-all in the shuffle makes more flows collide as they try to take advantage of the shorter paths. Furthermore, these flows are approximately the same size as the single-vs-multipath cutoff threshold. Increasing the flow sizes in high-speed (100/400G) networks by multiplexing multiple block transfers per flow would increase the performance of this type of application. In stage 3 write output, each reducer writes to a random remote host designated to store that block; thus the traffic pattern looks like the inverse of stage 1 read input. Thus, we observe similar performance as in stage 1.



(a) Published DC flow size CDF **(b)** Datamining trace [22] FCT **(c)** Websearch trace [6] FCT

Figure 13. Flow size distribution of published DC flow traces [6, 22, 35], and key evaluation results.

5.3 Published datacenter flow traces

We also used datacenter flow traces [6, 22, 35] to confirm our findings in section 5.1 and section 5.2. We plotted the flow size distributions of webserver/cache/Hadoop [35], datamining [22] and web search [6] applications in Figure 13a.

For this experiment, we used a similar setup as in section 5.1.2, with individual flow sizes drawn from the distributions in these traces, as shown in Figure 13a. We set up four concurrent flows per host to saturate the network and each flow runs in a closed loop using single-path routing.

For space reasons, we only show the result on Jellyfish networks at 100/400G for Datamining from VL2 [22] and Websearch from DCTCP [6]. But these two traces are representative across all traces, as they cover both small flows and large flows shown in Figure 13a. The fat tree results look similar to Jellyfish ones (except there are no parallel heterogeneous fat trees) and the smaller traces behave similarly to the datamining traffic. Interested readers can find the full result in Appendix A.

The results shown in Figure 13b and Figure 13c confirm our previous studies using synthetic traffic. In particular, Figure 13b shows that similar to the RPC-like application in section 5.2.1, short flows like the Datamining traffic can achieve lower latency on P-Nets, especially parallel heterogeneous networks, by exploiting the lower average hop count and better tolerance of multiple concurrent flows. Figure 13c, on the other hand, shows that similar to the simulated shuffle traffic in section 5.2.2, P-Net can achieve significant throughput improvement over serial low-bandwidth networks and closer to ideal high-throughput serial networks. A similar conclusion can be drawn for flows drawn from the other traces, as discussed in detail in Appendix A.

5.4 Fault tolerance

Another important benefit of P-Net is its better resiliency against network failures. Because P-Net consists of multiple dataplanes and thus more paths, it is less likely to lose all the shortest paths than serial networks with a single dataplane. This allows for more graceful performance degradation than traditional serial networks.

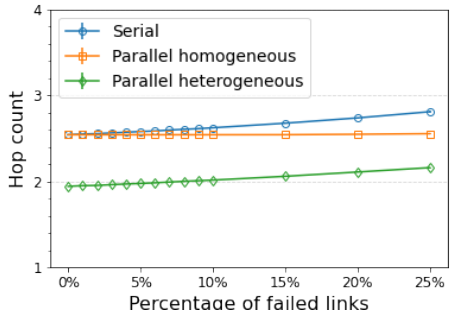


Figure 14. Average hop count across all src/dst pairs. Higher hop count indicates increased latency.

In Figure 14, we compare the impact on average hop count of all-pairs shortest paths in a set of Jellyfish networks with 686 hosts: serial, parallel homogeneous, and parallel heterogeneous. Note that serial low-bandwidth and serial high-bandwidth networks only differ in link speed, and are identical in hop count distribution; thus, we do not differentiate them. The two parallel Jellyfish networks both have 4× dataplanes. Link failures are random across the network.

We observe that serial networks lose short paths very fast, which leads to 22% more hops with 40% link failures, whereas parallel homogeneous networks only suffer by 3%. Parallel heterogeneous networks also lose the really short paths fast; thus, their advantage diminishes quickly as more link fails, but they still outperform the other two types. We note that expander networks like Jellyfish are already highly resilient to failures [38], but P-Nets further improve upon it.

In reality, this can be extremely helpful when operators deploy expander networks in some dataplanes of P-Net to support low-latency traffic like web search. The path length advantage of these expander networks, combined with the high failure resiliency of P-Net, can provide the lowest latency even in case of multiple network failures. Furthermore, P-Nets' rack-level network redundancy removes a major single point of failure in today's datacenter networks [28].

6 Practical Considerations

P-Net proposes a new networking solution by deploying multiple dataplanes, but operators may find it difficult to

deploy multiple copies of switches, cables, NICs, etc. Here, we discuss some of the optimizations to alleviate these problems.

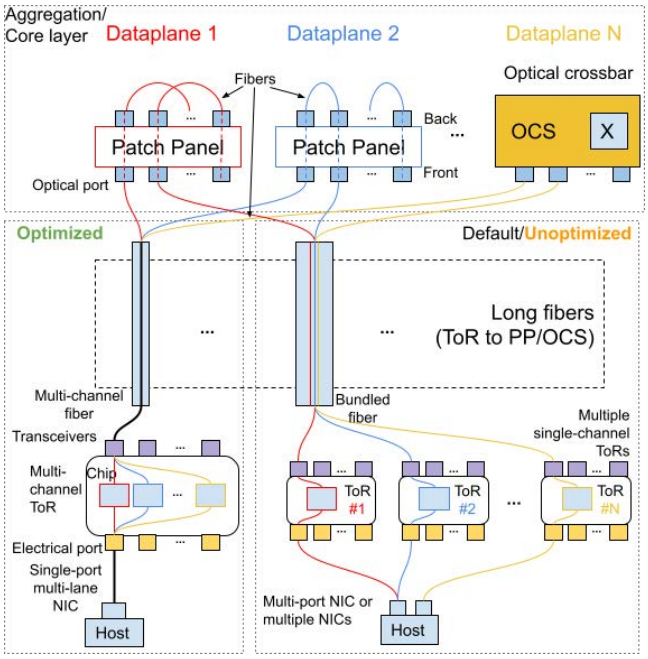


Figure 15. Deployment optimization of P-Net

6.1 Network deployment

First of all, P-Net configures switches for higher radix and lower per-port speed, and uses multiple sets of switches and cables to achieve equivalent total bandwidth of a traditional network. This comes at a cost of more distinct elements, e.g. switch boxes and fiber cables. Fortunately, modern networking and deployment solutions can greatly reduce redundant efforts. The top and middle part of Figure 15 shows the deployment optimization for P-Nets.

