GRIN: Utilizing the Empty Half of Full Bisection Networks

Alexandru Agache
University Politehnica of Bucharest

Costin Raiciu
University Politehnica of Bucharest

Abstract

Various full bisection designs have been proposed for datacenter networks. They are provisioned for the worst case in which every server wishes to send flat out and there is no congestion anywhere in the network. However, these topologies are prone to considerable underutilization in the average case encountered in practice. To utilize spare bandwidth we propose GRIN, a simple, cheap and easily deployable solution that simply wires up any free ports datacenter servers may have. GRIN allows each server to use up to a maximum amount of bandwidth dependent on its available network ports and the number of idle uplinks in the same rack. This design can be used to augment almost any existing datacenter network, with little initial effort and no additional maintenance costs.

1 Introduction

FatTree [1] and VL2 [3] are recently proposed datacenter network topologies that are being deployed into production networks and offer full-bisection bandwidth: a high profile example is Amazon's EC2 Infrastructure-as-a-Service cloud that utilizes a 10Gbps FatTree network for its more powerful cluster-compute instances, and a topology resembling VL2 for their other instances[8]. Other major players are using or deploying similar networks.

Full-bisection bandwidth networks are appealing because they allow datacenter operators and application designers to be mostly agnostic of network topology when deciding how to run distributed algorithms or where to place data. In theory, congestion can only appear on the host access links - by design, the network core should never become a bottleneck. For data intensive algorithms (such as the shuffle phase of map-reduce computation), a full-bisection network offers the best possible performance. On the downside, full-bisection networks incur a larger cost than oversubscribed networks.

Datacenters heavily rely on the concept of *resource pooling*: different applications' workloads are multiplexed onto the hardware, and any application can in principle expand to utilize as many resources as it needs as long as there is capacity anywhere in the datacenter. In effect, the resources are pooled in time (when different users access the same machine at different times) and in space (where distributed applications can scale up and down as needed). Also, mechanisms are put in place to ensure their fair use.

Measurement studies [3, 6] show that datacenter networks are underutilized. Many links are running hot for certain periods of time, while even more links are idle, which results in an underutilized core. In this paper we set out to extend the resource pooling principle to datacenter networks. Our goal is simple: when a host wants to send flat-out, it should be able to use as much of the idle capacity of the network as it needs. It should be possible to leverage capacity from everywhere: any flow should be able to fill any part of the network, as long as it wants to do so and the network is underutilized. The effect such network pooling could bring is very appealing: either the network core is fully utilized or there is no application that is bottlenecked by the network. In both cases, the network is providing the best possible performance to the applications.

Achieving this goal is clearly not feasible in today's datacenters where hosts connect using a single gigabit link to the network; this link becomes the bottleneck when hosts want to send flat-out. The question we answer in this paper is *how should datacenter topologies change to achieve resource pooling?*

We seek solutions that are both cheap and deployable in today's production networks, and apply to any full-bisection network, not just FatTree or VL2. Solutions should preserve the same worst-case bandwidth guaran-

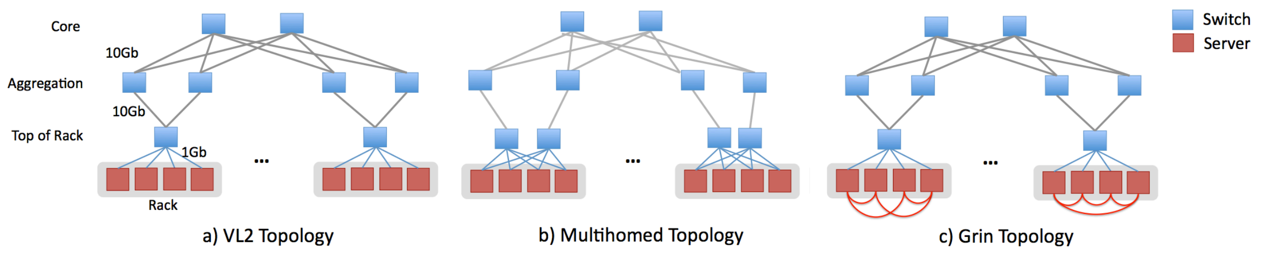


Figure 1: Enhancing a VL2 topology

tee provided by full bisection networks, and should ensure that hosts and applications are properly isolated.

Our proposed solution is very simple: any free port [1] existing in any *server* should be randomly connected to a free port of another server in the same rack. Servers can then communicate using a path provided by the original topology, or via one of the servers they are directly connected to. To be able to utilize multiple paths simultaneously, and to advertise additional paths, the servers use Multipath TCP [7]. Our solution can function seamlessly over existing datacenter networks.

