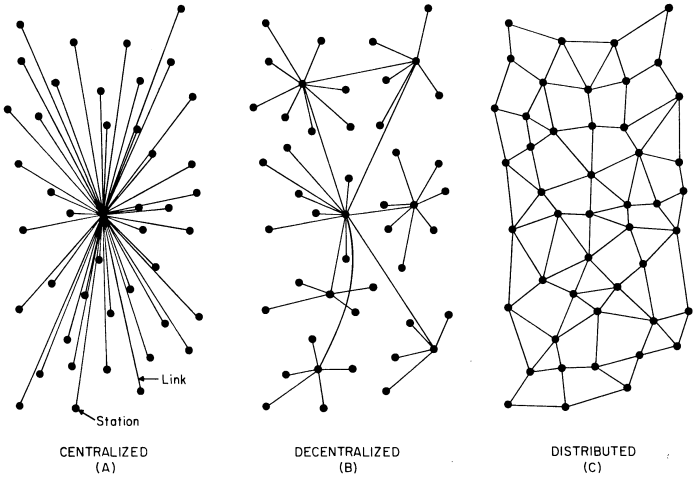
Rethinking Datacenter Durability

Computing and networking has never been well integrated. Unsolved fundamental distributed systems problems are now the most pervasive source of failures in today's datacenters. Application developers see the long pernicious shadow of these few core issues hundreds of times a day: lost transactions, lost data, byzantine failures, inconsistent data, unexplained crashes and deadlocks. Datacenter operators observe software bugs, replication failures, configuration drift, timing quirks and networking all coming together to wreak havoc, cascading into major failures and user-visible outages across all regions. Users constantly experience synchronization failures on files and applications across their many devices.

Datacenter architects today build their infrastructures using two kinds of boxes: *switches*[1] and *servers*[2]. They connect them using individual cables, which they bundle together to make them convenient to route within and around physical structures. This forms a *centralized* or *decentralized*[3] topology, where the switches become hubs and servers become leaves.

Paul Baran's classification provides insight: `CENTRALIZED (A)` shows 46 univalent nodes connected to a 47^{th} high radix or *valency*[4] hub. If the central hub dies, all nodes are cut off. `DECENTRALIZED (B)` shows 46 nodes and links, 7 nodes with a 5-7 valency serve as switches, and 40 as univalent (leaf) nodes. If a center node fails, the network fractures into isolated partitions; and only nodes within the local partitions can continue to communicate locally. `DISTRIBUTED (C)` shows 47 identical, multivalent (valency ~ 5) nodes, width 98 links. Resilience can now be maximized: failed nodes are routed around, and many links must fail on some nodes before they finally become isolated.



Paul Baran's Classification of Topologies

Desirable & Undesirable Topologies

We all know that centralized topologies exhibit bottlenecks and become Single Points of Failure (SPoF's). *Decentralized* topologies[5], pervasive in datacenters, is surprisingly *non-optimal* when looked at from a modern perspective of distributed microservices, container life-cycles, migration, elasticity and resilience[6]. The failure points in decentralized topologies are mitigated by parallel redundancy in switched networks[7].

Two types of teams co-evolved to manage datacenters: one to design and manage the networks, and one to program and manage the servers. This worked when datacenters had a single owner or tenant, their applications and physical equipment evolved slowly, and different business units worked in silo's to optimize their line of business. This is no longer a viable architecture in our highly dynamic multitenant datacenters today. Distributed applications can no longer afford to be held back by the slow pace of networking innovation.

Maximally Resilient Infrastructure

Within the same *or lower* capital cost structure for datacenters, we can create denser, and hence more resilient and flexible, physical topologies by connecting *swervers/cells*[8] directly in a neighbor to neighbor (N2N) or directly-connected arrangement rather than only through the switched network[9]. Paul Baran's fully *distributed* topology provides far more robustness than the current *decentralized* topologies. These dense networks represent an opportunity that has so far been underestimated by datacenter architects.

The time has come to recognize the remaining genius in Paul Baran's insights. And ask why they are not yet widely deployed in datacenters, where their benefits to resilience and security are needed most.

SDN: Programmable Networks

The industry's response has been to introduce the notion of Software-Defined Networking (SDN) to *program* network behavior. But SDN's programmability is confined to the *network control plane*, which remains under the control of network owners and operators. A new breed of high-performance forwarding chips based on PISA (Protocol Independent Switch Architecture)[10] can be programmed by the P4 language. While this is a step in the right direction, it is too low a level of programmability[11], and support conventional notions packet forwarding to destination addresses.

In P4, programmers declare how packets are processed, and a compiler generates configurations for a protocol-independent switch or NIC. For example, network programmers can program the switch to be a top-of-rack switch, a firewall, or a load-balancer; and add features for monitoring or automatic diagnostics. While programmable switches may be inevitable, and a promising approach to improve the performance of datacenters, they are still (a) under the control of the network owners and operators, and (b) limited by the low-level endpoint routing and forwarding paradigm of today's network engineering. What is needed to complete this revolution is to include the servers as first class members of this set of devices which are allowed to route packets as well as process them.

