# 1. Architecture

## 1.1 Back to Back Shannon Channels

From ./AE-Specifications-ETH/Shannon.tex

### 1.1.1 Two independent Metcalfe Channels (Max flow, no Interaction)

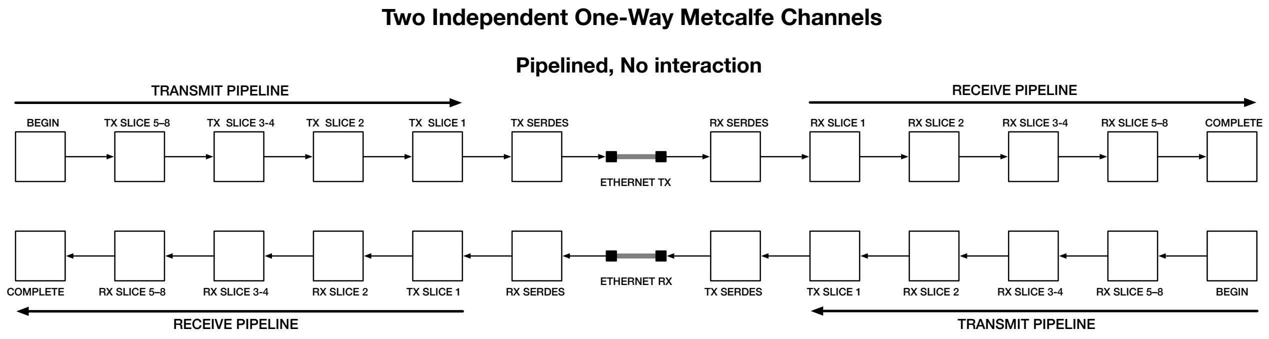


Figure 1.1: Two Independent Metcalfe Channels

### 1.1.2 Internal (SACK) Feedback on last slice

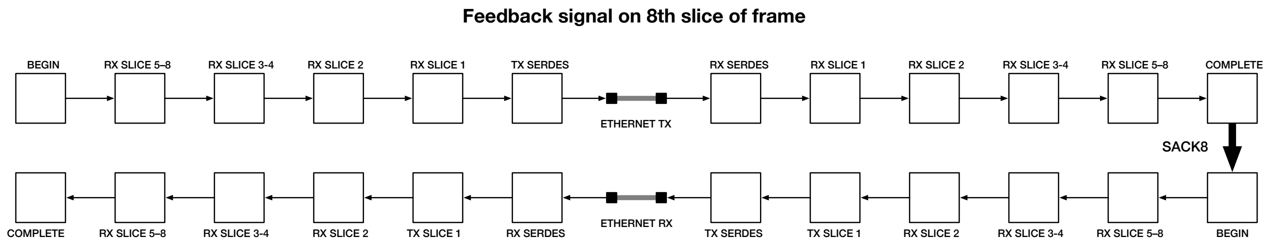


Figure 1.2: Feedback signal on slice 8

See SACK description by Sahas **?**

### 1.1.3 Internal (SACK) Feedback on first slice

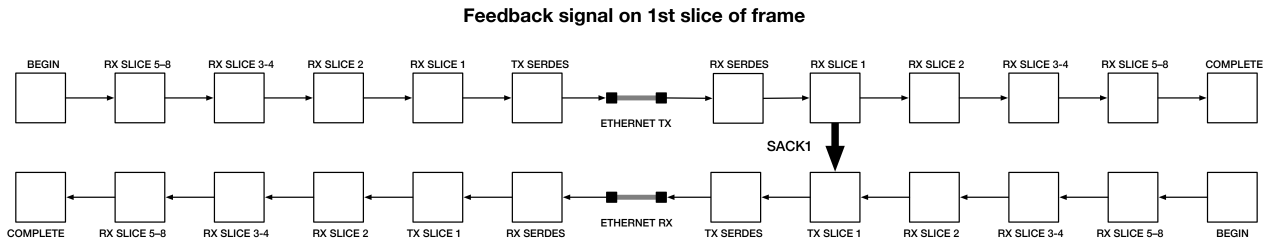


Figure 1.3: Feedback signal on slice 1

## 1.2 Architectural Framework: Four Shannon-like Levels

In the proposal for subdividing a 64-byte packet into 8-byte slices, we introduce partial acknowledgments (SACKs) at four (decrementing) boundaries (`11`, `10`, `01`, `00`). Each of these points reveals an incrementally deeper level of the receiver's certainty about the data, the hardware, and the appropriate next step in the protocol. We can interpret this progressive certainty in terms of four conceptual *layers* reminiscent of Shannon's *information* theory, but extended to address knowledge, semantics, and understanding. This layering describes how a receiver (e.g., the SmartNIC) transitions from raw incoming bits to meaningful messages that can be handed off to the host processor.

