Simple and Cheap Theia: ˇ Networking for Ultra-Dense Data Centers

meg walraed-sullivan
Microsoft Research
Redmond, WA, USA
megwal@microsoft.com

Jitendra Padhye
Microsoft Research
Redmond, WA, USA
padhye@microsoft.com

David A. Maltz
Microsoft
Redmond, WA, USA
dmaltz@microsoft.com

ABSTRACT

Recent trends to pack data centers with more CPUs per rack have led to a scenario in which each individual rack may contain hundreds, or even thousands, of compute nodes using system-on-chip (SoC) architectures. At this increased scale, traditional rack-level star topologies with a top-of-rack (ToR) switch as the hub and servers as the leaves are no longer feasible in terms of monetary cost, physical space, and oversubscription. We propose Theia, an architecture to connect hundreds of SoC nodes within a rack, using inexpensive, low-latency, hardware elements to group the rack's servers into subsets which we term SubRacks. We then replace the traditional per-rack ToR with a low-latency, passive, circuit-style patch panel that interconnects these SubRacks. We explore alternatives for the rack-level topology implemented by this patch panel, and we consider approaches for interconnecting racks within a data center. Finally, we investigate options for routing over these new topologies. Our proposal of Theia is unique in that it offers the flexibility of a packet-switched networking over a fixed circuit topology.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network Commuications, Network Topology*; C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Design, Measurement, Performance

Keywords

Data center networks; Network topologies

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
Hotnets '14, October 27–28, 2014, Los Angeles, CA, USA.
Copyright 2014 ACM 978-1-4503-3256-9/14/10 ...$15.00
http://dx.doi.org/10.1145/2670518.2673862.

1. THE PROBLEM

Building new data centers is expensive, so major cloud service providers are trying to pack existing data centers with more hardware[13]. One way to do this is to pack more CPUs in a standard rack. We are experimenting with new architectures in which a single rack can contain several hundred or even a few thousand compute nodes that use system-on-chip (SoC) [1, 16] architectures. Informally, such data centers are called *Ultra Dense* data centers, or UDDCs.

The extreme scale of the UDDC architecture leads to several challenges in the areas of system management, power and cooling provisioning, failure recovery, tailoring applications for SoCs, and of course, networking. Since this is HotNets, we will focus only on one key networking problem, the fact that standard ToR-based network architectures do not work for UDDCs.

To understand why this is so, consider how data center networks (DCNs) are architected today. In a typical DCN architecture [3, 7, 9], servers within a rack are connected in a star topology, with each server connected directly to a top-of-rack (ToR) switch, and the ToR switches form the leaves in a Clos-style topology. These architectures rely on high-speed ToR switches with high port counts. Ports are needed both to connect to the large number of servers in the rack, and also to connect the rack to the rest of the data center at an acceptable oversubscription ratio [9].

This ToR-based model of networking does not scale for UDDCs. A densely-packed UDDC rack would require a ToR with hundreds or even thousands of ports. Unfortunately, the technology to build high-radix ToRs becomes less feasible and more costly as link speeds increase. Building a high speed ToR with even a few hundred ports is prohibitively expensive. In fact, at 100G it may be infeasible to scale past 32 ports.

To address this problem, we propose a networking architecture called Theia to support UDDCs. Theia divides racks into smaller components called SubRacks. We connect servers within a SubRack using specialized hardware to avoid introducing latency within the rack as well as to reduce cost. We then interconnect SubRacks within a rack using a passive optical patch panel with a fixed topology. The patch

panel introduces no latency and requires no power. Racks within the data center are then connected to one another using "leftover" ports on SubRack switches and patch panels.

In this paper, we will describe the Theia architecture in more detail. We stress two things. First, this is a *preliminary* design. We are currently building a UDDC prototype to evaluate vendor hardware, and the Theia architecture will continue to evolve as we progress. Second, we aim for a design that is simple, practical and cheap. To this end, we proudly, and extensively beg, borrow and steal from prior work on data center networking. Our primary aim in writing this paper is to highlight the problem, and start a conversation on new networking architectures for UDDCs.

