1. Topology

1.1 From Ethernet to Æthernet

It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

1.1.1 Ethernet was Born

The Original concepts of Ethernet, developed 50 years ago, help us understand the thinking, theoretical concepts, and guide us to practical implementations. It is important to understand the design philosophy, so we can learn the most from the intuition that created the Ethernet revolution. It is instructive to trace the initial intellectual and conceptual steps when Ethernet was first developed, and make sure we are not missing some invaluable intuition.

The original Ethernet was a single coax cable (photon cavity) Half-Duplex 'Bus': alternating between listening and transmitting in 'slots' on the bus.

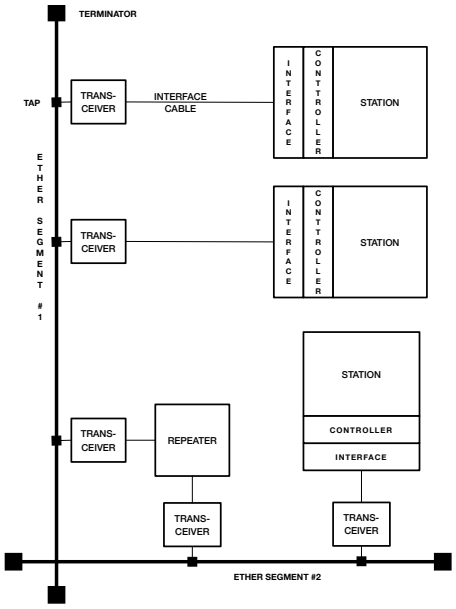


Figure 1.1: Original Ethernet Concepts

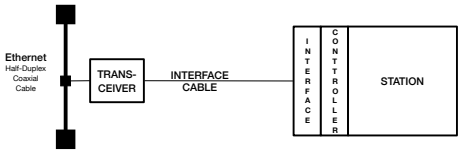


Figure 1.2: Ethernet Components

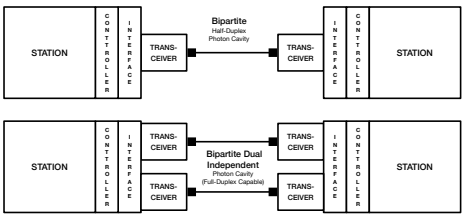


Figure 1.3: Chiplet Æthernet: HDX/FDX

These temporal 'slots' were time *intervals*: controlled and measured against a local oscillator (often a crystal), which had their own drift and stability characteristics. In a half-duplex world, such as on a single coax cable, this meant that the Transceiver (Transmitter + Receiver) would transmit for half the 'time' and listen for the other half the 'time', and be able to detect collisions (the receiver monitors what it itself is transmitting and compares it to what it is receiving to see.

The transmitting station is immediately provided with electrical (signal) feedback which lets it see (a) that it's transmitter was working, and (b) what other receivers might be seeing. From a Shannon perspective, we call this *Initial Information-Feedback (IIF)* and reserve the definition of *Perfect Information Feedback (PIF* for the case where the SerDes at the other end is reflecting what it sees. See section XX for the full theory behind the Dual Back-to-Back (DB2B) Shannon model works.

1.1.2 Ethernet Evolves

Ethernet rapidly evolved to a full duplex situation where there are two separate connections to the media. One in the transmit direction, and

one in the receive direction. However, if one direction is working, and the other direction is not, we could always (in Æthernet) revert back to communicating independently on each cable. This may not have the ideal performance characteristics, but remains a valuable redundancy tool to locally diagnose (and report) unidirectional errors, and flakey fiber connections. This is effectively a dual redundant 'communication' verification tool that can be algorithmically self diagnosing, perhaps in combination to Link training in Modern Ethernet.

1.2 AEthernet Configuration

We begin with two cells, and extend to three. This is the minimum irreducible graph for healing around a broken link. Links can be broken in both directions, or one direction at a time (unidirectional failures)

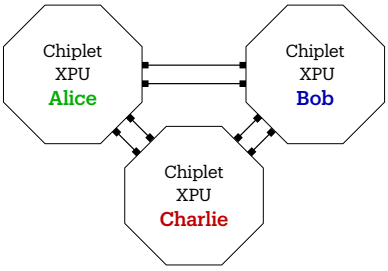


Figure 1.4: Uninitialized XPU Links

1.3 Chiplet XPUs

1.4 Configured Links

We go from Chiplet Servers to something more generalized - XPU's

This 3×3 Tile is the basic fault-tolerant Tile.

