Network Defined Software

In conventional datacenters, servers communicate over a *separate switched network* comprised of multiple individual switches. Switches are connected together with cables which together form a physical fabric. Servers are also connected by cables, but only to the Top of Rack (ToR) switches at the fabric edge. Conventional switches use special protocols to create logical *trees* with themselves as *root*; so that the resulting logical graph can be used to route messages between servers in a loop-free manner. Only the switches are aware of these trees, the servers (and especially the applications on them) are not.

Switched networks are generally under the control of a separate team of network owners and operators. Applications are unable to take advantage of the logical structures and resources available *within* the switched network, except through an Any to Any (A2A) abstraction: i.e., an illusion that any server may address (and talk to) any other server, *transparently*. Unfortunately, the resulting switched network is anything but transparent. The *shared* nature of this resource requires that it provide buffering, congestion control, access control, and management complexities, many of which are notorious for causing cascade failures and introducing vulnerabilities; in addition to the substantial capital and operating expenses of this model.

Network Defined Applications

We aim to provide a new kind of networked server infrastructure, where *applications* can be aware of and manage their own trees, and not just be passive players in an A2A abstraction. We call this new kind of server a `cell`: an encapsulated and autonomous unit of compute ($\geq$ 8 cores), Memory ($\geq$ 8 GB), storage ($\geq$ 8 TB) and network ports ($\geq$ 8). The critical difference is that all components required to build application infrastructures are in the cell – it does not need any other resources outside the box, except for other `cells` like itself, *and*, each cell has the necessary and sufficient ports for a resilient fabric to emerge.

When applications running on these cells are aware of the trees, they can enjoy guarantees not possible through Conventional DataCenter (CDC) switched networks. For example, while applications continue to communicate over a selected tree, they can expect ordering guarantees. This addresses the fundamental manageability, resilience, and security issues identified above. More importantly, it provides a simpler and more flexible way to build, manage and securely partition application graphs on modern cloud infrastructures. This layered, recursive, mathematically based structure of TRAPHs (Tree-gRAPHs) enables true multi-tenancy, relative addressing, VM, container, and λ-based application graphs.

Conventional Networks

CDC networks are organized in multi-rooted tree topologies in a separate switched networking fabric. They typically rely on Equal-Cost MultiPath (ECMP) to split flows across multiple paths, which can lead to significant load imbalance. Splitting individual flows in this way is also not preferred because of potential packet reordering that conventional wisdom suggests may negatively interact with TCP congestion control. However, the concern for TCP congestion control is only part of the problem, when links fail, random packet spraying can have an adverse impact on not only regular application performance, it can also have a devastating effect on various distributed algorithms that depend on liveness for their correct operation.

Application Networks

The switched network controls all the trees in a CDC, i.e., the roots of the trees are created in, and managed by, the switches. It would be far more helpful to root these trees from the servers, where a much more mature development and deployment system exists. Furthermore, "Anyone who has experience in building distributed systems would be aware that the easiest phase in the entire lifecycle of a system is the coding phase. The operationalization phase (making the system actually work in production) is the hardest and needs tooling, measurement and extensive testing to ensure things work as expected." [LinkedIn Ambry].

Being able to build and manage trees from an API driven by the application infrastructure allows developers to define their testing, measurement and monitoring environments dynamically, on their terms, and in their timescales – often down to milliseconds in microservice or container-based infrastructures; while the underlying infrastructure trees *confine* the developers to tenant domains they have been allocated to.

TRAPHs

In the N2N (switchless) topology we call TRAPHs (Tree-gRAPHs), every cell is a first class citizen in the new fabric, and has a tree rooted on itself. These trees form a general resource that can be used for both management (control) and computation (coherence and serialization). These functions can now be under the control of a combined DevOps infrastructure team, not just network managers, who know very little about the requirements and behavior of the distributed systems infrastructure.

TRAPHs on Convetional Networks?

One would think that, in principle, TRAPHs could be implemented on top of existing networks. After all, can't all the nodes already see each other? Well, that's precisely the issue, every node *can* see each other, and we have extra work to do managing ACL's and `iptables` to prevent that. Having switches be the only place trees can be managed and rooted requires a northbound API, and connectivity from the server to the management switch – which is not available, for example, when the switch fails, or starts acting up.

Rooting the TRAPHs the cells themselves, enables us to think differently. It enables us to escape the tyranny of *lists*, and to begin thinking in *graphs*: "Network defined Software". As long as trees are on just the switches, it is difficult for developers to think about how best to exploit them.