2. PRINCIPLES AND ASSUMPTIONS

We begin our redesign of the data center network with three key observations about SoC-based architectures:

Oversubscription is unavoidable. At the enormous scale of this new architecture, some amount of oversubscription is necessary. Therefore, rather than (fruitlessly) trying to avoid oversubscription, we should instead take care to introduce it in the appropriate places and amounts.

Passive components are necessary. Increasing the number of servers in the data center by an order of magnitude (or more) leads to the need for an order of magnitude more network ports (and likely network elements). In order to keep cost and power requirements manageable, some of these components will need to be passive.

Inter-rack communication is likely to be less frequent. Applications have the opportunity to leverage the enormous rack-locality enabled by such densely-packed racks. As such, it is crucial that we design the network within the rack for performance, at the cost of a narrower pipe between the rack and the remainder of the data center.

Why current architectures do not scale: The single ToR-per-rack model of networking becomes infeasible when the rack contains hundreds or thousands of SoCs. Such a ToR would need to have twice as many ports as there are SoCs in the rack to not be over subscribed, in order to account for uplinks to the remainder of the data center. Reducing the ToR's port count would quickly lead to a non trivial amount of oversubscription, and may not decrease the size of the ToR sufficiently; in fact, even a ToR with 1,000 10G downlinks to SoCs and only a single 10G uplink (therefore oversubscribed with a ratio of 1:1000) would be prohibitively expensive to build and would consume a significant portion of the physical space within the rack. At higher link speeds, these gaps only increase[1]. Therefore, for densely-packed racks of SoCs, it is clear that some sort of in-rack aggregation is necessary.

A naïve solution: A naïve approach to redesigning the network is to simply add more ToRs within the rack, as shown in Figure 1. The effect of this is to logically push the existing ToRs up a level in the hierarchy, creating a new hierarchy with an additional level. However, adding multiple standard ToR switches to the rack reduces the number of servers the rack can hold (due to space and power limitations), negating the rationale behind UDDCs. The additional ToRs would also significantly increase cost and management complexity. Since the naïve approach is not feasible, we design Theia, guided by the principles described above.

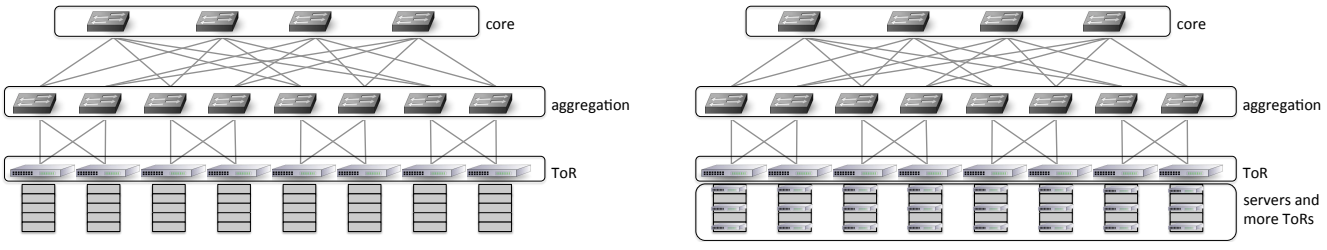
3. THE THEIA ARCHITECTURE

To add an additional level of aggregation to our data center, we divide the rack into several constituent components, each containing a subset of the rack's servers, and we coin these components SubRacks. A clean way to divide a rack into SubRacks is to group SoCs within a single rack-unit into a SubRack, leading to SubRacks of tens of SoCs. At this point, our task is to determine how to interconnect servers within a SubRack, SubRacks within a rack, and finally racks within a data center. We address each of these in turn, keeping the principles introduced above in mind.