Unactivated Links go from being dead to being alive by exchanging configuration packets. These establish the *direction* of the links from

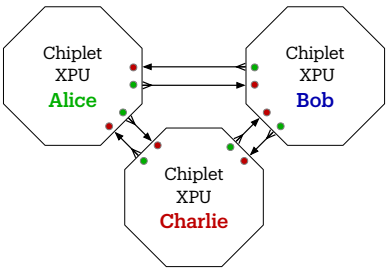
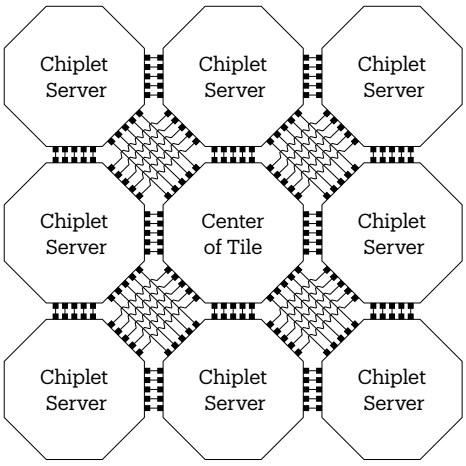


Figure 1.5: Configured Links for TX/RX

1.5 From Repeaters to Switches to Routers, and Back to Repeaters

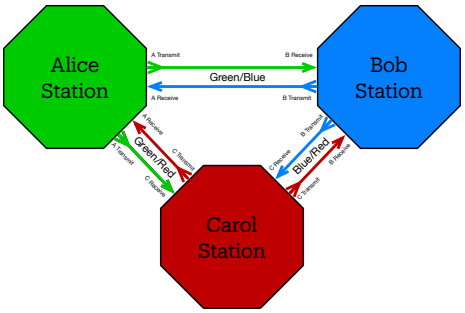
Bob Metcalfe's concept of a repeater was like what Heaviside discovered – a device to take a fading signal and boost it to recreate a strong signal so as to propagate further. That was the simplistic 'engineers' view of how it worked. The 'physicist's view' was far more interesting – they took a transmission line model, where characteristic impedance $Z_o = \sqrt{\frac{L_o}{C_o}}$.

It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.



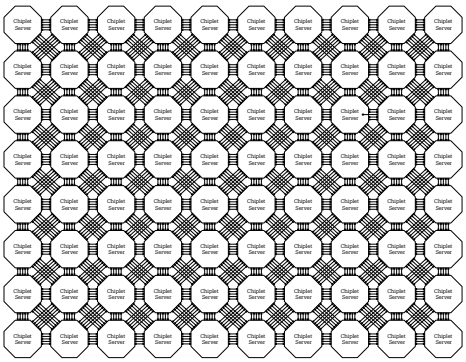
1.6 Minimum Triangle

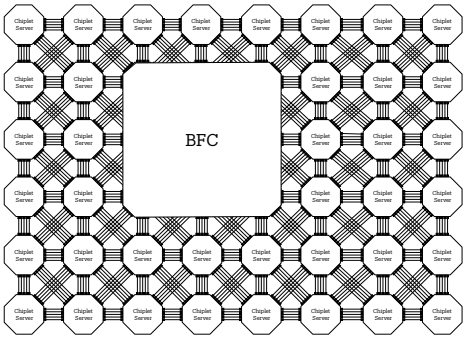
LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.



1.7 Big Flexible Chiplet

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.





1.8 Links are primary communication resources

The link between Alice and Bob is the resource to be shared. What goes over this link is controlled entirely by Alice and Bob.

The GAME the protocol plays ensures *fairness* (50/50 sharing of the link resources). Whatever is left over is offered as a resource to other cells further away.

This removes the need for POLICY – Each link has it's own policy, and can use as much of the bandwidth they need to perform their computation.

Whatever is left over can be offered (advertised) to the rest of the system as a utilization path.