For the aggregation/core layer, we use optical patch panels and optical circuit switches (OCSes) to simplify wiring. We borrow the idea of patch panels from [36] and [48], which has been shown to 1) significantly reduce wiring complexity by operating only on the patch panels, 2) leave room for more aggregation layers, and 3) allow easier reconfiguration and expansion. More importantly in P-Net, this enables us to "hide" the heterogeneity in heterogeneous P-Nets, as we show next in section 6.2. OCSes like Calient/Palomar and rotor switches can provide additional improvements over patch panels by replacing the back-side wirings with either software [3, 34] or pre-etched gratings [18]. Furthermore, by adopting optical switching, we can eliminate transceivers in the network core. This has been shown to be a key scaling mechanism into Terabit ethernet, as high-speed packet switches and transceivers consume extremely high power [21, 27, 37].

For switches, by default, P-Nets use separate ToR switches and fiber cables to improve redundancy, as shown in the right dotted box. Each color represents one dataplane, with its own fibers connecting separate ToR switches and the aggregation/core layer on top. This provides the highest redundancy, at the cost of increased wiring efforts.

P-Nets can also take advantage of modern multi-channel Ethernet technologies [8] and coalesce links from multiple dataplanes into a single physical cable, as shown in the left dotted box. We can deploy $4x100$G P-Nets using the four 100G channels in 400G Ethernet cables [8, 32]. We split long-running fibers at the end and connect individual channels to different patch panels or OCSes. On the switch side, modern multi-channel switches already split these channels at the chip level, as in scale-out and chassis networks. P-Nets can adopt the same design, but with a flattened layer of chips inside each switch box to reduce chip count and power density, while still providing connectivity to multiple dataplanes.

Last but not least, by using multiple dataplanes, P-Net allows operators to upgrade one dataplane at a time, without bringing down the entire network. Furthermore, software-controlled OCSes together with the incremental expansion support of expander-based networks means operators can more easily scale up their network.

6.2 "Hiding" heterogeneity

The concept of heterogeneous P-Nets may worry operators and network designers, as deploying and managing N different networks seem very challenging, but here we present ways to reduce that heterogeneity to the minimal level.

As we discussed above, patch panels and OCSes allow operators to *localize* the heterogeneity across dataplanes to a central location. This means that the long-running fibers and per-rack layout can be kept exactly the same, keeping deployment complexity across the datacenter floor low. Furthermore, by using software-controlled OCSes or pre-etched gratings that encode the connectivity, operators can completely hide the hardware heterogeneity in P-Nets.

6.3 End host/NIC support

For end hosts, we can further reduce the rack-level wiring complexity by using single-port-multi-channel NICs like the HPE 4 × 25Gb 1-port 620QSFP28 adapter [17] and FPGA-based NICs like Corundum [19]. The former provides four separate 25G channels to a server using a single 100G port at an extremely affordable price, and the latter provides similar multi-channel functionality with more flexibility. Operators can balance between ToR redundancy and cost by varying the number of physical uplinks.

6.4 End-host burst capacity

We note that the design of P-Net limits the per-port bandwidth to $1/N\times$ of the state of the art, which may hurt the performance of high-throughput applications like GPUs, TPUs and other accelerators. However, first, we observe that very few datacenter applications can saturate a single 100+G link

by itself [11, 35]. Second, multiple flows per host can be effectively spread across the parallel dataplanes in P-Nets. Lastly, these high-throughput applications often use specialized interconnect solutions anyway, due to their unique communication patterns and requirements, as shown in Google's 2D and 3D Torus networks used in their TPU clusters [25, 27].

6.5 Incast traffic

For incast scenarios, P-Net can spread the traffic across separate dataplanes to alleviate congestion in the network, but careful coordination is still needed to avoid overrunning end host NIC buffers. We defer this to future studies that might involve incast-aware transports like DCTCP [6] or NDP [23].

7 Future work and opportunities

Monitoring and diagnostics: P-Net's adoption of multiple dataplanes brings management and diagnostic challenges, since each dataplane is logically separate and often belongs to a different control domain. Existing systems will need to merge flow statistics from multiple dataplanes to accurately describe the network state and troubleshoot issues.

P-Net with different topology types: Although we discussed parallel heterogeneous networks, all dataplanes have the same type of topology. Another type of parallel heterogeneous network can consist of entirely different topologies across the dataplanes. For example, operators can deploy a combination of expander-based topologies and fat trees to handle both low-latency traffic and Hadoop-like data-intensive workloads. The major difficulty then becomes managing a mixture of network topologies within a datacenter.

Performance isolation: Because P-Net has multiple isolated dataplanes, operators can assign different traffic classes to different dataplanes to achieve performance isolation. For example, user-facing frontend traffic can be assigned to one dataplane, and background data analysis traffic can be assigned to another. Traffic from different tenants can also be assigned to different dataplanes to avoid interference. This strict performance isolation opens up new opportunities for control-plane solutions like pFabric [7], EyeQ [24] and pHost [20], which attempts to support a mixture of elephants and mice traffic on a single fabric.