Back-to-Back (B2B) Shannon Channels are "Perfect Information Feedback" (PIF) as senders see their own transmitted packet returning back from the receiver and thus can detect channel errors. Thus making CRCs Checksums, Parity and FEC unnecessary. Similar to "Perfect Information Feedback" in: Norm Abramson, "Packet switching with satellites," NCC, 1973

### 1.2.1 Layer 1: Information (Surprisal)

At the first level, information refers to the direct "yes/no" answer to a question of interest: the arrival or non-arrival of bits, which Shannon famously treated as the surprisal of a received symbol. At the SACK `00` boundary, when the receiver detects the first 8-byte slice without error, it learns that the link is alive and that the data matches expectations (i.e., no immediate mismatch). This is pure information because it distinguishes the event "we did receive slice #1 correctly" from "we did not." The mutual information gained here confirms a working cable and a functional SerDes.

At this early stage, the question posed is binary: "Did the hardware see valid bits?" The surprisal is that valid bits were received, as opposed to no signal or corrupted data.

### 1.2.2 Layer 2: Knowledge (Captured Information)

The second layer, knowledge, arises when the raw bits are stored or captured in a meaningful structure. This could be as simple as a recognized slice stored in buffer memory or a pipeline register. By the time the second slice arrives, the receiver has captured more bits—16 bytes in total—and placed them into NIC-internal registers. It can then perform further checks, such as alignment, partial CRC, or checking for expected header fields. The SACK `01` confirms that the hardware not only saw valid bits but also placed them in the correct buffer location.

At this point, the system has a partial understanding of the data. It knows that the 16 bytes are recognized and safely stored, awaiting deeper logic to interpret them.

### 1.2.3 Layer 3: Semantics (Meaning)

The third layer, semantics, involves the system deciding what the bits mean in terms of subsequent action. This layer determines which state machine or processing path is relevant for the given data. At the SACK `10` boundary, after 32 bytes have been received, the NIC has gathered enough information to partially decode the data. For example, it might be able to determine which protocol or message type is indicated. The NIC can confirm that buffer slots or ring descriptors are available and that the correct state machine is loaded (e.g., state machine A for small control frames, or state machine B for streaming payloads).

Once the NIC signals SACK `10`, the sender learns that the hardware has found the data coherent enough to continue. The semantics are recognized sufficiently to proceed without hazard. The receiver has now moved from simply knowing the bits are correct (Layers 1 and 2) to understanding how to proceed and which internal resources or state machines to activate.

### 1.2.4 Layer 4: Understanding (Syntax)

The final layer, understanding, refers to the recognition that the message fits into a finite set of concepts or message types that the NIC accepts. This implies that the message has a correct syntax recognized by the hardware. At the SACK `11` boundary, which occurs when slices 5–8 arrive, the full 64 bytes have been received and match a legitimate frame or message layout. The NIC is now ready to push the message onto the PCIe bus or an internal ring buffer for the host processor.

At this stage, the NIC has a full understanding of the message, knowing exactly how to finalize the packet, classify it, and pass it upstream for higher-level processing. No further layer-2 repairs are needed, and the message is ready for the next step in the protocol.

## 1.3 Bidirectional Shannon Channels

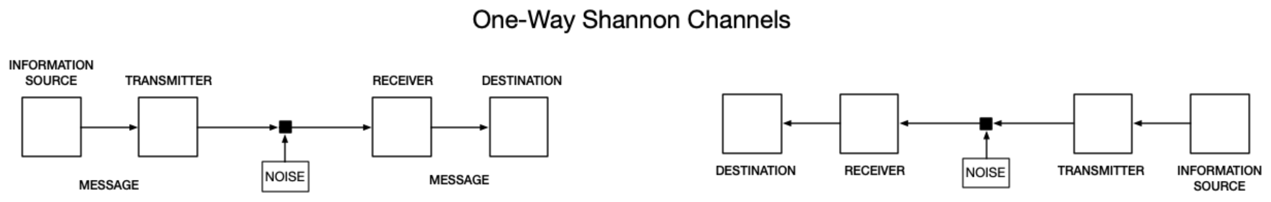


Figure 1.4: Shannon One-Way Channels.