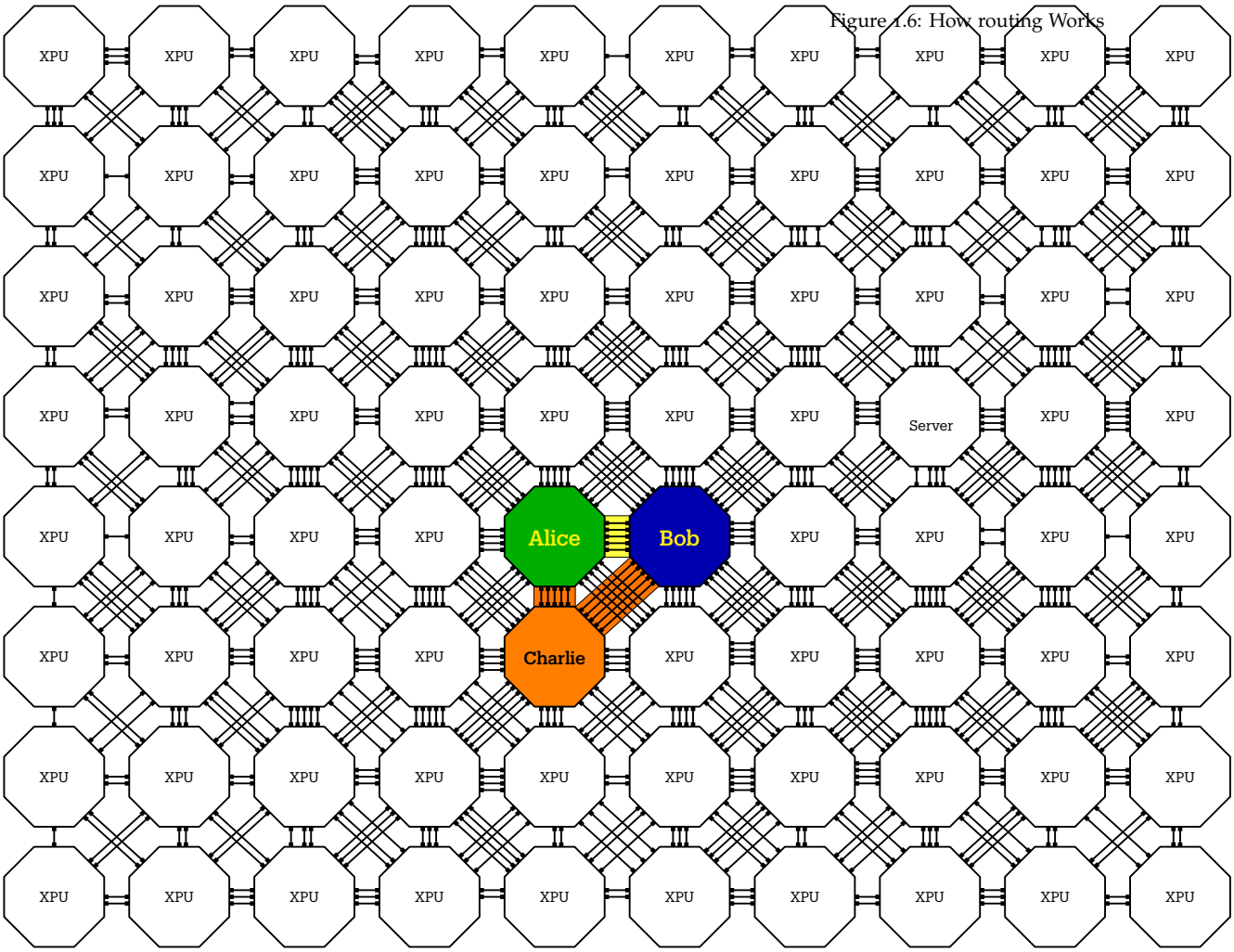


Figure 1.6: How routing Works

1.9 Colored Big Picture

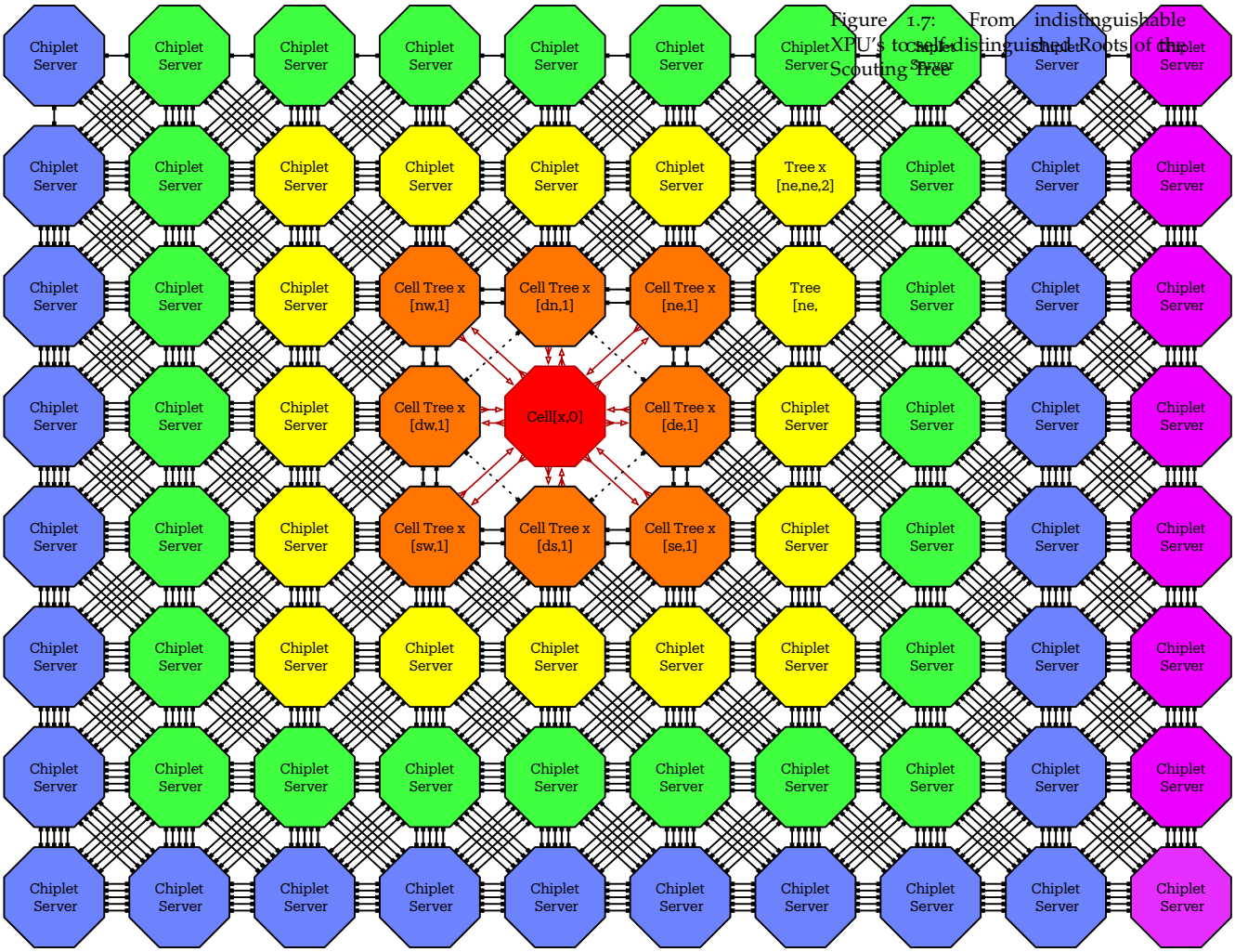


Figure 1.7: From indistinguishable XPU's to self distinguished Roots of the Scouting Tree

1.10 From Physical Cells (XPUs) to Logical Tiles

1.11 Consensus Tiles

1

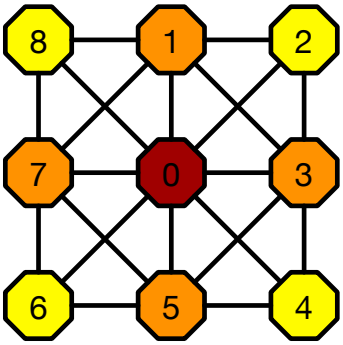


Figure 1.8: Consensus Tiles

1.12 Logical Tiles

Logical Tiles are built on top of physical tiles. They have the same 3x3 failure independence characteristics, but help define failure boundaries (the shared fate) of the system. These are discovered, not configured.

Because theese

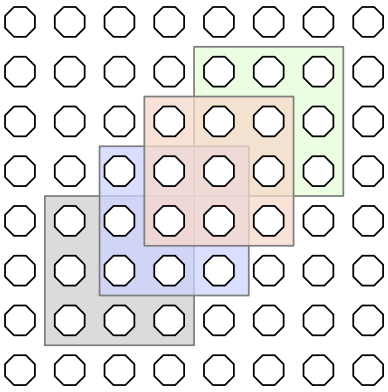


Figure 1.9: 3 x 3 tile

1.13 Tiles -2

1.14 Consensus Tiles 3

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

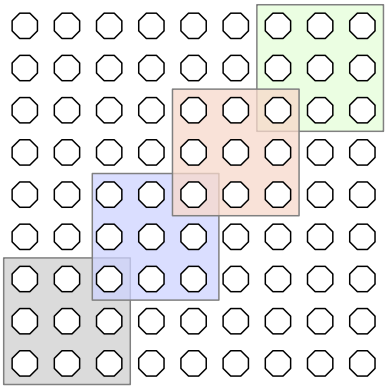


Figure 1.10: 9 x 9 Logical Mesh

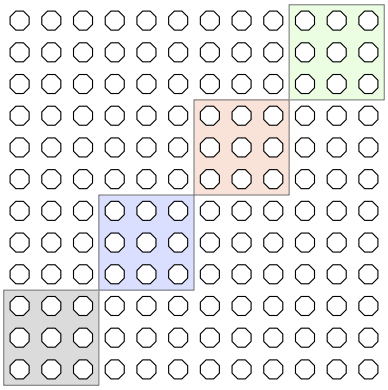