When combined with the `TIKTYKTIK` property of `links`, we can now begin to more comprehensively deal with some of the key issues in distributed systems that have plagued our industry for years. We distinguish two potentially cooperating disciplines: Software Defined Networking (where existing switched networks are made programmable), and Network Defined Software (where the cells are made programmable).

TRAPH management: Vantage Points

Going from development using Docker, to production can be problematic. Developers need only docker on their laptops to do their work. However, when going to production, additional concerns must be addressed: (from the bottom up): Docker, Kubernetes/Mesos, libnetwork, rolling deployment, monitoring, supervision, configuration changes, peer discovery, container hosting, code quality, security.

While these new tools can help smooth the transition of an application from the development environment on the laptop to the production environment on the server, they do very little for the network connectivity (who can talk to who, and who might interfere with them). TRAPHs enable realm operators, tenant managers and developers to cooperate. A blank TRAPH tenancy template provided by the DevOps team enables developers to incorporate a realistic network environment, and modify it using layered TRAPHs above to construct the communication environment needed by the application. When the time comes to test on the real systems, both the application code (in containers) and relationships (in TRAPH descriptions) are moved to a cordoned-off portion of the infrastructure for testing. When the system is ready for live traffic, a fraction of real traffic is siphoned from a production sub-tenant to the new version (the canary). When everything is seen to be working well, all the traffic is moved over, and the previous version is retired.

These different *vantage points* enable separation of concerns. Datacenter managers provision jurisdictions, jurisdiction managers provision tenants, and tenant managers provision sub-tenants. Although these distinctions are somewhat arbitrarily named, they are all based on the solid ground underlying the mathematics of graph covers, and all the richness and utility of graph theory itself. TRAPHs are layered graph covers. Initiated from a single cell (the vantage point), protected by redundant cells using robust consensus protocols to define the current leader. TRAPHs facilitate the clear separation of different roles in the system, by providing each of them their own vantage point with unique (to them) responsibilities and authority.

Naked vantage points are single cells which define their roles on their own tree (first mover applies). Redundant vantage points recruit local (`link`) neighbors to check them and provide failover should they die or become incapacitated. Among other things, this allows keys to be managed securely.

Vantage points grow trees rooted on themselves, defining a *set* of `cells` and `links` (vertices and edges in graph theory). The set of cells define a TRAPH. i.e. the Vantage point is the root, the tree spreads out along the transitive `link` connectivity; pruned `links` between cells in the same set are failover candidates within the TRAPH. Links between different TRAPHs are disabled from below.

TRAPHs and Secure Confinement

Networks need automation (just ask the U.S. military). But automating the actions that administrators do on their keyboards with scripts, or configuration management systems, has failed hopelessly to produce the robust, agile and secure foundations needed in today's command and control networks. Resilience comes from simplicity, and adding more complexity on top of existing complexity doesn't get you there. TRAPHs are "beyond automation", they provide a solid, mathematically sound underlay which allows developers and operations people to reason simply about the organization and state of their networks, and which can provable guarantees that certain properties are maintained through perturbations (failures, disasters, attacks) and the recovery of the infrastructure from them.

TRAPHs are a valuable tool to outwit intruders. If they penetrate part of the network, operators can logically cut off that segment, pulling the workloads, users, and address space out into another TRAPH somewhere else in the infrastructure where it can be quarantined, monitored, and analyzed. Intruders would be left with a non-functioning "honey TRAPH.". This is a generalization of an approach requested by DISA.

TRAPHs, Physical Topologies and Performance

Cut-based metrics such as bisection bandwidth and sparsest cut are commonly used to estimate throughput, because minimum cuts are assumed to measure worst-case throughput. However, while this is true for the case where the network carries only one flow, it does not hold for the common case of general traffic matrices (TM's). Jyothi, Singla, Godfrey & Kolla, the bulk of proposals, for all TMs, perform *worse* than networks based on *random graphs*, with the latter?s advantage increasing with scale.

Proposals for networks based on expander graphs, including Jellyfish's random topology[1], Long Hop, and Slim Fly, have nearly identical performance for uniform traffic (e.g. all-to-all).

Erasure Coding

Parallel File Systems

Load Balancing

Distributed Applications

When a switch fails in a CDC network, the servers suffer a liveness disruption, which triggers timeouts, which in turn triggers re-elections in consensus systems such as Paxos and Raft. One way to view the key problem is to consider properties like eventual consistency as a liveness property – not a safety property – which is trivially satisfiable by itself. In practice, liveness and safety properties must be taken together.