Shannon Channels are normally shown in one *direction* of flow – from Information source to Information Destination. Here we exploit two-way communication (signaling) Back to Back Channels with immediate (slice by slice) feedback. Then the equations tell us something interesting about the *symmetry* of set reconciliation on both sides of the link.

With back-to-back Shannon Channels, with (immediate) slice by slice feedback, we get Perfect Information Transfer (PIT) **?**. We can therefore dispense with Checksums, CRC's, FEC or even Parity, because the failure modes these EDC and ECC codes address are already covered by PIT. This has two advantages:

- The elimination of spatial redundancy on the wire makes the packets shorter
- The need to calculate increasingly complex codes reduces computation and energy dissipation on the link

Redundancy is a poor crutch when assumptions about uniform probability distributions are violated (which they almost always are in practice).

### 1.3.1 Metcalfe Half-Duplex

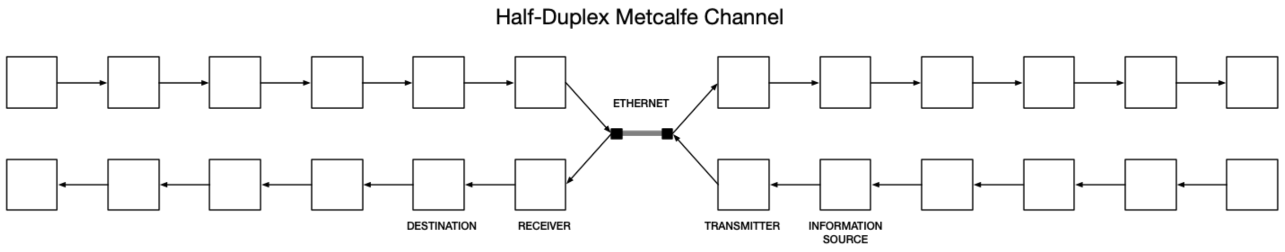


Figure 1.5: The original Metcalfe + Boggs Ethernet was a bus. A long cable where 'stations' were TAPs on the bus. This meant that each station had to both listen, and transmit from teach tap. In this figure we show two *indepen-den*[illegible]eams [illegible]ckets[illegible]g in [illegible] site[illegible]ction[illegible]wardp[illegible]gatio[illegible] backpropagation) through a single half-duplex link

### 1.3.2 Metcalfe Half-Duplex Channel

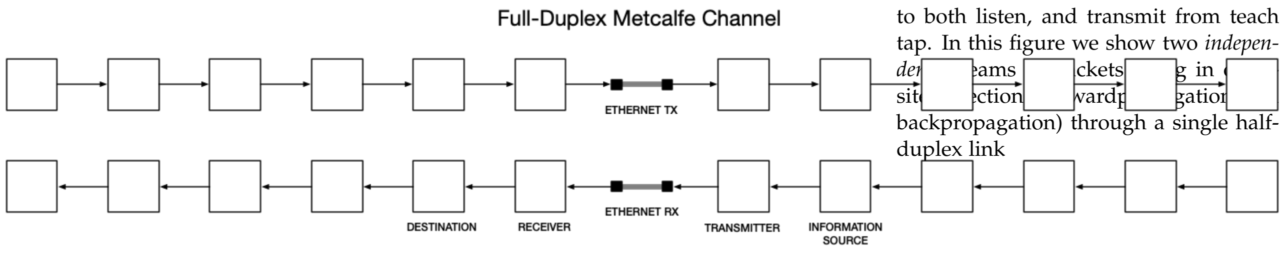


Figure 1.6: Modern Ethernet Links are bidirectional; two sub-channels: one for transmit, one for receive

### 1.3.3 Full-Duplex Bi-pipelined Shannon-Metcalfe Channel

The figure above is a simple, formally verifiable, mathematical description, from API to bits on the wire (Shannon channel).