3.1 Connecting Servers into SubRacks

As described earlier, using traditional ToR switches to connect servers within a SubRack is not practical due to space limitations and cost. Rather than try to devise a clever algorithmic solution to this problem, we simply use better hardware. Specifically, instead of ToR switches, we use an

[1]In case you are wondering whether these "tiny" SoCs need and/or can fill 40 or 100G pipes, the answer is yes—by using NICs that support hardware-based protocols like Infiniband.



(a) Traditional Architecture

(b) Deeper Architecture

Figure 1: Naïve Approach: Add Another Level of Hierarchy

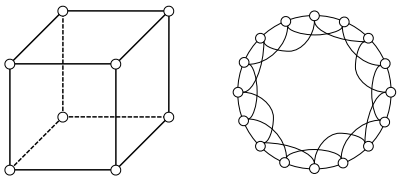


Figure 2: Sample Circuit Topologies

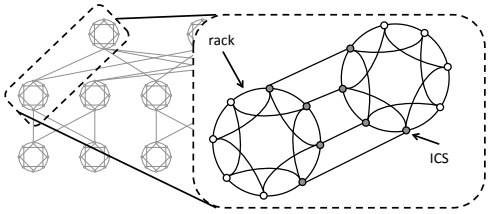


Figure 3: Tree of Circulant Graphs

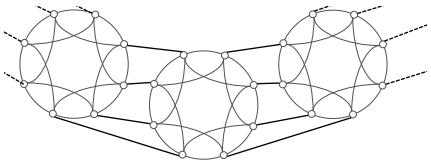


Figure 4: Connecting to Multiple Neighbors

in-chassis hardware switch-card (ICS) that interconnects all of the SoCs within a single chassis (SubRack). ICSs connect SoCs with copper rather than cables, sit within the same chassis as the SoCs themselves, and can be deployed with a variable number of downlinks to the SoCs and uplinks to the remainder of the rack. We design Theia to have on the order of ten or fewer uplinks per ICS (and the ICS's corresponding tens of servers) which yields a SubRack oversubscription ratio similar to that at the ToR level in today's data centers.

Note that at the ICS level, there is a fundamental tradeoff between oversubscription and aggregation regardless of the hardware used. If we reduce the oversubscription ratio at the ICS level, we would simply push the problem of connecting thousands of ports one level up in the hierarchy.

3.2 Connecting SubRacks into Racks

Our design of around ten uplinks per SubRack leaves us with the task of interconnecting a few hundred ports per rack. A traditional ToR is again impractical here, due to the prohibitive cost, the several rack-units of physical space that would be required, and the significant oversubscription that would be necessary to make the port count manageable.

Thus, we simply connect SubRack ICSs in a fixed topology using a passive optical patch panel. Since the patch panel does not contain any active components, it is compact, and draws no power. It is passive, and so does not add any queuing delay. Finally, the patch panel also makes cabling easier. Cabling in this case is reduced to simply a matter of connecting each port on each ICS to one input port on the patch panel, and the complexity of the topology implemented by the patch panel is hidden from the operator.

In other words, this level of aggregation in Theia comes at nearly no cost. The downside, of course, is in the loss of flexibility; instead of a switched topology we use a fixed-circuit topology. We discuss the implications of this in § 4. We chose not to use optical switches like those used in Helios[2] [8] and Mordia [18] to save cost, power and space.

The topology implemented by the patch panel to connect SubRacks is an open problem. We have considered several different options, including hypercubes [2, 6] and circulant graphs [10], as shown in Figure 2. We use the circulant graph in our initial Theia design for reasons that we discuss in § 4.

Note that this type of design may lead to scenarios in which for at least some pairs of SubRacks S_A and S_B, traffic cannot pass directly from servers in S_A to S_B, and instead must pass through SubRack S_C while en route. It is crucial, therefore, that the design of the circuit minimizes both the number of occurrences and lengths of these indirect paths.

[2]We note that Theia is a mother of Helios ...