The best case benefits are obvious: for every additional port used, a server can send 1Gbps more traffic if the neighbour's uplink is not utilized at that point in time. Our simulation experiments with synthetic traffic patterns and 2 additional ports show that when 30% hosts are active the throughput increase is between 70% and 170%, depending on the traffic matrix used.

There are also potential caveats: servers must route traffic for other servers and this can impact their performance. Unlike other topologies such as BCube[4], our design ensures that a server never forwards more than 2Gbps of traffic. Our experiments show that this will only utilize at most 10% of a single core at full speed, when dealing with MTU-sized packets.

2 Problem Statement

We want to change existing topologies to allow hosts to utilize idle parts of the network when other hosts are not active. Good solutions share the following properties:

Ability to scale: cost is a major factor that determines what is feasible to deploy in practice. Using more server ports should increase performance and incur little to no additional costs.

Fairness and isolation: access to the shared resource pool must be mediated such that each server gets a fair share of the total bandwidth. Misbehaving servers should be penalized, and they should not adversely affect the performance of the network.

Widely applicable: it should be possible to apply the solution to existing or future networks.

Incrementally deployable: it should be possible to deploy the solutions on live datacenter networks with the least possible disruption. This implies hardware or software changes to the network core (including routing algorithms) are out-of-scope. Further, upgrading only an existing subnet should bring appropriate benefits.

Full-bisection networks offer multiple paths between any pair of servers. In the VL2 network shown in Fig.1.a there are at least four paths between any pair of servers in different racks. Datacenter networks[2] give every host a single IP address and leave to the network the task of mapping traffic onto the available paths. The network runs an intra-domain routing algorithm such as OSPF on the aggregation and core switches (effectively these are routers). The algorithm's output consists of multiple equal-cost routes towards servers. When forwarding traffic, the switches use Equal Cost Multipath (ECMP) to hash each connection onto one of the available paths [3]

Multipath TCP is an evolution of TCP currently being standardized by the IETF: it takes a TCP connection and splits it across multiple paths, while offering applications the illusion they are still working over TCP. MPTCP has already been proposed as a replacement for TCP in datacenters [7]. Its biggest benefit for VL2 and FatTree is avoiding collisions caused by ECMP when multiple connections are placed onto the same congested path, despite the existence of idle capacity elsewhere in the network.

Any solution we devise will have to use multiple NICs at servers. TCP is unable to split a flow across multiple interfaces, so adding more NICs will bring significantly less performance benefits. MPTCP allows a trans-

[1] A quick study shows that between 1 to 3 free ports should be commonly available in servers today, as all major equipment vendors provide either dual-port or quad-port gigabit NICs in their default blade configurations.

[2] at least as seen from Amazon's EC2 cloud

[3] VLANs are another popular solution to expose multiple paths to hosts. Hosts will have as many addresses as possibly paths, and will implement flow placement on paths.

port connection to grow beyond the 1Gbps offered by one NIC port. It also provides by design some of the goals we require - in particular it enforces fairness across collections of links [11].

3 Solutions

3.1 Multihoming

Barring extensive changes to the original topology, the most straightforward solution is to multihome servers by using additional TOR switches. We add a TOR switch for every additional server port (see Fig.1.b), so that each server is connected to each of the multiple TOR switches from its rack. In order to keep the rest of the topology unchanged, we evenly divide the uplinks of the original TOR switch between all the local TOR switches. The resulting topology is oversubscribed, but now each server can potentially use much more bandwidth.

Multihoming brings additional costs in terms of switching equipment, rack-space, energy usage and maintenance. As every additional server port could require an additional switch, this solution does not scale well with the number of server ports used. Moreover, for a topology such as VL2, adding TOR switches will leave some of their 10Gbps ports unused, even though they contribute to the total cost of the network.

Each server must receive an additional IP address for every new link. After addresses are assigned, MPTCP and ECMP are enough to utilize the network: servers only have to choose the destination address for each subflow, and the routing will do the rest.

3.2 GRIN

Another solution is to interconnect servers directly using their free network ports, while keeping the original topology unchanged. Each pair of servers that are directly connected in this manner become neighbours. Intuitively, when a server does not need to use its main network interface, it may allow one or more of its neighbours to "borrow" it, by forwarding packets received from them (or packets addressed to them) to their final destination. This solution is depicted in Figure 1.c.

When a server wishes to transmit, it can use both the uplink and the links leading to its neighbours. Conversely, the destination can be reached through both its uplink and via its neighbours. We call the links used to interconnect servers *horizontal or GRIN links*, and those that connect servers to the original topology *uplinks*.