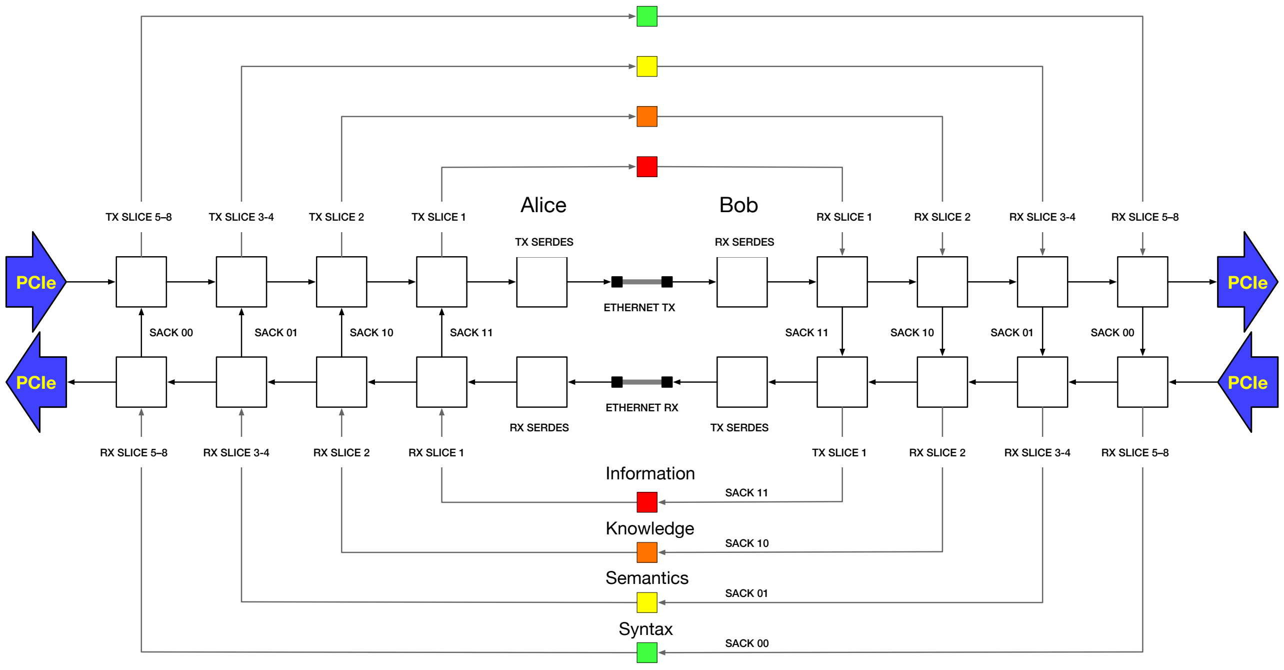


Figure 1.7: Complete model: Bi-pipelined full duplex exchange of Æthernet frames. Complete with internal "Slice ACKnowledges" (SACKs) – sequenced with increasing *common knowledge* depth inside SerDes/FPGA

## 1.4 Short-Range ANT (Local) Scouting

./AE-Specifications-ETH/standalone/Scouting.tex

Once a link has been established, they are recorded in the local `knowledge` of the cell, and used as a baseline for future algorithmic and policy decisions.

Immediately after establishing a reliable connection, `CELL`s may emit `ANT-SCOUTS` to explore their local environment. These are ANT's, which obey an initial source routing algorithm, but when encountering a failed or disconnected port in another cell, respond with either clockwise, anticlockwse packet forwarding, which keeps the scout local, or *random*, with a hopcount limit, which allows exploration further afield.

See ANT Specification for details

## 1.5 Long-Range BEE (Global) Scouting

`CELL`s may also emit `BEE-SCOUTS` to explore the extremities of their environment. These are BEE's which obey only one rule: proceed in the same direction as the radial port. BEE's emitted on the *n* port may only go *n*. BEEs emitted on the *se* port may only go *se*. Until they encounter a disconnected port, whereupon they execute a return path algorithm, accumulating information at each `CELL` and returning it to the root.