3.3 Connecting Racks Across the Data Center

Finally, we consider the interconnection of racks within the data center. As UDDC racks have significantly higher server count than those in a traditional data center, we envision that more applications will be able to keep their computations and storage accesses rack-local and thus we intentionally design a narrower pipe from one rack to another.

We form the interconnect between racks by using parallel connections from multiple points within each rack. That is, we devote a subset of the links that would otherwise be used to wire the internals of the patch panel instead to connect an ICS in one rack to a corresponding ICS in a neighboring rack.[3] In this way, we form large, data center-level topologies in which the constituent nodes are racks of SubRacks, and "links" between these nodes are groups of links from sets of ICSs within a rack.

Figure 3 provides a visualization of this concept. The figure zooms in on a portion of a Clos-style topology, in which individual nodes in the tree are circulant graphs of ICSs. The links between these circulant graph nodes are formed by groups of links between corresponding pairs of (shaded) ICSs within neighboring racks. Therefore, each ICS is a single node within a circulant graph, and each rack of eight ICSs is a single node in the larger Clos topology. In general, depending on the type of topology selected for the data center, it will likely be the case that each rack has multiple neighboring racks and varying subsets of ICSs may be used to connect the rack to each of its neighbors, as in Figure 4.

The implications of this design are non trivial; not only do we have a tradeoff between connectivity within the rack and oversubscription to the rest of the data center, but we also have an architecture in which traffic destined from a server in rack R_A to a server in rack R_B may pass through rack R_C's patch panel (and ICSs) in the process. That is, ICSs carry not only traffic for their own racks, but also "through-traffic" as it passes from one rack to another. This is analogous to the

[3]Note that this connection still happens via the patch panel for cabling simplicity. We simply connect certain ports of a rack's patch panel to other rack's patch panels rather than to local ICSs.

discussion of through-traffic between SubRacks in § 3.2; we examine this property more closely in § 4.

Note that we have uncovered the tradeoff alluded to by our third principle (§ 1), in terms of the number of links used for intra-rack connectivity versus those used for inter-rack connectivity. We can make this trade based on expected workloads. For instance, for a highly rack-local workload, we devote most of the links from the rack's ICSs to the rack's patch panel; traffic leaving the rack is oversubscribed. On the other hand, for an application with more all-to-all requirements we use more links to connect each rack to other racks, limiting oversubscription at the cost of a reduction in intra-rack bandwidth. This tradeoff corresponds to a change in the number of straight-line links in Figures 3 and 4.

3.4 Architecture Summary

Thus we propose a new architecture for UDDCs in which densely packed racks of hundreds or even thousands of SoCs are aggregated into SubRacks within a rack, each with a high-speed ICS to provide connectivity to the rest of the rack. These SubRacks are connected to one another by a passive circuit component, or patch panel. We select this circuit-based design to enable a high-performance interconnect with little cabling complexity and no power requirement. We then repurpose a subset of "leftover" links within the patch panel to connect racks to one another, building e.g. a Clos-style network in which each node is a full rack of SoCs.

4. PATCH PANEL CIRCUIT DESIGN

We now consider intra-rack connectivity, that is, the topology implemented by the patch panel. Our ideal topology:

- Minimizes the amount of traffic that passes through ICSs in SubRack S_B while in transit from SubRack S_a to SubRack S_C,
- Supports a wide range of graph sizes (i.e. numbers of ICSs) as well as node degrees (ICS port counts) and does not have a dependency between the number of nodes in the graph and the degree of each node, as with topologies such as hypercubes [2, 6, 11][4], and,
- Reduces the disruption caused by failures; since ICSs are responsible for a large number of SoCs, and since they may also carry through-traffic, their failures can have significant impact. Additionally, due to the large number of ICSs across the data center, there are simply more components to fail and so failures will be more frequent.