> "Safety and liveness are two important kinds of properties provided by all distributed systems. Safety guarantees promise that nothing bad happens, while liveness guarantees promise that something good eventually happens. Every distributed system makes some form of safety and liveness guarantees, and some are stronger than others. For example, atomic consistency guarantees that operations will appear to happen instantaneously across the system (safety) but operations won't always succeed in the presence of network partitions (liveness, in the form of availability)." [Peter Bailis].

TRAPHs + `links` provide a new form of liveness, based on *events*, instead of *durations*. This allows us to, for example, replace timeouts. The problem with timeouts on a shared network is that you can never get them right, because what someone else is doing determines the optimal value of the timeout. Distributed systems end up having dependencies as they scale arising from feedback loops where they affect each other mutually. The result is unexpected and sometimes large scale cascade failures which cause outages in the application infrastructure.

Secure Separation & Confinement

Cells incorporate an extremely simple and low level execution environment we call subvirtualization. This first execution environment resides directly on the hardware, below all operating systems, kernels, hypervisors, device drivers or any other control mechanisms. This separation kernel or "Secure MicroVisor (SMV)" is simple, less than 50K lines of code. It statically allocates execution environments (coarse – i.e., 1GB) chunks of memory, and a fixed subset of the available cores to *hypercontainers*, which resemble conventional virtual machines (with or without fat hypervisors), and unikernels, which make no distinction between the library and the kernel. These execution environments are each separated from each other, and under the control of a static, unimpeachable SMV.

This distinction provides an important capability that a conventional server and its OS environment does not: a secure place to distinguish the control plane from the data plane from the management plane. Normally, switches do this by virtue of being a completely separate completely separate operating system running on a separate box. This distinction is now provided in `cells` by using the principle of a separation kernel to guarantee the separation of these different execution environments within a single cell.

This, in turn, allows us to provide a simple and secure way to manage a heterarchical set of cells as TRAPHs, each as a separable, recursive, control plane (each controlled by a single vantage point). From the bottom to the top, this enables TRAPHs (sets of cells and links) to provide (hierarchical) sub-graph confinement. They replace the editing of ACL's and iptables as the method of network segregation. From an application perspective, the subvirtualization layer provides a set of "local services" that:

- Because the NIC will have the active part of the state in the `link`, it can invoke the node directly, and the node can involve the link directly,
- By not exposing the the APIs to higher layers, we have effectively separated the control plane from the data plane inside the cell, obviating the need for a separate switched network to achieve this isolation.
- We can do things like break and heal links in the NAL which can isolate.

Other Systems That Have Tried, But Are Unable To Achieve These Goals

Shortest Path Bridging

Shortest Path Bridging (SPB) allows all links to be active through multiple equal cost paths, provides faster convergence times to reduce down time, and simplifies the use of load balancing in mesh network topologies (partially connected and/or fully connected) by allowing traffic to load share across all paths of a network. SPB is designed to *virtually eliminate* human error during configuration and preserves the plug-and-play nature that established Ethernet as the de facto protocol at Layer 2.

SPB is similar to TRAPHs in several respects. However, it works only among switches. It is designed to provide a similar Zero Touch goal, but only for switches. I.e. it maintains the distinction between servers and switches. The DTA algorithm (and patents) were established before the IEEE released the SPB protocol, but it is still appropriate to ask if SPB can be used instead, especially because it has multiple well tried implementations that have proven its effectiveness. Similarities include:

- Multiple rooted trees (Dijkstra's for SPB, DTA's for TRAPHs).
- Intended for Zero-Touch (to virtually eliminate human error during configuration).

Key differences include:

- TRAPHS work only on cells (multi-valent servers), SPB works only on switches.
- TRAPHS use the RAFI protocol, which is a purely port based (relative) addressing. SPB works like conventional switches, from one Ethernet broadcast domain to another.
- TRAPHS are under the control of the application developers. SPB is under the control of the network operator.

Dijkstra vs. DTA

Dijkstra's algorithm uses the principle that if Node R is on the minimum path between nodes P and Q, this knowledge of Q implies knowledge of the minimum path from P to R.