See BEE specification for details

## 1.6 ANT Specification: Triangle Packet Clocks in $3 \times 3$ Tiles

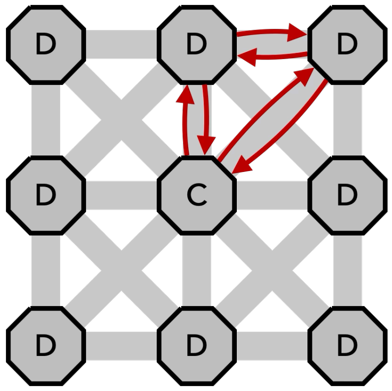


Figure 1.8: Race-Free Triangle Token

Packet clocks are initiated by the coordinator, on any of it's active links. The ANT (source routing) algorithm goes out on any port, and are programmed to turn left or right at the first available active port. The convention is turning right makes it go clockwise, and turning left makes it go antilockwise, but this is an artificial distinction. As with real ants, they can get lost, and never find their way back to the nest, and they die (or return to the nest by inverting their source routing paths). This "limited range", is part of the Security mechanism.

## 1.7 ANT Specification: Race-Free Packet Clocks in $3 \times 3$ Tiles

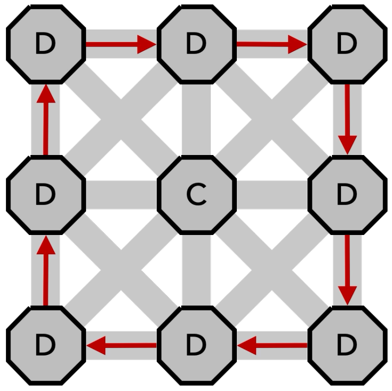
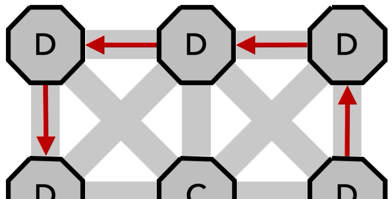


Figure 1.9: Square Race-Free 1-hop Clock

Once the cell has discovered it's local environment, it may establish packet clocks. These are ANTs, which go out with a pre-defined pattern, to return events the cell on a *periodic* basis. Because there is no *background* of time, this system will create events, which are guaranteed to occur without race conditions, but will catastrophically fail if there are an broken links around the circuit.



Packets clocks may be initiated around the closest (one-hop) cell tiles, next closest (two-hop) tiles, furthest (three-hop) tiles. Atomicity

### 1.7.1 ANT Specification Building a Compass Clock

8 physical ports per cell. Inactive ports may be:

- Failed (out of service)
- Standby, ready to go
- Off, saving energy

### 1.7.2 ANT Specification: Counter Circulating Race-Free ANTS

C (Carol, Charlie, Coordinator, Chief) may initiate clockwise, counter-clockwise, and/or both at once. Each is exploring the health of the connectivity local to the center cell. This is what ANT Algorithms (source routed, or random) tokens.

Ports at edge of mesh connected back to same cell on a different port to traverse routing table 2nd time to create virtual cut-through torus.

If the ANT gets blocked, and either runs out of hopcout resource, it does 'reverse path forwarding' back to the C CELL. and reports what it finds. It can either carry all its state in the packet, or (ror BEEs), clean up on its way and erase its footprints in the CELLs it visited.

### 1.7.3 7 x 7 Nodes Packet Clock

## 1.8 Beyond Packet Clocks

Packet clocks don't scale (they are not intended to). Instead, they provide circulating logical loops **?**. The local system policy will establish the radius limit for local exploration. Everything beyond that is in the domain of the BEE scouts .

Packet clocks can circulate at any physical hop distance. The one-hop agents are described above. The two figures on the right show an example of an ANT which goes two hops, or three hops, before the ANT turns left or right. This give a CELL the opportunity to explore larger hop distances from the coordinator

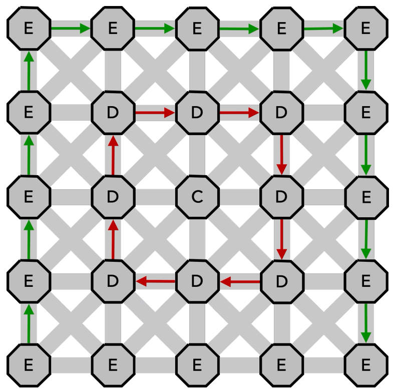


Figure 1.12: Green Packet Rateless Clock

## 1.9 Packet Clocks in Larger Tiles

### 1.9.1 BEE scouts

BEE Scouts explore the boundaries of their environment. The are emitted by the Coordinator, and travel as far as they can in ONE direction, `{dn,ne,de, se, ds, sw, dw, nw}`, and then *return on the reciprocal path (Compass-Point vector direction)* to inform the hive (root) what