Figure 1.11: 9 x 9 Logical Mesh

1.15 Logical Overlays

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

1.15.1 Unfolded Clos

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

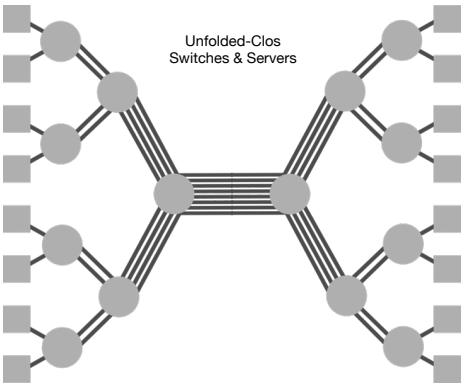


Figure 1.12: Unfolded Clos

1.15.2 Virtual Unfolded Clos

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

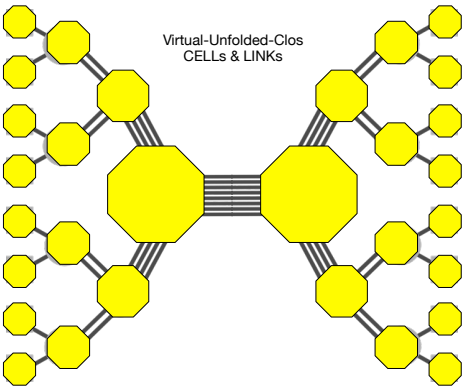


Figure 1.13: Virtual Unfolded Clos. Fish-Eye View of the LINK from any arbitrary LINK

1.16 Physical Overlays

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

1.17 Dragonfly

LOREM IPSUM It would be a mistake to assume conventional network concepts and terminology that you already know and love will remain unscathed in this project. While we have no intention of reinventing the wheel, some new concepts and terminology will be necessary in order to escape the quagmire of incrementalism of the last five decades.