8 Related work

Parallel dataplane networks have begun to be adopted by the industry as we move towards higher link speeds. Facebook [9] and LinkedIn [47] have built parallel fat trees to achieve equivalent 400G speed in the network core using 4×100G dataplanes. Compared with P-Nets, their topologies are homogeneous, and end hosts are limited to using one of the dataplanes via ECMP. This leads to lower operational costs than chassis networks, but they do not have the additional benefits of heterogeneous P-Nets.

In the research community, our previous workshop paper [30] also explored parallel networks design, but was limited to parallel fat trees. This paper extends on it by 1) further exploring parallel heterogeneous networks, in which flows can pick the dataplane(s) with the shortest hops to achieve lower latency and higher total throughput, 2) studying and answering practical issues of heterogeneous P-Net deployment, and 3) empirically evaluating the performance of several datacenter network traffic patterns in P-Nets.

P-Net is not the first to use multiple uplinks in datacenter environment. GRIN [4] and Subways [28] are two proposals that utilize multiple NICs/ports per end host to improve network performance. GRIN [4] connects servers in the same or adjacent racks together using the additional ports, allowing each end host to opportunistically send/receive traffic via its paired neighbor. It can boost end-host burst capacity at the cost of additional cross-rack wirings. Subways [28] takes a slightly different approach by connecting end hosts directly to the neighboring rack's ToR switch. It uses adaptive load balancing to lower congestion, and similar to P-Net, provides better fault tolerance via redundant ToR switches.

Stardust [50] is another datacenter network design that separates a single link into multiple lower-speed links. It proposed a new Clos-based network design that uses cell-based forwarding, simplified routing, and redesigned hardware to improve the performance and power savings of datacenter networks, but is still a serial network with one dataplane. P-Net focuses on an orthogonal problem of datacenter network design, which is whether and how can we use multiple dataplanes to support growing application demand.

9 Conclusions

Evolving datacenter networks to meet the bandwidth demands of next-generation systems is a challenge, in part due to limited scaling of packet switch chip speed. Careful packaging of switch chips within chassis helps reach part of the scaling goal, but becomes cost- and energy-prohibitive at future link speeds. In this paper, we explored the space of parallel dataplane networks (P-Net) to add a new degree of scaling to network designs, which helps bridge the gap between fast-growing application demand and current-generation hardware capacity. We showed that naive routing in P-Nets leads to sub-optimal performance, and that with careful path selection and multipath transport, we can scale beyond existing switch chip bandwidth limit and achieve low latency. By eschewing the idea of a single network, P-Net can achieve fewer hops, higher fault tolerance and lower power consumption with minimal added deployment complexity.

Acknowledgement

We would like to thank our shepherd Isaac Keslassy and anonymous CoNEXT reviewers for their valuable feedback. This work was supported by NSF via grant CNS-1911104.

References

[1] 2014. 802.1QBP - equal cost multiple paths. https://www.ieee802.org/1/pages/802.1bp.html

[2] 2021. Google code archive - long-term storage for Google code project hosting. https://code.google.com/archive/p/k-shortest-paths/

[3] 2021. S Series Optical Circuit Switch | CALIENT Technologies. https://www.calient.net/products/s-series-photonic-switch/

[4] Alexandru Agache, Razvan Deaconescu, and Costin Raiciu. 2015. Increasing Datacenter Network Utilisation with GRIN. In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (Oakland, CA) (NSDI'15). USENIX Association, USA, 29–42.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. In Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (Seattle, WA, USA) (SIGCOMM '08). Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/1402958.1402967

[6] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In Proceedings of the ACM SIGCOMM 2010 Conference (New Delhi, India) (SIGCOMM '10). Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/1851182.1851192

[7] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. PFabric: Minimal near-Optimal Datacenter Transport. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (Hong Kong, China) (SIGCOMM '13). Association for Computing Machinery, New York, NY, USA, 435–446. https://doi.org/10.1145/2486001.2486031

[8] Ethernet Alliance. 2020. The 2020 Ethernet Roadmap. https://ethernetalliance.org/technology/2020-roadmap/

[9] Alexey Andreyev, Xu Wang, and Alex Eckert. 2019. Reinventing our data center network with F16, Minipack. https://engineering.fb.com/data-center-engineering/f16-minipack/

[10] Yuval Bachar. 2018. Introducing "6-pack": the first open hardware modular switch. https://engineering.fb.com/2015/02/11/production-engineering/introducing-6-pack-the-first-open-hardware-modular-switch/

[11] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding Host Network Stack Overheads. In Proceedings of the 2021 ACM SIGCOMM 2021 Conference (Virtual Event, USA) (SIGCOMM '21). Association for Computing Machinery, New York, NY, USA, 65–77. https://doi.org/10.1145/3452296.3472888

[12] Charles Clos. 1953. A study of non-blocking switching networks. The Bell System Technical Journal 32, 2 (1953), 406–424. https://doi.org/10.1002/j.1538-7305.1953.tb01433.x

[13] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. 2015. One Trillion Edges: Graph Processing at Facebook-Scale. Proc. VLDB Endow. 8, 12 (Aug. 2015), 1804–1815. https://doi.org/10.14778/2824032.2824077

[14] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (San Francisco, CA) (OSDI'04). USENIX Association, Berkeley, CA, USA, 137–149. http://static.usenix.org/event/osdi04/tech/full_papers/dean/dean.pdf