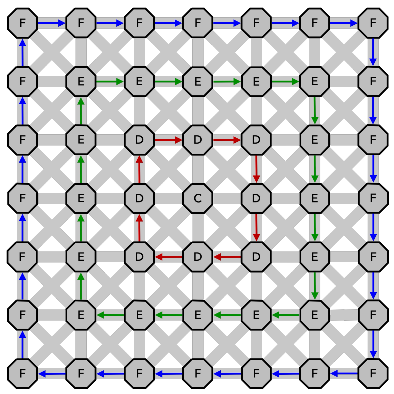


Figure 1.13: 3rd-hop circular packet clocks. Blue Links Complete

they discovered, so the root can build it's model of the topology, and Edge resources to perform their function.

### 1.9.2 N x N Nodes Packet Search Rays (BEEs)

BEEs are radial distance scouting agents. Single packets that go in only one direction, and when they reach the end (extremities of the Cellular interconnect) they execute a reverse path forwarding algorithm, collecting knowledge on their way, delivering this knowledge back to the root, whose agent uses the returned information to build it's model of it's topology and available resources to offer 'services' to the applications.

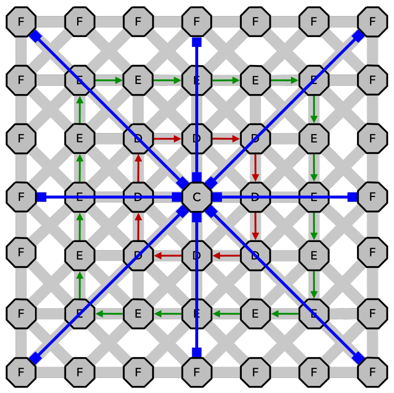


Figure 1.14: BEE Algorithms explore beyond ANT algorithms

These don't have to be square, or rectangular. BEE algorithms work on any arbitrary Topology.

Radial (Ray) source-routed scouts have two parameters (a) which port they go out on, and continue indefinitely until they reach a boundary (or exhaust their hop count resource). And then they return along exactly the same path, accumulating knowledge of the CELLS on their way (e.g. properties of the cell, do they have a CPU, a GPU, an IPU, or QPU?). Most Bees make it back home to the nest (C) but it is also possible for a failure to occur between the outbound BEE and the homebound BEE. In which case the packet try's to make it's way to 'Lost and Found', the control structures identified by the Coordinator to provide GEV notification of failures. Lost and Found is most likely to be discovered by the one or more of the BEEs. Edge nodes (on the corners of the interconnect), will always be able to 'find' Lost and Found (and other external control paths controlled by monitoring or configuration LOGICAL Administrator Agentss ) with a 'due north or south' `dn,ds`, 'due-east or west' `de,dw` BEE Scout.

## 1.10 Local decisions and emergent global organization

- Scouting/Discovery Phase: Biologically inspired methods (e.g., ant-colony-inspired or pheromone-based algorithms) often employ "scout" packets or "explorer" agents that roam the network. These scouts collect local congestion or path-quality information and deposit some form of "trail" (akin to pheromones).
- Emergent Routing Table Updates: Each router or switch updates local routing information (sometimes called a local "pheromone table"). Over time, paths that prove consistently "good" get reinforced; less efficient paths fade. This local, probabilistic approach can converge on globally efficient routes with no central coordination.

### 1.10.1 Relevance to On-Chip or 2D Mesh Topologies

- Local Compass Directions: In a regular mesh (e.g., 2D grid) or torus, each router has up to 4 (N, E, S, W) or 8 ports (adding NW, NE, SW, SE). A biologically inspired algorithm can treat each output port as a possible "direction of travel."
- Natural Fit for Scouting: The local directional structure matches how "ants" or "foraging agents" might look around in each direction, choosing a route based on local pheromone levels (akin to local congestion or link utilization).

Thus, the scouting/discovery mechanism is all about gathering local "pathworthiness" data and then directing future traffic toward better routes—exactly how a local compass-based system can easily be integrated.