We have considered a number of alternatives for the patch panel topology. For Theia, we chose not to focus on topologies such as DCell [12], BCube [11], and hypercubes and toruses [2, 6], since these topologies work best with particular, well-constrained numbers of nodes and often have a dependency between the radix of each node and the total number of nodes in the graph. Additionally, topologies like DCell [12], BCube [11] include two types of nodes, servers and switches, and servers play a switching role in the topology. In contrast, we will only have a single type of node, the ICS, and servers will not switch packets in our design.

We also considered Jellyfish [19], because it offers a low average hop count between pairs of nodes. This can help to minimize the through-traffic that ICSs carry on behalf of other SubRacks. Another beneficial feature of Jellyfish topologies is the ability to grow organically. That is, the addition of a new node to the topology does not require rewiring of existing nodes. However, while this feature is quite valuable on a large scale, such as that of an entire data center, it is less crucial at the rack-level, especially when the topologies implemented by our patch panels are static. Furthermore, our network operators were unwilling to work with random rack-level topologies, as it made it difficult for them to maintain a human-readable map of the network.

Given the three requirements above, we focus our initial exploration on the circulant graph. Such graphs have desirable latency and failure resilience properties and retain these properties and their regular structure across a wide range of graph sizes and port counts. Another benefit of the circulant graph is that it is isomorphic with respect to many node and/or link swaps. This means that in many cases, a miscabling at the patch panel level will lead to a graph that is isomorphic to the intended topology. Miswirings are a common and expensive problem in today's data centers and reducing their impact can help significantly.

A circulant graph is defined by a number of nodes, $N = \{n_0,...,n_{N-1}\}$ and a set of strides $S = \{...,s,...\}$, such that for each $s \in S$, n has links to nodes $n \pm s$ (with all arithmetic modulo N). Note that $|S|$ is upper-bounded by half of the number of ports per node. Thus the graph on the right side of Figure 2 is a circulant graph with N=16 and S=$\{1,2\}$. As we show below, circulant graphs are able to minimize the amount of through-traffic quite well, even across failures, and we can build performant and failure-resilient circulant graphs for varying values of N and $|S|$. Furthermore, the selections of N and $|S|$ are independent of one another.

Once the values for N and $|S|$ have been decided for a circulant graph, it is simply a matter of selecting the appropriate set of strides to optimize for desired properties. There is a large amount of literature devoted to the computation of optimal stride sets in circulant graphs [4, 5, 15, 21, 22], each of which leverages a particular heuristic to make computations over large circulant graphs more performant at the cost of some accuracy. For our relatively small graphs of only tens of nodes and a few hundred links, we are able to simply perform brute-force calculations of optimal stride sets.

We explore circulant graphs with respect to a number of different properties, including:

Average Path Lengths This corresponds to both the latency encountered by packets that traverse the circulant graph as well as a measure of the amount of through-traffic

[4]This is important since we wish to tune the number of ICS ports used to connect SubRacks to one another and those used to connect to other racks, based on expected workloads.

in the graph; if the average path length between pairs of nodes in the graph is close to 1, this means that few pairs of nodes send traffic through intermediate nodes.

Prevalence of Long Paths We also consider counts of pairs of nodes with short and long paths. For instance, if the average path length across the graph is 3, it may be preferable for this to result from most node-pairs having 1-hop paths and a few having say, 6-hop paths, rather than from all node-pairs having 3-hop paths.

Failure Resilience We study the failure resilience of circulant graphs along multiple metrics. First, we study how many nodes (ICSs) can fail before a graph is disconnected, since partitioning the network within a rack would lead to poor application performance, especially for applications that expect to leverage UDDCs' enormous rack-locality. We also consider the effects of failures on the path-length properties above. Longer path lengths increase the chance of "interference" between flows and can lead to poor performance.

We built a simulator to measure these properties. Our simulator generates circulant graphs (given values for N and S) and calculates the forwarding tables that would be used by standard shortest path algorithms.[5] Note that if the stride set elements of a circulant graph are not coprime as a group, the circulant graph is not connected, even in the absence of failures. Our simulator filters all such pathological graphs.