Which servers should we interconnect? It is best to connect those servers that usually do not need to access the network at the same time, otherwise interconnection will not bring major gains beyond improving local throughput. Many distributed applications have workloads that correlate the idle and busy periods across all the servers they are running on, as do the shuffle or data-output phases of map-reduce computations. These applications tend to distribute their work across many racks to acquire sufficient servers and to improve fault tolerance. Ideally, we would know beforehand which nodes run which applications and we would interconnect servers running different applications in the hope that the usage peaks from the different applications are spread out in time. However, this is not possible as the same server is assigned different tasks over short timescales.

Instead, we make the simplifying assumption that *servers in the same rack use their uplinks independently*. If this holds true, it is sufficient to randomly interconnect servers in the same rack, which is very easy to cable. Otherwise, we have two options: either connect servers from different racks, which increases cabling complexity, or change the application schedulers to spread servers across racks as much as possible [4].

Once links are in place, how should we use them? There are a number of options to consider when choosing a path between two random servers, A and B:

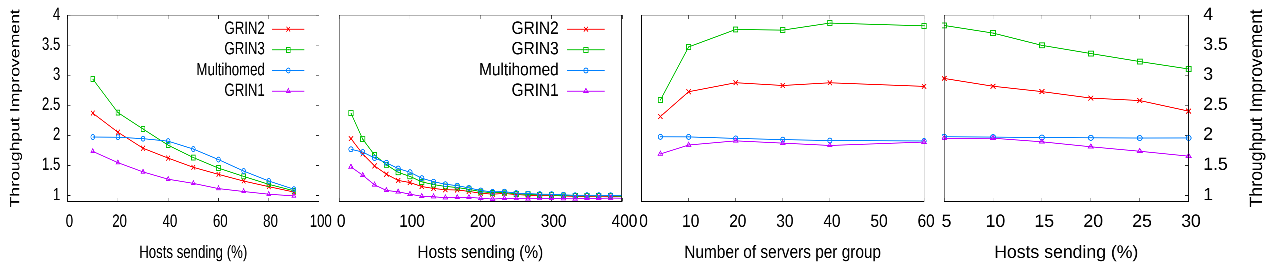
- we may choose one of the paths available in the original topology
- a path may consist entirely of GRIN links
- we may choose any number of intermediate servers, and form a path using the concatenation of all intermediate paths

In order to keep the routing scheme fairly simple and the number of hops relatively small, we are only going to consider paths that contain at most one segment of the first kind. Our experiments have shown that one of the better routing solutions is a random scheme, similar to ECMP. We also found that there is no real improvement if we allow traffic to be forwarded via more than two GRIN links (one following A and one before B).

Whenever server A wishes to connect to server B, it will start by establishing a MPTCP connection as it would have done in the original topology. After this initial phase, server B may advertise the existence of its other interfaces and the address associated with the uplink of the corresponding neighbour. These advertisements are made using a specific feature of the MPTCP protocol called *address advertisement*. Server A will now be aware of an address set that contains the original address of B, together with addresses of B's neighbours.

To start a new subflow, server A will randomly pick an intermediate source between itself and all of its neighbors, and an intermediate destination from B's advertised

[4]The Multihomed topology also relies on the same assumption



(a) Permutation (b) Random (c) Group (d) All-to-all

Figure 2: Simulation results

address set. Each packet is addressed to the intermediate destination, and the real destination address is recorded as an IP option. The packets will be routed from server A to the intermediate source simply by putting them on the appropriate horizontal link. Whenever a server receives a packet which is not addressed to it on one of its horizontal links, that packet will simply be forwarded to the uplink, because all horizontal path segments have a maximum length of one hop. If a packet destined for another server (according to the real destination) is received from the uplink, the server will simply forward it to its appropriate destination via a GRIN link.

Since local routing is done over at most one hop, it does not make sense to interconnect two servers with more than one link. As each server has a single uplink, doing so would only waste available server ports without providing additional bandwidth.

GRIN Properties GRIN links can be assigned local, non-routable addresses; these addresses are not advertised in the original topology reducing deployment costs. GRIN scales seamlessly with the number of server ports, because it does not require additional switching equipment, power or rack space. GRIN is agnostic of the topology used—it can be applied to any topology where servers have unconnected network ports.