[15] Pingping Dong, Wenjun Yang, Wensheng Tang, Jiawei Huang, Haodong Wang, Yi Pan, and Jianxin Wang. 2018. Reducing transport latency for short flows with multipath TCP. Journal of Network and Computer Applications 108 (2018), 20 – 36. https://doi.org/10.1016/j.jnca.2018.02.005

[16] P. Dong, W. Yang, K. Xue, W. Tang, K. Gao, and J. Huang. 2019. Tuning the Aggressive Slow-Start Behavior of MPTCP for Short Flows. IEEE Access 7 (2019), 6010–6024. https://doi.org/10.1109/ACCESS.2018.2889339

[17] Hewlett Packard Enterprise. 2021. HPE Ethernet 4x25Gb 1-port 620QSFP28 Adapter - Overview. https://support.hpe.com/hpesc/public/docDisplay?docId=emr_na-c05220334

[18] Y. Shaya Fainman, Joseph Ford, William M. Mellette, Shayan Mookherjea George Porter, Alex C. Snoeren, George Papen, Saman Saeedi, John Cunningham, Ashok Krishnamoorthy, Michael Gehl, Christopher T. DeRose, Paul S. Davids, Douglas C. Trotter, Andrew L. Starbuck, Christina M. Dallo, Dana Hood, Andrew Pomerene, and Anthony Lentine. 2019. LEED: A Lightwave Energy-Efficient Datacenter. In 2019 Optical Fiber Communications Conference and Exhibition (OFC). 1–3.

[19] Alex Forencich, Alex C. Snoeren, George Porter, and George Papen. 2020. Corundum: An Open-Source 100-Gbps Nic. In 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). 38–46.

[20] Peter X. Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. PHost: Distributed near-Optimal Datacenter Transport over Commodity Network Fabric. In Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (Heidelberg, Germany) (CoNEXT '15). Association for Computing Machinery, New York, NY, USA, Article 1, 12 pages. https://doi.org/10.1145/2716281.2836086

[21] Manya Ghobadi, Ashkan Seyedi, Chongjin Xie, Hong Liu, and Rob Stone. 2021. ACM SIGCOMM 2021 Workshop on Optical Systems: Industrial Panel. (Aug 2021). https://conferences.sigcomm.org/sigcomm/2021/workshop-optsys.html

[22] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A Scalable and Flexible Data Center Network. In Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (Barcelona, Spain) (SIGCOMM '09). Association for Computing Machinery, New York, NY, USA, 51–62. https://doi.org/10.1145/1592568.1592576

[23] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (Los Angeles, CA, USA) (SIGCOMM '17). Association for Computing Machinery, New York, NY, USA, 29–42. https://doi.org/10.1145/3098822.3098825

[24] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, Albert Greenberg, and Changhoon Kim. 2013. EyeQ: Practical Network Performance Isolation at the Edge. In 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). USENIX Association, Lombard, IL, 297–311. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/jeyakumar

[25] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. 2020. A Domain-Specific Supercomputer for Training Deep Neural Networks. Commun. ACM 63, 7 (June 2020), 67–78. https://doi.org/10.1145/3360307

[26] D. Li and J. Wu. 2014. On the design and analysis of Data Center Network architectures for interconnecting dual-port servers. In IEEE INFOCOM 2014 - IEEE Conference on Computer Communications. 1851–1859.

[27] Hong Liu. 2021. ACM SIGCOMM 2021 Workshop on Optical Systems invited talk: The Evolving Role of Optics for Datacenter Network and Machine Learning. https://conferences.sigcomm.org/sigcomm/2021/workshop-optsys.html

[28] Vincent Liu, Danyang Zhuo, Simon Peter, Arvind Krishnamurthy, and Thomas Anderson. 2015. Subways: A Case for Redundant, Inexpensive Data Center Edge Links. In Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (Heidelberg, Germany) (CoNEXT '15). Association for Computing Machinery, New York, NY, USA, Article 27, 13 pages. https://doi.org/10.1145/2716281.2836112

[29] Gurobi Optimization LLC. 2021. Gurobi Optimizer. https://www.gurobi.com/products/gurobi-optimizer/

[30] William M. Mellette, Alex C. Snoeren, and George Porter. 2016. P-FatTree: A Multi-channel Datacenter Network Topology. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets-XV). Atlanta, GA.

[31] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized Pipeline Parallelism for DNN Training. In Proceedings of the 27th ACM Symposium on Operating Systems Principles (Huntsville, Ontario, Canada) (SOSP '19). Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3341301.3359646

[32] Arista Networks. 2019. Arista 7368X4 Series. https://www.arista.com/en/products/7368x4-series

[33] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: A Centralized "Zero-Queue" Datacenter Network. In Proceedings of the 2014 ACM Conference on SIGCOMM (Chicago, Illinois, USA) (SIGCOMM '14). Association for Computing Machinery, New York, NY, USA, 307–318. https://doi.org/10.1145/2619239.2626309

[34] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. 2022. Jupiter Evolving: Transforming Google's Datacenter Network via Optical Circuit Switches and Software-Defined Networking. In Proceedings of the ACM SIGCOMM 2022 Conference (Amsterdam, Netherlands) (SIGCOMM '22). Association for Computing Machinery, New York, NY, USA, 66–85. https://doi.org/10.1145/3544216.3544265

[35] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In Proceedings of the ACM SIGCOMM Conference. London, England.