4.1 Average Path Lengths

To measure circulant graphs' performance in terms of path lengths, we used our simulator to generate a number of different graphs based on the parameter ranges we expect to see in practice. We varied the number of nodes (i.e. ICSs) in each graph from 16 to 32 in steps of 2. (This is along the lines of the capacity of the Open Compute Rack [17].) We varied the port count on each node from 2 to 10, again in steps of 2; this corresponds to the expectations for ICS sizes that we introduced in § 3. Figure 5 shows the best average path length for each graph, across all possible stride sets for a particular port count.

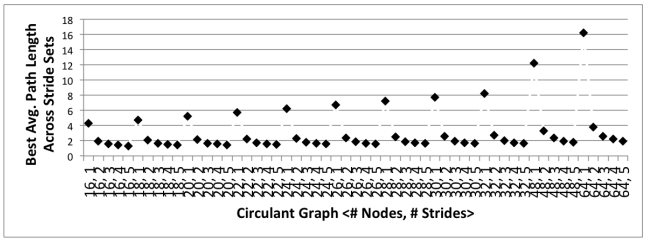


Figure 5: Average Path Lengths, Optimized Across All S, for Circulant Graphs with N=16-32 and $|S|$=1-5

As the figure shows, for a sufficient number of strides, $|S|$, the average path length between pairs of nodes stays fairly close to 1, meaning that the amount of through-traffic carried by ICSs is quite low. As expected, when the graphs grow larger it is more difficult to maintain low path lengths with fewer strides. An interesting aspect to note is that occasionally adjacent entries in the plot are nearly identical. That is, there are graph sizes for which the number of strides does not need to be higher than a particular threshold in order to achieve a low average path length. For instance, the graph with 32 nodes achieves nearly the same average path length with 4 strides as it does with 5 strides, allowing a network operator to free up 2 ports per ICS for other use.

4.2 Prevalence of Long Paths

We next consider the counts of node-pairs with distinct path lengths. Since the smallest path length between a pair of nodes is, by definition, a single hop, we expect that if the average path length for a graph is close to 1, it is likely the case that most pairs of nodes have a 1 or 2-hop path between them.[6] However, it is the outliers, that pull this average away from 1, that we are concerned with. We would like to know whether there are a few outliers with fairly long paths, many with medium-length paths, and so on.

To measure this, we generated graphs with 16 6-port nodes ($|S|$=3), with 24 8-port nodes ($|S|$=4), and with 32 10-port nodes ($|S|$=5). We chose a representative sampling of good, mediocre, and poor stride sets, and show for each the breakdown of node pairs with single-hop path lengths, those with 2-hop paths, and so on. In Figure 6, each x-axis value corresponds to a unique graph (N and S) and each y-axis bar shows the percentage and counts of node pairs for each path length.

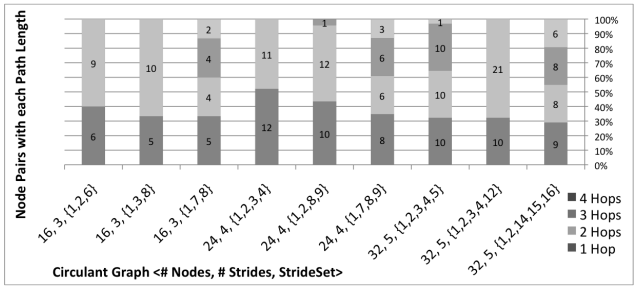


Figure 6: Varying Stride Choices for Graphs with (N=16, $|S|$=3),(N=24, $|S|$=4),(N=32, $|S|$=5)

The figure shows the wide variation in stride set choices for a circulant graph of a given size. For instance, consider the options shown for the 16-node graph with $|S|$=3. With a stride set of {1,2,6}, all pairs of nodes that cannot have 1-hop paths (due to the limiting factor of $|S|$) have 2-hop paths, whereas with a stride set of {1,7,8}, several nodes have 3- and 4-hop paths. Thus the former graph has better performance for less cost in terms of port count.