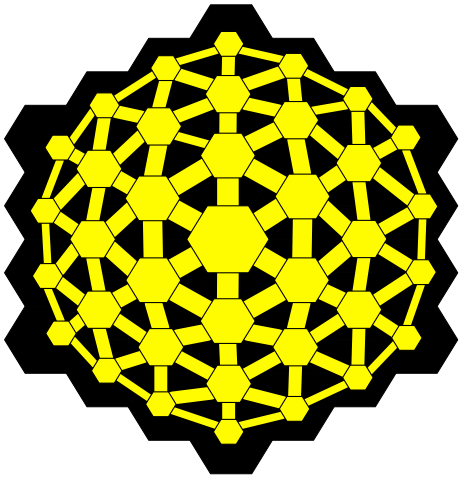


Figure 1.14: Fish-Eye Lens View of the network from an arbitrary Cell

1.18 Graph Aware Determinism

When treating the "Network" as an *opaque cloud*, it's easy to underestimate how varied network partitions when link failures are asymmetrical: A can see B, but B can't see A. In a 4 node setup, there are over 1295 potential partitions, and a flaky network can reproduce them all. From a distributed systems (event ordering in a cluster) as an availability equation, we can easily overestimate how reliable they are, by 3 orders of magnitude.

Link failures are invisible (hidden) in a Clos. They are 100% Visible to us in a local graph of *triangular* relationships.

And that's only the clean (binary) binary failures. Real system *flakey* connections are much worse.

1.18.1 Transactions need a coordinator?

The Æthernet protocol is designed to be exquisitely sensitive to packet loss and corruption We monitor, detect, diagnose link failures, and recover reversibly and automatically.

1.18.2 A Resilience Metric for Mesh Networks

1.18.3 Graph Laplacian and Algebraic Connectivity

The Graph Laplacian. For a simple, undirected graph $G = (V, E)$ with $n = |V|$ vertices, the *combinatorial Laplacian* matrix L is defined as

$$L \;=\; D \;-\; A,$$

where

- A is the $n \times n$ adjacency matrix, with $A_{ij} = 1$ if there is an edge between i and j, and 0 otherwise,
- D is the $n \times n$ diagonal *degree matrix*, whose diagonal entries are $D_{ii} = \deg(i)$.

The Laplacian L is central in spectral graph theory, encoding many connectivity properties of G.

Algebraic Connectivity (λ_2). Let the eigenvalues of L be ordered as

$$0 \;=\; \lambda_1 \;\leq\; \lambda_2 \;\leq\; \cdots \;\leq\; \lambda_n.$$

The second-smallest eigenvalue, λ_2, is the *algebraic connectivity* (or Fiedler value). It satisfies

- $\lambda_2 > 0$ if and only if G is connected,
- A larger λ_2 generally indicates stronger connectivity and a larger cut is required to disconnect G.

Thus, λ_2 is often seen as a "spectral" measure of how robustly G remains connected under certain disruptions.

1.18.4 Classical Connectivity Measures

Beyond λ_2, there are other classical measures:

1. **Edge Connectivity** $\lambda(G)$: The minimum number of edges whose removal disconnects G.
2. **Vertex Connectivity** $\kappa(G)$: The minimum number of vertices whose removal disconnects G.
3. **Expansion or Isoperimetric Constants**: Relate cut sizes to the cardinalities of sets being separated.

These capture *global* connectivity but may not reflect the incremental or adversarial removal of edges in a constrained-valency network.

1.18.5 Incremental Link Failures in Constrained-Valency Networks

In HPC or data-center systems (e.g. with IPUs or smartNICs), each node has limited valency (e.g. 8 ports), and edges can fail one by one. A single λ_2 value may not capture how partial or progressive failures degrade connectivity.

Why a single λ_2 may not suffice.

- λ_2 is a *one-shot* global measure. It does not directly model how connectivity degrades as edges fail in sequence.
- Some topologies might remain connected but experience severe bottlenecks after a few critical edges fail, which does not show up immediately in a single baseline λ_2.

1.18.6 Potential Approaches for a "Resilience Metric"

1.18.7 Spectral-Based Extensions

(a) Expected λ_2 under random failures. If edges fail independently with probability p, form a random subgraph G_p. One could define:

$$\mathbb{E}\big[\lambda_2(G_p)\big]$$

as a measure of *average* resilience. Larger expected algebraic connectivity implies better tolerance to random edge losses.

Worst-case sequence of λ_2 values. Define:

$$R(k) \;=\; \min_{\substack{F\subseteq E\\ |F|=k}} \lambda_2\big(G-F\big),$$

where $G - F$ is the graph with edges F removed. $R(k)$ measures the smallest λ_2 achievable *after k edge removals*. A graph is more resilient if $R(k)$ remains high for larger k. If $R(k)$ drops to 0, it indicates that with k removed edges, G can be disconnected.