[36] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (London, United Kingdom) (SIGCOMM '15). Association for Computing Machinery, New York, NY, USA, 183–197. https://doi.org/10.1145/2785956.2787508

[37] Rachee Singh, Nikolaj Bjorner, Sharon Shoham, Yawei Yin, John Arnold, and Jamie Gaudette. 2021. Cost-Effective Capacity Provisioning in Wide Area Networks with Shoofly. In Proceedings of the 2021 ACM SIGCOMM 2021 Conference (Virtual Event, USA) (SIGCOMM '21). Association for Computing Machinery, New York, NY, USA, 534–546. https://doi.org/10.1145/3452296.3472895

[38] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. 2012. Jellyfish: Networking Data Centers Randomly. In Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). USENIX, San Jose, CA, 225–238. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/singla

[39] Rob Stone. 2021. ACM SIGCOMM 2021 Workshop on Optical Systems invited talk: Co-packaged Optics in the Data Center. https://conferences.sigcomm.org/sigcomm/2021/workshop-optsys.html

[40] Ole Tange. 2018. GNU Parallel 2018. Ole Tange. https://doi.org/10.5281/zenodo.1146014

[41] Amin Vahdat, Mohammad Al-Fares, Nathan Farrington, Radhika Niranjan Mysore, George Porter, and Sivasankar Radhakrishnan. 2010. Scale-Out Networking in the Data Center. IEEE MICRO 30, 4 (Aug. 2010), 29–41.

[42] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. 2016. Xpander: Towards Optimal-Performance Datacenters. In Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (Irvine, California, USA) (CoNEXT '16). Association for Computing Machinery, New York, NY, USA, 205–219. https://doi.org/10.1145/2999572.2999580

[43] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. 2011. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (Boston, MA) (NSDI'11). USENIX Association, USA, 99–112.

[44] Xin Wu and Xiaowei Yang. 2012. DARD: Distributed Adaptive Routing for Datacenter Networks. In 2012 IEEE 32nd International Conference on Distributed Computing Systems. 32–41. https://doi.org/10.1109/ICDCS.2012.69

[45] Jin Y Yen. 1971. Finding the k shortest loopless paths in a network. management Science 17, 11 (1971), 712–716.

[46] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). USENIX Association, San Jose, CA, 15–28. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia

[47] Shawn Zandi. 2016. Project Altair. https://engineering.linkedin.com/blog/2016/03/project-altair--the-evolution-of-linkedins-data-center-network

[48] Mingyang Zhang, Radhika Niranjan Mysore, Sucha Supittayapornpong, and Ramesh Govindan. 2019. Understanding Lifecycle Management Complexity of Datacenter Topologies. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). USENIX Association, Boston, MA, 235–254. https://www.usenix.org/conference/nsdi19/presentation/zhang

[49] Hua Zhong, PingPing Dong, WenSheng Tang, Bo Yang, and JingYun Xie. 2020. A Short Flows Fast Transmission Algorithm Based on MPTCP Congestion Control. In Artificial Intelligence and Security, Xingming Sun, Jinwei Wang, and Elisa Bertino (Eds.). Springer International Publishing, Cham, 786–797.

[50] Noa Zilberman, Gabi Bracha, and Golan Schzukin. 2019. Stardust: Divide and Conquer in the Data Center Network. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). USENIX Association, Boston, MA, 141–160. https://www.usenix.org/conference/nsdi19/presentation/zilberman

A Full evaluation result for DC flow traces

Figures 16 - 20 show the CDF distribution of flow completion times (FCTs) of all five traces discussed in section 5.3, the flow size distribution of which is plotted in Figure 13a. We compared the result at both 10/40G (top rows) and 100/400G (bottom rows), as well as both fat tree topologies (left columns) and Jellyfish topologies (right columns).

As we can see from these results, at 10/40G, P-Net can achieve lower latency on most flows, as it can provide better load balancing and accommodate multiple flows per host than serial networks, thus achieving close to ideal high-throughput network's performance. In addition, the path length advantage in parallel heterogeneous networks can sometimes provide further improvements over the parallel homogeneous ones. At 100/400G, the path length advantage of heterogeneous 4 × 100G P-Net shows its full strength, allowing certain short flows to achieve even lower latency than an ideal 400G serial network.

B Artifact Appendix

B.1 Abstract

The artifact for this paper consists of a C++ packet simulator on top of `htsim` from the NDP paper [23], a set of scripts to invoke the simulator and/or LP solver Gurobi [29], and finally the plotting scripts in the form of Jupyter notebooks to reproduce the figures in the evaluation section.

The results directly support the analysis of P-Net in the evaluation section (section 5), and we expect nearly identical results when reproducing the data points in most graphs. We labeled the plotting scripts so that each one corresponds to one sub-section of the evaluation section.

In terms of minimal requirements, there's no real hardware requirements, except that a high clock rate and multi-core CPU can help reproduce the result faster, simply because discrete time packet simulators like *htsim* are inherently unparallelizable (for the most part), and similarly LP solver at larger network sizes can be exponentially slower, which can take up to days for the largest experiments.

For software requirements:

- Platform/OS: Ubuntu 18.04 (or other Linux distro) + a few standard GNU/Linux tools (e.g. `parallel`, `numactl`, ...)
- C++ simulator: `cmake` 3.10+, `gcc-9/g++-9`, OpenMP (`libomp-dev`), Boost library (`libboost-all-dev`), `htsim` from NDP [23] (included)
- LP solver: Gurobi [29], free academic license available
- Plotting script: Jupyter notebook, numpy and matplotlib.