In the Dynamic Tree algorithm (DTA), dynamic trees are built for every cell as a root of that tree P. This implies that every other cell "knows of" the root, and the unique direction in which to find it[2]. It also means that the root *does not* know of the existence of P and Q, on *this, or it's tree*, except for cells one hop away. This allows the tree owned by R to self-stabilize through failures of cells and links, based only on local information at every other cell on the tree, without having to inform the root R, unless it 'was previously' one link from R.

This is what enables the DTA to be resilient to failures, and hence scalable. The same reasoning applies to every cell, which builds its own tree, just like R. This is a similar principle to dynamic programming, applied to graph theory.

The Dynamic Tree Algorithm (DTA) is substantially different to Dijkstra's Algorithm (DA). While DA is a GEV algorithm[3]; it expects to put all nodes into a priority queue (with an initial distance set to ∞) during the initialization. In DTA, the only node that exists $\exists$ in the beginning is the root; all other nodes are discovered, and knowledge of them is maintained in a strictly distributed fashion, as a local property one hop away only from each local (LOV) observer. This means that DTA is an LOV algorithm. It doesn't know either the number, or the identity of the nodes on its network. This is deliberate. Not only does this make the system scalable, it provides a foundation for them to create private paths on the graph. The root in DAT knows only its nearest neighbors: those connected directly to its ports, 1 hop from them. This recursive relationship provides a true distributed data structure: no one cell, including the root, has complete GEV knowledge of the entire tree. This allows the tree to grow and shrink dynamically [ref] on its frontier, without having to send packets to the root. This is one aspect of the DTA that enables it to scale. It also allows it to be resilient in ways that GEV algorithms, such as Dijkstra, cannot. [Ariel Felner UCS paper.].

Fundamentally, DTA is an LOV algorithm; each cell knows only about its nearest neighbors, which it interacts with continuously to maintain presence. Those neighbors, in turn, know only about their nearest neighbors. These transitive relationships can grow to any arbitrary size. The data structures which define the graph are fully distributed; no one cell has a complete view.

Other Aspects Enabled By TRAPHs

Resiliency Measure: Counting Trees

Cayley's Formula N^{N-2} tells us how many different trees we can construct on a complete graph of N vertices (cells/nodes). Cayley's Formula is a special case of complete graphs, where every node is connected to every other node (unbounded degree/valency). in DTA, we explicitly bound the degree or valency of the cells to say ≤ 8. This reflects the local connectivity in our physical fabric[4], with a potential extension to a couple more connections we call Klienberg Links. The more general problem of counting spanning trees in an undirected graph is addressed by the matrix tree theorem[5].

The adjacency list is a GEV structure, collected (using `treecast`) in order to display the graph on the screen for human viewing. Once we have the adjacency list, and the degree diagonal, it is straightforward to subtract one from the other, and calculate the determinant, forming a potentially useful measure of "resiliency" of the network, i.e., the number of different spanning trees.

Goodness measures

Resilience comes from the very large number of alternate spanning trees available

Security comes from the many alternate paths available to verify the identity of other entities in the network.

Security also comes from the

Combining TRAPHs with Links

The TRAPH approach provides some capabilities that are more valuable than merely an alternate routing scheme that is free of duplicated and re-ordered packets, it provides a basis on which to construct solutions to some of the most pernicious problems in distributed computing today. From Universally composable synchronous networks:

> "In synchronous networks, protocols can achieve security guarantees that not possible in an asynchronous world: they can simultaneously achieve input completeness (all honest parties' inputs are included in the computation) and guaranteed termination (honest parties do not "hang" indefinitely).
>
> The widely-used framework of universal composability (UC) is inherently asynchronous, but several approaches for adding synchrony to the framework have been proposed. However, existing proposals do not provide the expected guarantees[6].

Traditional servers are the source of synchronization in conventional architectures. In TRAPHs, links are the source of synchronization. Unlike servers, links can be slowed down, stopped, restarted and reversed. It is this fundamentally simple nature of links that solves many more problems in distributed systems.

All timing in TRAPHs is based on events. Because lost application events are replaced by other events guaranteed by the link, we can avoid many of the problems in modern systems which assume *durations* in time (i.e. timeouts), e.g. timeout storms in consensus, reconstruction storms in erasure-coded storage, etc.

A question of Costs

Networks cost 25-30% of a large compute cluster.

Definitions and Questions

Safety says if you get an answer, it will be the right answer.

Liveness says you will get an answer *eventually*

Communicating events loops don't have plan interference. Communicating event loops. Logically there is a singe thread of execution for the event loop. Until it finishes its work, no-one else can take over. In Multithreading, the OS will take an interrupt on any instruction boundary.