There are two primary concerns raised by GRIN: the overhead caused by server forwarding and fairness in the use of neighbours' uplinks. One-hop forwarding restricts the maximum amount of traffic each server may be required to forward to twice the speed of its uplink, i.e. 2Gbps. Fairness means that a server should have priority while using its own uplink. When sending, this can be achieved simply by forwarding local packets first. When receiving, the server can selectively drop packets to control the amount of bandwidth available to foreign flows.

4 Evaluation

We expect both Multihomed and GRIN topologies to perform significantly better than the original topology on average, and at least as well in the worst case. While traffic patterns usually found in datacenter networks are too complex to be covered by a handful of traffic matrices, we chose a few that should provide at least some insight regarding how will our approach behave in a real-life scenario. Intuitively, we are going to see the greatest improvements when either a small number of servers are using the network, or there are a lot of servers transmitting to only a handful of receivers.

Our experiments are based on a simulated VL2 topology consisting of 120 servers, with 20 servers in each rack. Increasing network size up to sixfold didn't lead to any sensibly different results. Using the htsim simulator, we measured the average data rate of each active receiver and compared it to the maximum possible value for the original topology. Each multipath connection is long lived and uses 16 subflows. This number was chosen in order to make sure the random routing scheme will explore as many of the relevant paths as possible, and to mitigate the effect of collisions in the VL2 topology.

We tested three GRIN topologies, using 1 - 3 additional server ports, and one multihomed topology that uses one additional server port. They are referred to as *GRIN1*, *GRIN2*, *GRIN3* and *Multihomed*, respectively. Multihomed requires the same total number of ports as GRIN2, and costs arguably more, while also requiring additional rack space.

We used four traffic matrices in our tests: permutation, group, all-to-all and random. For each traffic matrix, we varied the percentage of active servers or the number of connections per server. We evaluated 20 random instances of every case. Given the small size of the network, we only allow connections between servers from different racks for permutation and all-to-all matrices. In much larger networks, random connections would seldom end up being local. We assume that the network interface is the only limiting factor for the amount of data that can be sent or received by any server.

In a permutation traffic matrix, each active participant transmits to a random destination, and each destination

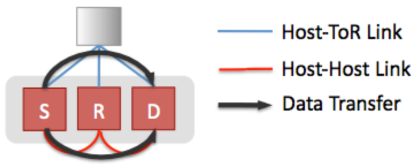


Figure 3: Experimental setup to test forwarding overhead

receives from a single source. As illustrated in figure 2a, lower percentages of active servers lead to significantly better results for GRIN topologies. The performance of these topologies, however, degrades quicker than that of Multihomed, which provides better throughput when 25% and 40% of servers, respectively, become active.

The group connection matrix simulates scatter-gather communication. It randomly assigns every server to a certain group, such that all groups have the same number of members. Within each group, one random server will be the destination, and all others will act as sources. This is the most favorable situation for GRIN topologies, because the large number of sources will fill every link of the receiving server. As seen in figure 2c, the performance benefit starts out at a minimum for the smallest group size, and then increases to near the maximum value relatively quick.

The all-to-all simulation (figure 2d) offers no surprise; the best results are found for the smallest number of active servers, and then gradually decline. The results for Multihomed decline slower, but start at a significantly lower value compared to GRIN2 or GRIN3.

Our final experiments were made for the random traffic matrix. Each case consists of a number of connections (expressed as a percentage of the number of servers), and for each connection we choose at random both the source and the destination. The results, seen in figure 2b, resemble those found in the permutation case.

4.1 Forwarding Performance

If traffic forwarding interfered with other server tasks it could potentially decrease the performance of the datacenter as a whole, defeating the purpose of this work. That is why in this section we conduct a preliminary study to test just how costly forwarding is in practice.

We used the setup shown in Fig.3; all three servers have 2 dual-core Xeon processors running at 2.66GHZ. We installed the Linux MPTCP stack on servers S and D and tested throughput by running iperf. The forwarder runs FreeBSD and the netmap [9] architecture, allowing very fast packet processing in user context.

iperf manages to transfer 1.86Gbps of goodput from S to D. Forwarding MTU-sized packets at 1Gbps is relatively cheap: server R spends only 6% of one of its cores for this task. This is the common case where bulk transfers are relayed over neighbouring idle uplinks. If traffic is forwarded in both directions the load increases to 10% of one core. We also tested the forwarder with minimum sized packets, to seek an upper bound for CPU usage. We found that R can forward 1.5Mpps running one of its cores at 25% utilization.

Even a load of 6% on one core can interfere with other server tasks when the server is running at 100% utilization. The forwarder can increase the loss rate on the forwarded traffic, which will automatically reduce the traffic rate, or it can even switch off forwarding entirely - the traffic will simply use another path to the destination.