B.2 Artifact check-list (meta-information)

- **Program:** `gurobi`, C++ packet simulator on top of `htsim`, Jupyter notebook and matplotlib for plotting
- **Compilation:** gcc-9/g++-9 with OpenMP and Boost library.
- **Run-time environment:** Ubuntu 18/20

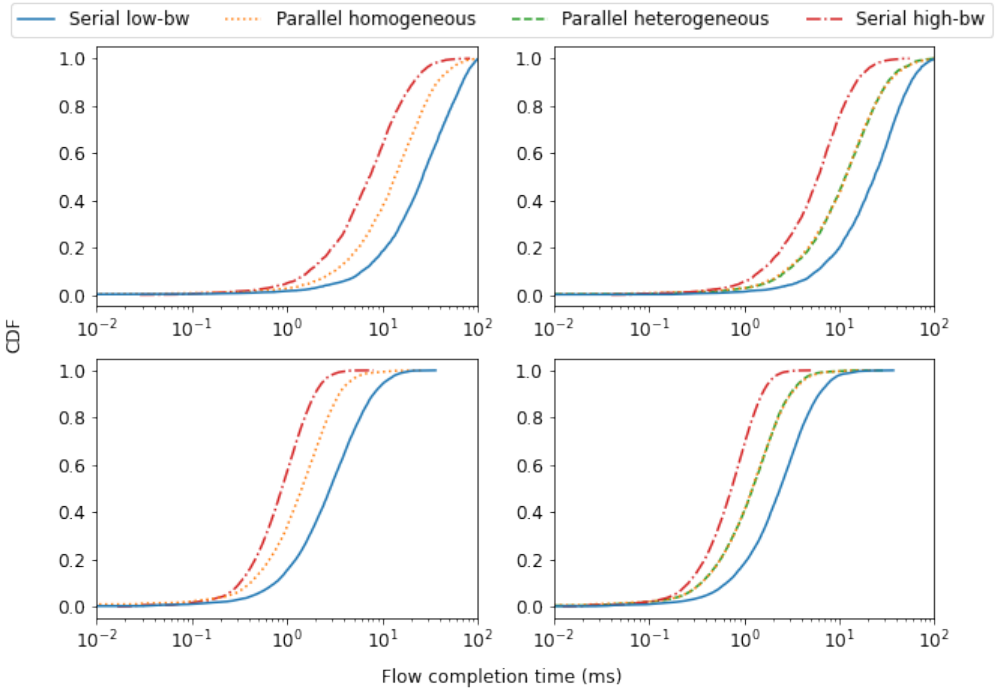


Figure 16. Flow completion time distribution based on Websearch traces from [6]. Top row plots are for 10/40G and bottom row plots are for 100/400G. Left column plots are for fat trees and right column plots are for Jellyfish.

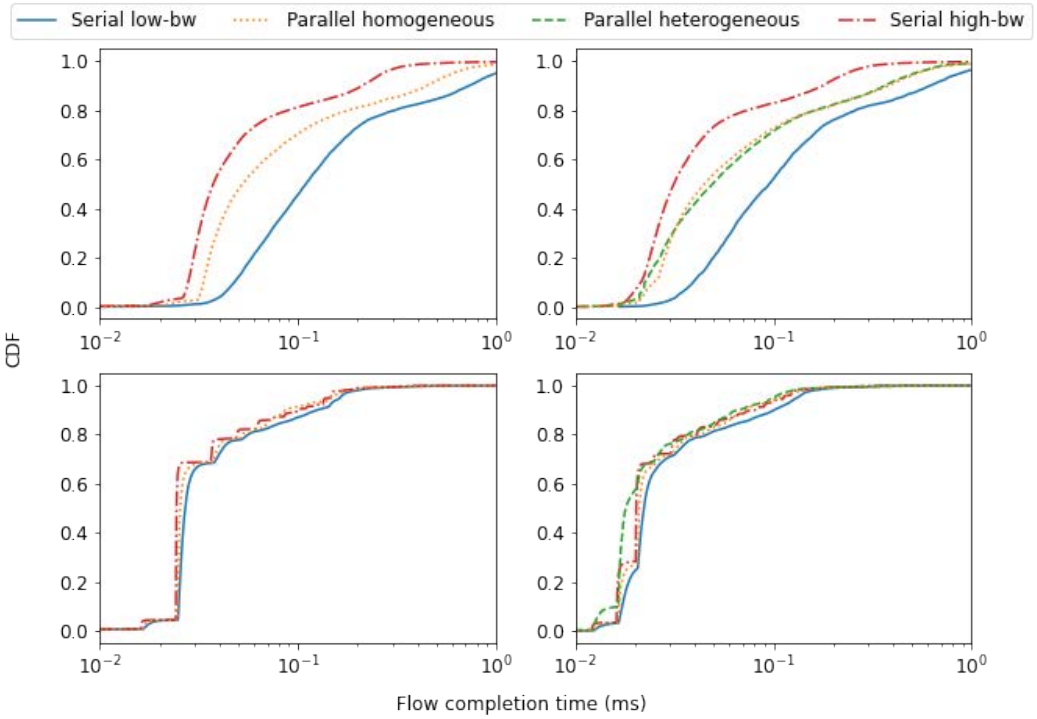


Figure 17. Flow completion time distribution based on webserver traces from [35]. Top row plots are for 10/40G and bottom row plots are for 100/400G. Left column plots are for fat trees and right column plots are for Jellyfish.