1.18.8 Connectivity-Based Ideas

(a) k-Edge Connectivity Functions. Beyond the single value of $\lambda(G)$ (the edge connectivity), define

$$\phi(k) \;=\; \min_{\substack{F\subseteq E\\ |F|=k}} \Big(\text{size of the largest connected component of } G-F\Big).$$

If $\phi(k)$ remains large, it means removing any k edges fails to isolate more than a small fraction of nodes. This complements λ_2 by focusing on *component sizes*.

(b) Edge-disjoint path counts. Using Menger's Theorem, one can track the number of edge-disjoint paths between certain pairs of nodes. Higher numbers of disjoint paths generally imply more resilient connectivity.

1.18.9 Weighted or Dynamic Laplacian

A *dynamic* Laplacian $L(\mathbf{w})$ might assign weights w_e to edges. If an edge is fully failed, $w_e = 0$. Then one can track how $\lambda_2\big(L(\mathbf{w})\big)$ evolves as edges degrade from weight 1 to weight 0, either in random or adversarial patterns.

1.18.10 A Concrete Proposal

A practical "resilience function" might be:

$$R(k) \;=\; \min_{\substack{F\subseteq E\\ |F|=k}} \lambda_2\big(G-F\big),$$

where the minimum is taken over all subsets F of k edges. Then:

- $R(0) = \lambda_2(G)$ is the baseline algebraic connectivity.
- If $R(k) > 0$, the graph *cannot be disconnected* by removing any k edges.
- The rate at which $R(k)$ decreases with k reflects how fast the network's connectivity deteriorates under incremental failures.

1.18.11 Computational Observations

Exact computation of $R(k)$ can be expensive for large graphs because there are $\binom{|E|}{k}$ subsets. One may:

- Use *heuristics* or *approximation algorithms* to identify critical edges,
- Leverage *min-cut* or *max-flow* bounds to quickly estimate how easy it is to disconnect the graph,
- Perform *sampling* over subsets F if a random measure of resilience suffices.
- The *graph Laplacian* (and in particular λ_2) is a powerful spectral tool. It already gives a measure of connectivity robustness.

- For *incremental* or *adversarial* link failures, a single λ_2 value may not capture the full picture. A *function* $R(k)$ over subsets of size k can indicate how robustly the graph handles multiple simultaneous failures.
- In *constrained-valency* networks, certain edges are more critical, because each node has fewer possible alternate paths. Thus, a spectral-based metric that accounts for edge removals (like $R(k)$) can better reflect real-world vulnerability.
- Combined with classical connectivity measures (e.g. $\lambda(G)$, $\kappa(G)$), a Laplacian-based incremental approach provides a practical, mathematically grounded way to define and quantify *resilience* of a network topology.

1.19 Distributed Systems are Trees on Top of DAGs on Top of Graphs

From ./AE-Specifications-ETH/standalone/Trees-DAGs-Graphs.tex

This essay explores the layered graph-theoretic nature of distributed systems. At the lowest layer, physical and logical interconnects form undirected **graphs**. On top of this lie **DAGs** representing dependency, scheduling, and locking relationships. At the top, application-level consistency and authority are imposed via **trees** such as namespace hierarchies, leadership structures, and commit chains. We further examine how modern datacenters, populated by diverse xPUs (CPUs, GPUs, IPUs, DPUs), break the illusion of shared memory and necessitate protocol designs that exploit the native graph structure using mechanisms such as RDMA.

1.19.1 Graphs: The Physical and Logical Fabric

The physical topology of a datacenter is a graph: nodes represent compute units (CPUs, GPUs, IPUs, etc.) and edges represent communication links (Ethernet, NVLink, InfiniBand, etc.). These links may have diverse properties:

- Bandwidth and latency asymmetries
- Failures or congestion under load
- Scheduled or dynamic routing paths

Unlike the shared memory abstraction, these links form a non-uniform, fault-prone, and inherently asynchronous substrate. Real computation in modern datacenters occurs *on this graph*—not above it.

1.19.2 DAGs: Causality and Locking

On top of the physical graph lies a directed acyclic graph (DAG) representing **causality, scheduling, and consistency constraints**. DAGs arise in:

- **Transaction dependencies**: Operations must follow a directed order to preserve causality.
- **Lock hierarchies**: Preventing deadlock requires acquiring locks in a fixed topological order.
- **Build systems and job schedulers**: Tasks must respect dependencies.

Locking as a DAG

Databases employ lock hierarchies structured as DAGs to prevent circular waits. For example, the following might form a hierarchy:

1. Lock table
2. Then row

3. Then field

Each level narrows scope and follows a partial order. Enforcing that locks are acquired in topological order avoids cycles and hence deadlock.

1.19.3 Trees: Names, Commit Chains, and Leaders

At the top of the stack are trees. These structures are usually logical:

- **Namespace hierarchies**: e.g., file systems, DNS.
- **Leadership trees**: elected leaders per region, rack, or quorum.
- **Consensus and commits**: commit chains or logs form trees (or more precisely, forests with fork resolution).