[5]For these computations, we assume the presence of shortest-path routing, with either tie-breaking or ECMP-style [14] multi-path routing in the case of multiple options.

[6]The number of pairs of nodes with 1-hop paths is limited by $|S|$; a node has 1-hop paths to at most $2|S|$ other nodes.

4.3 Failure Resilience

Next, we examine the resilience of circulant graphs to failures. For each of the graphs shown in Figure 5, we test all possible single link failures, all combinations of 2-link failures, of 3-link failures, and so on until the graph becomes disconnected. For each count of f failed links, we also record the average path lengths across pairs of nodes, and across all sets of f failures. Note that the upper bound for disconnection is trivially the degree of each node, $2|S|$, because we can disconnect a node from the graph by simply failing all of its links.

It turns out that circulant graphs are quite resilient to failures. For all of our simulations, the graph remained connected until it reached $f = 2|S|$, at which point it was trivial to find a set of f links all incident upon the same node. When we added lines to Figure 5 to show the average path lengths for each possible value of f, we found that these path lengths were barely affected by failures, even up to just before the point of disconnection. In fact, the lines were so close to the zero-failures case of Figure 5 that adding them made the figure illegible. This highlights the benefits of the circulant graph's multipath characteristics; there are many short, link-disjoint paths between each pair of nodes and even the introduction of numerous simultaneous link failures does not significantly affect the graph's performance characteristics.

5. ROUTING

In addition to designing Theia's architecture, we need to build the appropriate routing primitives and protocols to run over top of the hardware. As we build out our testbed, we expect to revise and refine these proposals to tackle practical challenges as they emerge. Here, we present our initial ideas.

5.1 Routing Within the Patch Panel

We are in the process of building a routing algorithm that operates over circulant graphs to efficiently use the short paths that they enable. Intuitively, our algorithm works as follows: At each hop along the path, a packet takes the longest possible next link without passing its destination. In this case, "longest" does not refer to physical length, but rather to the difference between the node numbers of the nodes at either end of the link. Of course, it is possible to construct circulant graphs for which the longest next hop is not always preferable.

While our current algorithm is highly tuned to the particular circulant graphs with which we are experimenting, it may also be possible to borrow from existing approaches such as Chord [20]. Fortunately, since our relatively small circulant graphs contain only tens of nodes and hundreds of links, we can compute such paths quickly and easily for any given graph. In fact, these calculations (and subsequent re-computations upon failure) can easily be performed on a small ICS CPU, without the help of a centralized controller.

As we showed in § 4, circulant graphs are quite efficient in terms of path length. The patch panel provides a high-speed interconnect on which to implement these circulant graphs; In fact, our initial measurements show that two servers with 10G NICs are able to communicate at roughly 9.9Gbps across the patch panel in the absence of cross traffic.

5.2 Routing Across Racks

Our current proposal for Theia's inter-rack communication is quite simple. It operates similarly to the traditional cross-data center methods today. Across the larger Clos topology, racks exchange connectivity information and collectively calculate shortest paths over a graph in which each rack is a node. A packet to traveling from rack R_1 to rack R_2 may travel through one or more intermediate racks R_i. To accomplish this, it is routed through each R_i's circuit to an ICS with a connection to an appropriate neighboring rack, as shown in Figure 7.

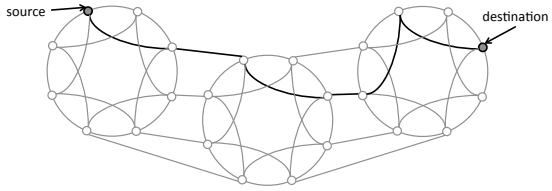


Figure 7: Routing Across Racks

This approach has the drawback of increasing the hop count for inter-rack traffic. However, we hope that the low latency of the patch panel and ICSs as well as the expected rack-locality of our workloads will help to mitigate this added cost. We are investigating other inter-rack topologies as part of our ongoing work.