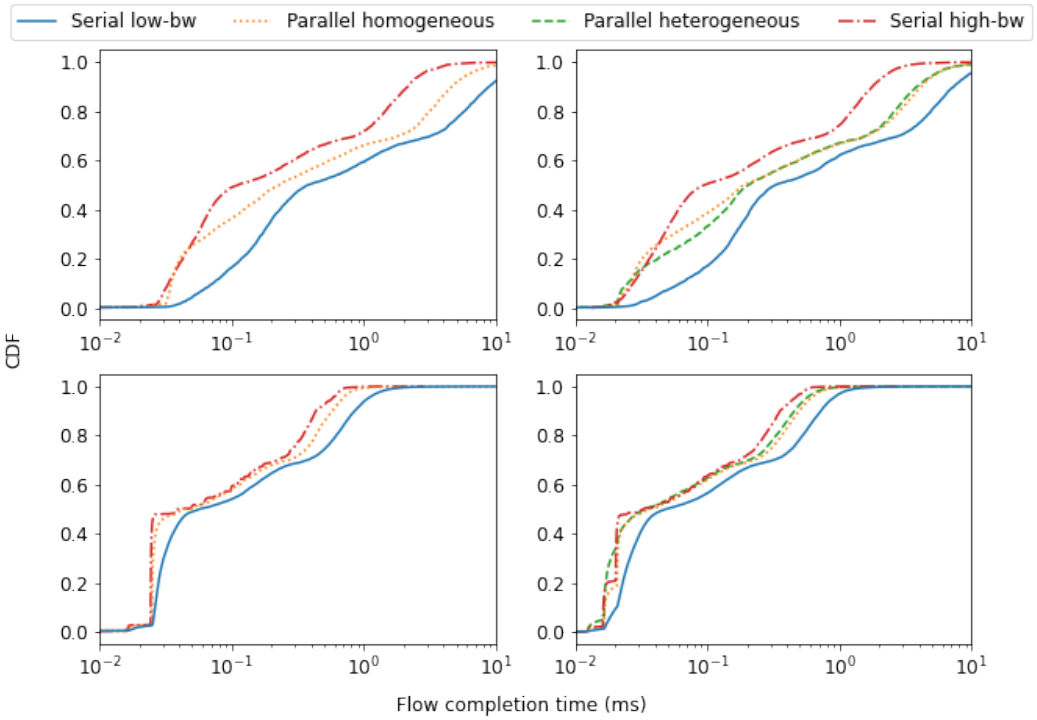


Figure 18. Flow completion time distribution based on cache traces from [35]. Top row plots are for 10/40G and bottom row plots are for 100/400G. Left column plots are for fat trees and right column plots are for Jellyfish.

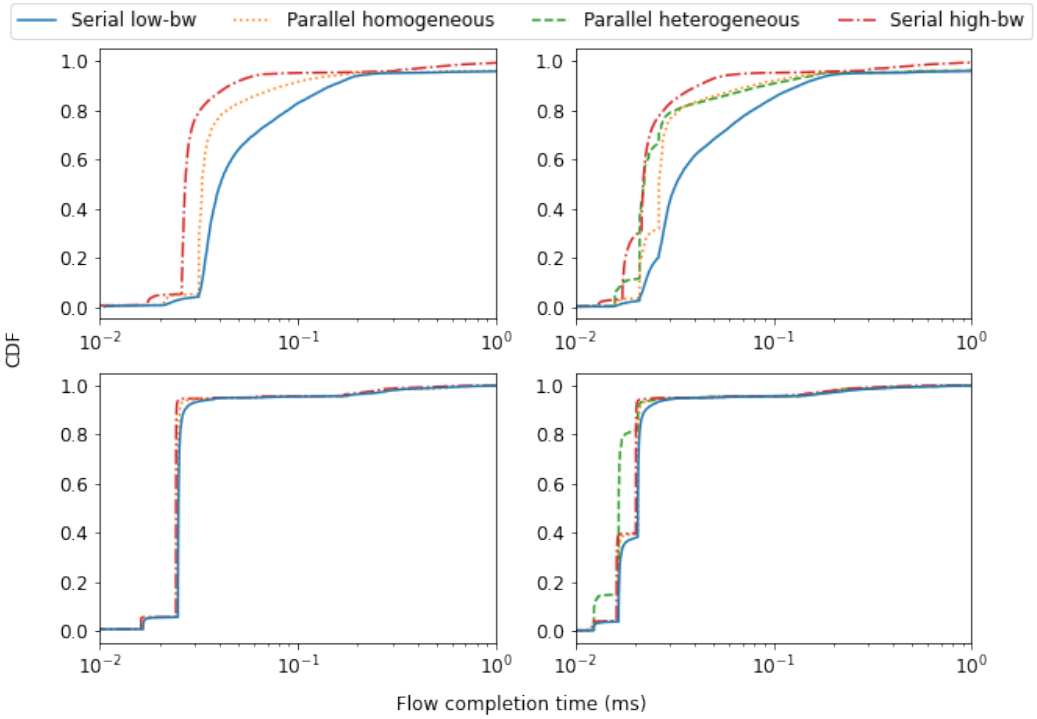


Figure 19. Flow completion time distribution based on Hadoop traces from [35]. Top row plots are for 10/40G and bottom row plots are for 100/400G. Left column plots are for fat trees and right column plots are for Jellyfish.