5 Related Work

Various solutions have been proposed for increasing core utilization in full bisection networks, such as MPTCP [7] or Hedera [2]. These, however, aim to make full-bisection topologies behave like a non-blocking switch by routing flows in the network to avoid collisions.

Other approaches, such as Flyways [5] or C-Through [10], are based on augmenting an oversubscribed network with additional communication channels that can be used to improve throughput between different groups of servers when the initial latency is not an issue. They try to create the illusion of full bisection in oversubscribed networks; however, the network activity of a single server is still limited by its uplink to 1Gbps.

Server forwarding is also used in topologies such as BCube, where the forwarding effort is much greater, as it increases with each additional connection. GRIN's routing scheme inherently limits the amount of traffic forwarded by each server.

6 Conclusions

Wiring up server ports within the same rack with GRIN is a simple yet powerful solution to utilize available core bandwidth in the likely occurrence that not all datacenter hosts are using their uplinks at the same time. This solution is cheap and feasible to implement over any full-bisection topology. The improvement is significant for idle networks, with every additional connected port bringing 1Gbps more throughput; benefits gradually decline with each server that becomes active. Further, practical tests show that the forwarding overhead is almost negligible for today's datacenter servers. GRIN offers better peak performance, is cheaper, easier to deploy and scales better than the Multihomed topology.

Our initial exploration in this paper leaves many interesting open questions. Practical testing, together with understanding fairness and isolation properties of GRIN with regards to existing traffic are issues that stand out.

If a latency-sensitive flow ends up going through one or more horizontal links, its completion time may suffer an unacceptable delay caused by the additional hops and the fact that each server has priority using its own uplink. Moreover, long flows can significantly increase network utilization, which in turn will have a negative impact on the the overall perceived latency. There are a number of possible solutions to these issues, such as never choosing a path with horizontal segments for latency-sensitive flows and employing the DiffServ architecture. In order to do this, however, we must find a way to differentiate between long and short flows with reasonable accuracy.

Finally, there are unexplored routing possibilities, some of which require the ability to use horizontal paths greater than one hop in length. This may bring additional throughput benefits, but at the same time can considerably increase server routing effort.

Acknowledgments

This work has been partially supported by a Cisco University Research Award and the CHANGE FP7 project.

References

[1] AL-FARES, M., LOUKISSAS, A., AND VAHDAT, A. A scalable, commodity data center network architecture. In *Proceedings of ACM SIGCOMM 2008*.

[2] AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., HUANG, N., AND VAHDAT, A. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of USENIX NSDI 2010*.

[3] GREENBERG, A., JAIN, N., KANDULA, S., KIM, C., LAHIRI, P., MALTZ, D., PATEL, P., AND SENGUPTA, S. VL2: A scalable and flexible data center network. In *Proceedings of ACM SIGCOMM 2009*.

[4] GUO, C., LU, G., LI, D., WU, H., ZHANG, X., SHI, Y., TIAN, C., ZHANG, Y., AND LU, S. BCube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of ACM SIGCOMM 2009*.

[5] HALPERIN, D., KANDULA, S., PADHYE, J., BAHL P., AND WETHERALL, D. Augmenting data center networks with multi-gigabit wireless links. In *Proceedings of ACM SIGCOMM 2011*.

[6] KANDULA, S., SENGUPTA, S., GREENBERG, A., PATEL, P., AND CHAIKEN, R. The nature of data center traffic: measurements & analysis. In *Proceedings of ACM IMC 2009*.

[7] RAICIU, C., BARRE, S., PLUNTKE, C., GREENHALGH, A., WISCHIK, D., AND HANDLEY, M. Improving datacenter performance and robustness with multipath TCP. In *Proceedings of ACM SIGCOMM 2011*.

[8] RAICIU, C., IONESCU, M., AND NICULESCU, D. Opening up black box networks with CloudTalk. In *Proceedings of USENIX HotCloud 2012*.

[9] RIZZO, L. netmap: a novel framework for fast packet I/O. In *Proceedings of USENIX ATC 2012*.

[10] WANG, G., ANDERSEN, D. G., KAMINSKY, M., PAPAGIÃNNAKI, K., NG, T. E., KOZUCH, M., AND RYAN, M. c-Through: Part-time optics in data centers. In *Proceedings of ACM SIGCOMM 2010*.

[11] WISCHIK, D., RAICIU, C., GREENHALGH, A., AND HANDLEY, M. Design, implementation and evaluation of congestion control for multipath TCP. In *Proceedings of USENIX NSDI 2011*.