TRAPHs: Programmable Topologies

We believe there is a critical layer missing between application and infrastructure: the layer which presents the evolving *graph* relationships of modern distributed applications, as a substrate that application programmers can own and manage, but which work in conjunction with the simpler forwarding functions within the NICs and switches. Presenting precise, programmable, and deterministic-when-needed, topologies as tools and resources to the application architect. For example, application programmers can program these TRAPHs (Tree gRAPHs) with a Graph Virtual Machine (GVM) that provides services such as distributed consensus, atomic broadcast, and presence management among members of a cluster or microservice set.

Examples

The advantage of TRAPHs over shared network topologies, is that application developers can program their behavior instead of having to wait for permission, or suffer the externalities of the network optimizing itself without regard to the application's health. This simplifies some important use cases such as:

Managing Infrastructure as Sets, Graphs & Tensors, Instead of Files & Lists All routing is predicated on building shortest path trees, e.g. Bellman Ford for L2, or Dijkstra at L3. The roots for these trees are in the switches, and thus under the administrative control of network owners and operators. With TRAPHs, large subgraphs, or graph-covers comprising nodes and links allocated to a particular tenant, may be used by that tenant for any topology whatsoever, including allocation to a sub-tenant. *Graph computing* on TRAPHs (Tree-gRAPHs) enable automatic mapping of the natural DAG relationships of distributed applications on nested datacenter infrastructure resources.

Logical & Virtual Segregation Planes Enable capability-based security graphs, to provide secure containment of communication environments for multi-tenant infrastructures. E.g., exchange the management of lists (ACL's and `iptables`) by replacing them with richer and more manageable `graph equations`. Virtual Segregation Planes: Graph application Web services description. Erasure coding description. Machine learning description.

Coherent graph overlays where cache *heterarchies* can co-exist to automatically manage the placement and eviction of caches based on request patterns. One example would be a coherent configuration file system, which provided a unified mechanism to keep configuration files synchronized, for Docker, etc. Another would be a *coherent* `memcached`.

> Allowing application developers to build and manage their own routing substrate under API control would dramatically improve the performance, efficiency and flexibility of modern infrastructures, reducing inter-tenant interference, enabling privacy, and improving manageability.

Notes

[1]We refer to all network devices as switches. Those that route at layer 3 are simply layer 3 switches.

[2]There are also many hybrid kinds of boxes, often labelled as *appliances*, such as load-balancers, firewalls, etc.

[3]In Paul Baran's terminology (Distributed Communications).

[4]We use valency to denote the number of physical ports on a cell. This is to distinguish us from radix, used in switches, but is equivalent to degree of a vertex (δ(V)) in graph theory.

[5]Decentralized topologies may be economically appropriate for large-scale geographically disparate systems like the interstate roadways and airline networks.

[6]Today's datacenter networks evolved from their roots in the ad-hoc connection of Ethernet broadcast domains with their switches housed in wiring closets, and managed by individuals with specialized expertise in routing and proprietary management interfaces

[7]modern Clos networks typically have two, three or four *slices* of spline and leaf switches, along with multiple sets of cables.

[8]We combine switches and servers, into a single concept: `cells`, and make them *substitutable* (i.e. although they may not be identical in all aspects of their capabilities, they can at least be managed as one 'type'.), which, in turn, makes them *fungible*, and easier to manage. The additional density of the physical topology, afforded by the `cell`'s 'middle' range valency (5 to 9), enables far richer virtual topologies to be built.

[9]Just as with the naysayers in Paul Baran's time, skeptics today will often invoke *argumentum ad ignorantium* (arguing that a proposition is true because it has not yet been proven false (or vice versa). When pressed they will often fall back to the argument that we've always done it this way, or, it has to be legacy compatible, etc. We contend that modern datacenter networks can gain significant advantages by adopting Paul Baran's *fully distributed* model. This may not have been obvious in the past because networks were relatively static, and security was not considered a particularly difficult or important problem.

[10]PISA is Protocol Independent, it allows P4 programs to specify forwarding behavior for packets. It is suitable for describing everything from high-performance forwarding ASICs to software switches. Field Reconfigurability allows network engineers to change the way their switches process packets after they are deployed.

[11]P4 presents a paradigm of "programming with tables" to developers. This paradigm is somewhat unnatural to imperative (or functional) programmers, and it takes some time to get accustomed to the abstraction. It also, occasionally, leads to awkward ways of expressing functionality [Paxos Made Switch-y].

Allowing application developers to participate in the setup and tear-down of dynamic topologies, would result is a datacenter fabric that is simpler, more resilient, easier to manage, lower cost, more secure and fundamentally more recoverable.