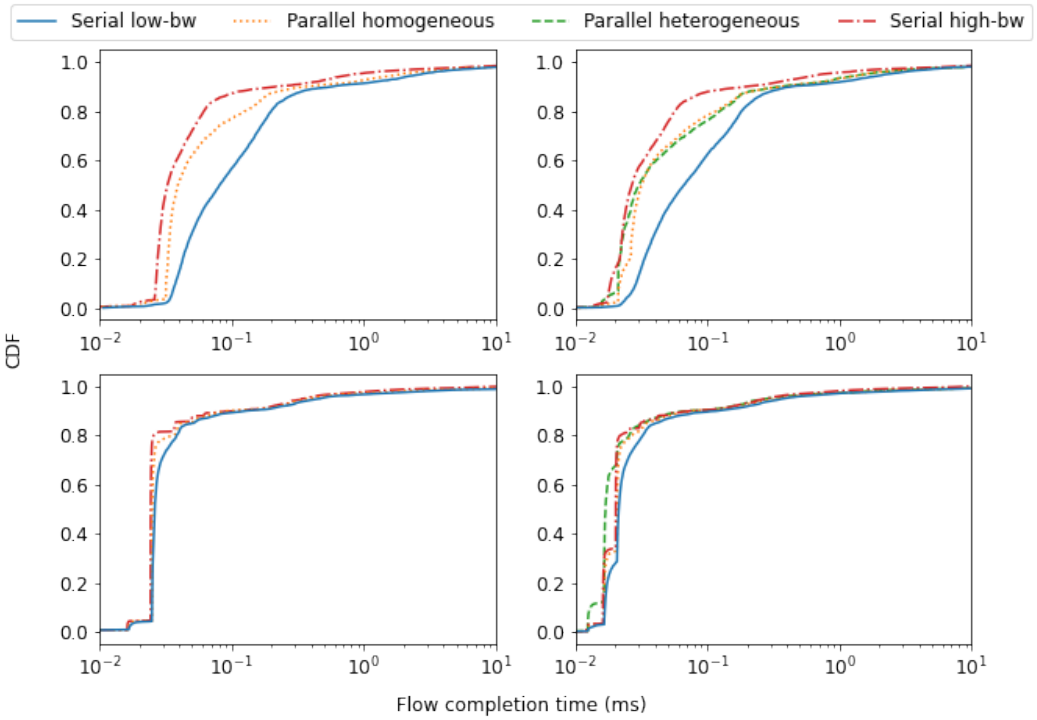


Figure 20. Flow completion time distribution based on Datamining traces from [22]. Top row plots are for 10/40G and bottom row plots are for 100/400G. Left column plots are for fat trees and right column plots are for Jellyfish.

- **Hardware:** High clock rate and large core count CPU and decent RAM for faster/parallel runs. Only Gurobi has a real constraint on RAM size, which might be up to tens of GBs for the largest size (1024 hosts).
- **Execution:** Normal C++ program and Liunx command line tools. Can potentially run multiple instances in parallel to speed up the overall process.
- **Metrics:** flow throughput (from LP solver), flow completion time and hop count (from packet simulator).
- **Output:** CSV/text/binary files from LP solver / simulator, then parsed by plotting script to generate figures.
- **Experiments:** Corresponds to subsections of the evaluation section: e.g. ideal throughput from LP solver, and flow completion times and hop count distribution of various traffic types from packet simulator.
- **How much disk space required (approximately)?:** up to tens of GB disk space for intermediate files from `htsim` in one or two large scale experiments; otherwise up to a few GBs per experiment. The more packets we send, the more data is in the `htsim` intermediate files. After each run, we can use the parser to calculate the metric we care about and discard these files.
- **How much time is needed to prepare workflow (approximately)?:** 1-2 hours.
- **How much time is needed to complete experiments (approximately)?:** the longest experiment can take up to hours to complete (e.g. gurobi all-to-all traffic on 1024 hosts, or flow completion times of 1GB flows), but most others take minutes to half an hour. Overall process can be sped up by running multiple experiments in parallel, or with reduced topology sizes and/or flow sizes.
- **Publicly available?:** yes, on both Zenodo and Github.
- **Code licenses (if publicly available)?:** MIT
- **Data licenses (if publicly available)?:** n/a
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.7196266

B.3 Description

B.3.1 How to access. You can access the source code through either:

- Github: https://github.com/nilyibo/conext22-parallel-networks, or
- Zenodo: https://doi.org/10.5281/zenodo.7196266

B.3.2 Hardware dependencies. No hard constraint, but higher CPU core frequency and higher core count can speed up the experiments considerably. Larger amount of RAM can also enable Gurobi to run at a larger size, e.g. the 1024-host setup in this paper.

B.3.3 Software dependencies.

- Platform/OS: we ran our experiments on Ubuntu 18.04, but any Linux platform should work. We also used a few standard GNU/Linux tools for orchestration and data pre-processing, including but not limited to: `parallel`, `numactl`, ...
- C++ simulator: we compiled the simulator with `cmake` 3.10+, `gcc-9/g++-9`, OpenMP (`libomp-dev`), Boost library (`libboost-all-dev`) and `htsim` repo cloned from NDP [23] (included).
- LP solver (Gurobi [29]): we downloaded from their official website and requested academic license.
- Plotting script: we used Jupyter notebook, numpy and matplotlib.

B.3.4 Data sets. We used both synthetic traffic generated using our script (section 5.1 - 5.2) and flow size distributions from published data center traces (section 5.3). For the latter, we captured the CDF curves from figures in these papers and saved them as CSV files, which is also included in the archive.

B.4 Installation

We provide setup scripts to install the dependencies required to build the simulators, as well as the build script. Details are in the README in the archive.

B.5 Experiment workflow

Depending on the type of experiment (e.g. throughput from LP solver in section 5.1.1, flow completion times from packet simulator in section 5.1.2 through section 5.3, and hop count from the same simulator in flow-path-only mode in section 5.4), we invoke different programs (LP solver or packet simulator) with different parameters.

This will produce some binary/text/CSV files, which is then pre-processed by htsim parser (in binary case) or standard GNU/Linux tools to a concise format, and subsequently used by plotting script to generate the figures in the evaluation section.

We provide the details of each experiment's workflow in the README.

B.6 Evaluation and expected results

To help reproduce the evaluation, we prepared one run script and one plotting script per subsection or type of evaluation. The run script will invoke either the LP solver, or the packet simulator to produce the output and run necessary pre-processing to produce simple text/CSV files. The plotting script can then read from these text/CSV files and produce the figures in this paper.

We expect the figures to have similar trends/values as shown in the paper, except Figure 9, where we observed some variation due to non-steady state in smaller sized flows (< 100MB).