Multithreading , even with a single core, gives you all the problems of asynchronous computation. The time slice can happen at any assembler instruction, so you can't control the interleaving.

Spanning tree is a tree over a graph in which some of the edge weights are minimized [Skiena – Minimum Spanning Trees].

Median graph is an undirected graph in which three vertices a, b and c, has a unique median: $m(a, b, c)$ that belongs to shortest paths between each pair of a, b, and c. Note that we extend this principle in the tree Paxos mechanism to identify the natural leader of that consensus set. Although this has not been proved, we expect that any *odd* number of cells may be used for such consensus. The semi-planar physical connectivity of our fabric allows up to 9 nodes to be connected in a single cluster. The node in the center is the leader. the nodes one hop away (i.e. N, NE, E, SE, S, SW, W, NW) are the followers. If any follower detects a failure in its link entanglement it notifies the "ring" of followers around the master using an NTF (non-tree function), described in the patents[7].

Combining TRAPHs with Wireless

A primary distinction in the Earth Computing Fabric, is its use of `links` (described separately), which provide certain properties not available in a conventional switched fabric. The owned N2N links are directly connected between `cells` without switches, are 100% consistent, regular and predictable, and we will not see anything unexpected on it because we control the links 100%. Nothing unusual can happen without being noticed: the link will be effectively broken. The protocol is explicitly different to Ethernet, although we can use existing Ethernet NIC hardware. e.g. if the packet bits don't conform, the link is broken, you cannot do that with (for example) wireless environment, unless, each link is a separate channel. We believe this is a solvable problem, but we do not wish to be distracted by the intricate details of wireless protocols and standards at this point.

On a wired network, we can approximate a single photon with a single packet on the wire, even though, in principle, we need only 2-4 bits (or a 2x2 or 4x4 matrix of elements). This is much more difficult to achieve on a wireless network. For many of the same reasons we cannot make `TIKTYKTIK` work through a switched network, we also cannot make it work through a wireless network. The `TIKTYKTIK` property is *monogamous* it works only between a bipartite pair of `cells` on a single bidirectional communication path.

TRAPH Performance

What happens when the processor isn't fast enough is that traffic gets dropped. A lot of traffic may be dropped, anyway, since that is how congestion is handled, if done correctly. RED (Random Early Detection) is a method used to randomly drop packets in queues in order to prevent them from filling and tail-dropping packets. This can help to prevent TCP synchronization. A lot of drops occur on switches, where multiple ports of a speed may need to send to another single port of the same speed.

Notes

[1]Jellyfish is more cost-efficient than a fat-tree, supporting as many as 25% more servers at full capacity using the same equipment at the scale of a few thousand nodes, and this advantage improves with scale.

[2]DTA keeps an ordered list of the most preferred to the least-preferred direction, or port, to the root, based on the local order of arrival of the `discover` packets. This represents preallocated failover trees so that recovery from a link or cell failure is as fast and deterministic as possible

[3]See: Ariel Felner, Dijkstra's Algorithm versus Uniform Cost Search or a Case Against Dijkstra's Algorithm.

[4]The result will be smaller if cells can only connect with immediate neighbors in a 1, 2 or 3 dimensional space. Earth Computing has between 6 and 8 ports per cell, i.e. $\Delta \leq 6$ or $\Delta \leq 10$. Note δ is used for the degree of a particular cell, whereas Δ is used for the maximum degree permitted by any cell in the network. Assume 2 ports on the motherboard. Assume PCIe cards have an additional 4 ports. Thus, with One quad ethernet PCIe card we get $\delta = 4 + 2 = 6$ ports. With Two quad ethernet PCIe cards we get $\delta = 4 + 4 + 2 = 10$ ports. We would expect to use the PCIe ethernet ports as local links. We could use the two motherboard ports as Kleinberg Links.

[5]Note: this is the approach Simone Severini uses to construct the Graph Laplacian.

[6]A classic response to this problem is proposed by Katz, et al., for a novel approach to defining synchrony in the UC framework by introducing functionalities exactly meant to model, respectively, bounded-delay networks and loosely synchronized clocks. They show that the expected guarantees of synchronous computation can be achieved given these functionalities, and that previous similar models can all be expressed within their new framework.

[7]Note: the median graph is very closely related to a Distributive Lattice, which is the basis of the data structures in each cell on which the metadata tensor is based