6. CONCLUSION

In this paper, we made the case that the traditional ToR-based architecture does not scale for UDDCs, due to the high density of compute nodes. We then presented a preliminary design, Theia, that relies on in-rack switch-cards and pre-wired patch panels to address this challenge. Simple simulations show that by using a circulant graph topology in the patch panel, our design can offer good performance and failure resilience across a wide variety of node and port counts.

Needless to say, much work remains to be done. We have investigated only circulant graphs to wire the patch panels; we plan to consider other topologies as well. We also need to investigate inter-rack networking in more detail. We currently build a simple Clos network to connect the racks using leftover ports on the patch panel, but many other possibilities exists, including circuit-switched networks like Helios and Mordia. We need to tackle issues such as addressing schemes and routing protocols. Finally, we also plan to carry out detailed performance evaluations over our testbed.

We stress again that networking is only one of the numerous challenges that a comprehensive UDDC architecture must tackle, and our hope is that this paper inspires further exploration into the design of UDDCs.

7. REFERENCES

[1] Advanced Micro Devices. AMD Embedded G-Series Gamily of Devices. http://www.amd.com/en-us/products/embedded/processors/g-series.

[2] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber. HyperX: Topology, Routing, and Packaging of Efficient Large-scale Networks. ACM SC 2009.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. ACM SIGCOMM 2008.

[4] R. Beivide, C. Martínez, C. Izu, J. Gutierrez, J.-A. Gregorio, and J. Miguel-Alonso. Chordal Topologies for Interconnection Networks. In *High Performance Computing*, volume 2858 of *Lecture Notes in Computer Science*, pages 385–392. Springer Berlin Heidelberg, 2003.

[5] J.-C. Bermond, F. Comellas, and D. F. Hsu. Distributed Loop Computer Networks: A Survey. *J. Parallel Distrib. Comput.*, 24(1):2–10, 1995.

[6] L. N. Bhuyan and D. P. Agrawal. Generalized Hypercube and Hyperbus Structures for a Computer Network. *IEEE Transactions on Computing*, 33(4):323–333, Apr 1984.

[7] Cisco Systems, Inc. Cisco Data Center Infrastructure 2.5 Design Guide. www.cisco.com/univercd/cc/td/doc/solution/, 2008.

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. ACM SIGCOMM 2010.

[9] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. ACM SIGCOMM 2009.

[10] J. L. Gross and J. Yellen. *Graph Theory and Its Applications, Second Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2005.

[11] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: A High Performance, Server-Centric Network Architecture for Modular Data Centers. ACM SIGCOMM 2009.

[12] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. ACM SIGCOMM 2008.

[13] Hewlett-Packard. HP Moonshot System: The World's First Software-Defined Servers, Technical White Paper. http://h10032.www1.hp.com/ctg/Manual/c03728406.pdf, 2013.

[14] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, IETF, 2000.

[15] F. K. Hwang. A Survey on Multi-loop Networks. *Theor. Comput. Sci.*, 299(1-3):107–121, Apr 2003.

[16] Intel Corporation. Intel Plans System On Chip (SoC) Designs. http://www.intel.com/pressroom/kits/soc/.

[17] Open Compute Project. Open Rack. http://www.opencompute.org/projects/open-rack/.

[18] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat. Integrating Microsecond Circuit Switching into the Data Center. *SIGCOMM Comput. Commun. Rev.*, 43(4):447–458, Aug 2013.

[19] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking Data Centers Randomly. USENIX NSDI 2012.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[21] I. Stojmenovic. Multiplicative Circulant Networks Topological Properties and Communication Algorithms. *Discrete Appl. Math.*, 77(3):281–305, Aug 1997.

[22] D. Wang and J. McNair. Circulant-Graph-Based Fault-Tolerant Routing for All-Optical WDM LANs. IEEE GLOBECOM 2010.