These trees impose structure on the otherwise messy DAGs and graphs below, enabling:

- Easier authority delegation
- Fault domain containment
- Clear lineage and rollback support

1.19.4 Breaking the Shared Memory Illusion

Shared memory simplifies programming but breaks down in distributed xPU environments:

- Memory isn't uniformly addressable
- Coherence protocols are expensive or infeasible
- Latency variance introduces uncertainty in synchronization

RDMA: Network as Memory Bus

Remote Direct Memory Access (RDMA) partially restores shared memory semantics:

- Allows direct writes/reads between NICs with low latency
- Bypasses kernel and CPU involvement
- Supports zero-copy semantics for performance

But RDMA also forces a shift:

- You must think **asynchronously**
- Buffers must be explicitly registered and tracked
- Failures are explicit, not hidden

1.19.5 Exploiting the Graph: The Path Forward

To fully exploit xPU networks:

- Treat communication paths as first-class citizens
- Build coordination mechanisms that reflect graph topology
- Favor protocols that can adapt dynamically to congestion and partitioning

New system designs should:

1. Replace locking with message-passing wherever feasible

2. Encode application semantics in DAGs, not linear logs
3. Use explicit versioning and conflict resolution mechanisms

1.19.6 Conclusion

Distributed systems are not built on the abstraction of shared memory. They are constructed on a layered composition:

Graphs: physical connectivity

DAGs: causal and logical dependencies

Trees: naming, consensus, and leadership

The challenge of distributed systems is to harmonize these layers while respecting the physical realities of the system. To do so, we must leave behind illusions of synchrony and embrace graph-native programming models.

1.20 Mathematica as a Specification Language

From ./AE-Specifications-ETH/standalone/Mathematica-Spec-Language.tex

Exploring **formally executable specifications** in **datacenter architecture** touches the core of verifiability, reproducibility, and automation in modern systems.

Definition: Formally Executable Specification

A *formally executable specification* is:

- **Precise and unambiguous:** defined mathematically or via formal syntax.
- **Executable:** interpretable or simulatable.
- **Deterministically testable:** consistent output for consistent input.

In datacenter contexts, this implies that hardware, networking, storage, and compute orchestration policies are:

- Executable in simulation or emulation environments,
- Amenable to formal verification for correctness, safety, and performance.

Why It Matters in Datacenters

- **Correctness:** validate failover, routing, and policy enforcement.
- **Optimization:** evaluate configurations automatically.
- **Security:** prove isolation and policy compliance.
- **Confidence:** ensure safe deployment at scale.

Relevant Tools and Technologies

Domain	Tools
Network Architecture	P4, TLA+, NetKAT, Batfish
Storage Systems	TLA+, Ivy, Alloy, Z3 SMT
Orchestration	Kubernetes CRDs, Pulumi, OPA, Nomad
Formal Languages	TLA+, Coq, Lean, Dafny, Alloy
Execution	Mininet, NS-3, OMNeT++, QEMU, Verilator

Figure 1.15: Selected tools for formally modeling datacenter systems

Example: Rack-Aware Topology Specification

Imagine a model with:

- Compute nodes linked via ToR (Top-of-Rack) switches,
- Spine switches in a leaf-spine topology,
- Multi-path routing and QoS,
- VM placement and replication constraints.

The spec could:

- Simulate failures and load distribution,
- Detect routing loops or black holes,
- Evaluate bandwidth and latency guarantees,
- Prove placement constraints meet SLAs.

Vision: "Datacenter-as-Code" Verified

- High-level specs compile into deployable artifacts,
- Every change is property-checked and testable,
- Infrastructure becomes version-controlled logic, replacing spreadsheets and tribal lore.

Evaluating Mathematica for Executable Specification

Mathematica is a powerful computational platform. Its value depends on whether expressiveness or formal rigor is the priority.

1.20.1 Strengths of Mathematica

Limitations Compared to Formal Languages

- **Formal Semantics:** lacks type theory foundations (Coq, Lean).
- **Verification:** no native model checking or invariant proofs.
- **Concurrency:** no Lamport clocks or message-passing models.
- **Determinism:** pattern matching may be nondeterministic.
- **Refinement:** lacks formal spec-to-implementation pathways.

Suitable Use Cases

- Modeling tradeoffs in resource allocation,
- Simulating flows using graph theory,
- Prototyping performance constraints,
- Symbolic scheduling and placement logic,
- Writing executable whitepapers with computation and code.

Where It Falls Short

- Verifying safety and liveness across all states,
- Proving conformance or refinement,
- Modeling concurrency and faults rigorously,
- Integrating with RTL verification pipelines,
- Participating in formal proof communities.

Category	Capability
Symbolic Computation	Excellent for pipelines, graphs, latency models
Executability	Immediate execution and visualization
Expressiveness	Supports discrete, continuous, algebraic models
Rapid Prototyping	Rich in units, semantics, interactivity
Logic Tools	First-order logic, SAT solving, quantifiers
Documentation	Notebooks are self-contained and reproducible

Figure 1.16: Strengths of Mathematica in system modeling

1.20.2 Summary Judgment

Mathematica is:

- **Excellent** for exploratory, high-level modeling and simulation,
- **Weak** for formal verification, proofs, and correctness guarantees,
- **Valuable** as a literate architecture spec tool, but not a full formal methods platform.

1.20.3 Appendix A: TLA+ Model – Rack-Aware Topology

RackAwareSpec.tla

```
---------------------------- MODULE RackAwareSpec ----------------------------
EXTENDS Naturals, Sequences

CONSTANTS Racks, Nodes, Links

VARIABLES rackStatus, linkStatus, trafficMap

(*--algorithm RackAware
variables rackStatus \in [Racks -> {"up", "down"}],
          linkStatus \in [Links -> {"up", "down"}],
          trafficMap \in [Nodes -> [Nodes -> {"ok", "blocked", "reroute"}]];

define
  IsAvailable(n) == \E r \in Racks: rackStatus[r] = "up" /\ n \in Nodes /\ TRUE
end define;

begin
  Init ==
    /\ \A r \in Racks: rackStatus[r] = "up"
    /\ \A l \in Links: linkStatus[l] = "up"
    /\ \A s, d \in Nodes: trafficMap[s][d] = "ok";

  Next ==
    \E r \in Racks:
```

```
      /\ rackStatus[r] = "up"
      /\ rackStatus' = [rackStatus EXCEPT ![r] = "down"]
      /\ UNCHANGED <<linkStatus, trafficMap>>
  \/ \E l \in Links:
      /\ linkStatus[l] = "up"
      /\ linkStatus' = [linkStatus EXCEPT ![l] = "down"]
      /\ UNCHANGED <<rackStatus, trafficMap>>;

end algorithm;
*****************************************************************************
```

Appendix B: Alloy Model – Storage Placement Constraints

StorageModel.als

```
module StorageModel

abstract sig Rack {}
sig Node {
  hostRack: one Rack,
  stores: set Volume
}
sig Volume {
  replicas: some Node
}

fact ReplicationFactor {
  all v: Volume | #v.replicas = 3
}

fact NoReplicaOnSameRack {
  all v: Volume |
    all disj n1, n2: v.replicas |
      n1.hostRack != n2.hostRack
}

pred ShowExample {}

run ShowExample for 3 Rack, 6 Node, 2 Volume
```

1.21 CLOS

From ./AE-Specifications-ETH/sections/Clos.tex

1.21.1 Topology set-up (same 200 servers)

Clos fabric 20 racks, each with 10 servers. Every server now owns **four** NIC ports, all cabled to its top-of-rack (ToR) switch, giving $200 \times 4 = 800$ host links. Each ToR uplinks once to *each* of the four spine switches. A pair of core switches terminates the third level.

8-*regular mesh* The same 200 servers, each equipped with **eight** NIC ports wired into an undirected 8-regular graph. The link count is

$$L_{\text{mesh}} = \frac{200 \times 8}{2} = 800,$$

exactly matching the number of host cables in the Clos system.

1.21.2 Cable inventory

Link class	Clos count	Mesh count
Server–ToR	800	–
ToR–Spine	80	–
Spine–Core	8	–
Server–Server (mesh)	–	800
Total physical links	888	800

Table 1.1: Cable counts after upgrading each Clos server to four NIC ports. The mesh uses the same 800 cables as data-carrying edges, eliminating the 88 upward cables and the entire switch hierarchy above the racks.

1.21.3 Failure-mode magnitude

Treat each link as an independent four-state component $\Sigma = \{00, 01, 10, 11\}$. The number of distinct network states is 4^L, so the number of failure patterns is $4^L - 1$.

$$\log_{10}(4^L) \;=\; 0.60206\, L.$$

Topology	L	Failure modes (order of magnitude)
Clos (4 ports)	888	$\sim 10^{535}$
Mesh (8-regular)	800	$\sim 10^{482}$

Although the Clos now contains more cables, inter-rack traffic is still forced through only 88 uplinks. The mesh distributes both traffic and failure risk across *all* 800 cables.

1.21.4 Path-diversity impact

Clos

- A rack-to-rack flow traverses six vertical hops (Server $\rightarrow$ ToR $\rightarrow$ Spine $\rightarrow$ Core and back down).
- End-to-end success probability is roughly p^6, where p is the per-link health probability.

Mesh

- Every server has eight one-hop neighbours; many multi-hop detours remain even after several failures.
- Loss of one cable only lowers a single server's degree from 8 to 7; global reachability is unaffected.

1.21.5 Key observations

1. **Vertical choke-points remain**. Extra NICs in the Clos enlarge rack bandwidth but do not remove the dependence on 88 spine–core cables.
2. **Risk distribution**. The mesh spreads failure impact evenly; the Clos still concentrates risk in its upper layers.
3. **Equipment footprint**. The mesh eliminates 30 switches (20 ToRs, 4 spines, 2 cores), trading them for denser lateral cabling.
4. **Graceful degradation**. Clos bisection bandwidth falls in 12.5% or 5% steps; mesh capacity decays proportionally to failed cables, with no cliff.