### 1.10.2 Bufferless (Hot-Potato) Routing

- No Packet Buffers (or Very Limited Buffers): In a bufferless architecture, every router typically either immediately forwards or deflects each incoming packet. Packets cannot wait in large queues when an output port is congested.
- Hot-Potato / Deflection Character: When the preferred output port is unavailable, the packet is sent out of a different (less ideal) port—"hot-potato" style—rather than being buffered.

### 1.10.3 Connection with Biologically Inspired Approaches

- Continuous Movement: Biologically inspired scouts are already designed to wander and discover; in a bufferless system, "wandering" (via deflections) is also central. This synergy means a router can apply a heuristic (like a pheromone table) to pick the "best available port" quickly, but if that port is busy, the packet must choose an alternate direction.
- Adaptive Reinforcement Over Time: In a bufferless design, a packet cannot linger while waiting for the optimal output. However, local "pheromone" or "congestion" metrics can still help route the majority of packets down better ports more often. Over time, high-traffic edges might become less appealing, guiding packets to less-congested directions.

### 1.10.4 Deflection Routing

- Forced Misrouting / Deflection: If the desired or minimal-distance output port cannot be taken (due to contention), the router picks another output. The packet may travel away from its ultimate destination (a "deflection"), but eventually, it should be re-routed back on track.

- Common in Low- or No-Buffer Architectures: Deflection routing is one way to handle resource contention when buffer space is unavailable.

### 1.10.5 Tying It Back to the Compass Ports (N, E, S, W, NW, NE, SW, SE)

- Local Prioritization: In an 8-port (or 4-port) router, one can define a strict or heuristic priority among the directions. For example, a packet traveling generally "north-east" might prefer the N or E port if free; if both are busy, it might deflect NE, or in the worst case, deflect NW or SE.
- Biologically Inspired Ranking: The "pheromone" concept can be used to rank the output directions. The highest "pheromone" port is tried first, then so on down the rank. This effectively merges a local heuristic (pheromone) with forced deflection for whichever ports remain free.

In practice, such a scheme allows packets to "scout" and reinforce certain directions while still ensuring that they never have to wait for a blocked port.

### 1.10.6 Example Flow in an 8-Port Router

1. Receive a Packet coming in from, say, the south port.
2. Look Up Destination (or partial coordinate heading). For instance, the packet is trying to reach a node in the north-east region, so N or E might be favored.
3. Check Local "Pheromone" or Routing Table: Suppose the local pheromone table says port NE is the best guess based on past traffic patterns.
4. If NE Port Is Free: Forward the packet NE.
5. If NE Port Is Busy: Check next best local direction (N, E, or NW/SE fallback).
6. If All Preferred Ports Are Busy: Packet is deflected to any open port (could be even SW in the worst case).
7. Local Table Update: The router sees how that choice ended up affecting the packet (if it eventually left the region quickly or ended up in a congested area). Over time, these experiences feed back into local pheromone levels.

Despite the forced misrouting (deflections), the biologically inspired feedback approach often keeps net throughput healthy and tries to avoid systematic congestion "hot spots."

### 1.10.7 Connection with the Literature

1. 1. Hot-Potato Routing (Deflection Routing):

- Baran, P. (1962). On Distributed Communications Networks. IEEE Transactions on Communications. (Early ideas of "hot-potato" and distributed routing).
- Dally, W., & Towles, B. (2004). Principles and Practices of Interconnection Networks. (Excellent overview of deflection routing in modern network design).

2. 2. Biologically Inspired / Ant-Based Routing:
   - Di Caro, G. A., & Dorigo, M. (1997). AntNet: Distributed stigmergetic control for communications networks. Journal of Artificial Intelligence Research.
   - Schoonderwoerd, R., Holland, O., Bruten, J., & Rothkrantz, L. (1996). Ant-based load balancing in telecommunications networks. Adaptive Behavior.
3. 3. Network-on-Chip with Deflection/Bufferless Approaches:
   - Moraes, F. et al. (2004). A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping. SBCCI.
   - Fallin, C., et al. (2012). CHIPPER: A Low-Complexity Bufferless Deflection Router. HPCA.

These resources flesh out how bufferless or deflection routing is implemented (especially in on-chip contexts) and how biologically inspired heuristics can be adapted to local, minimal-knowledge scouting decisions.

### 1.10.8 Concluding Remarks

- Shared Tenets: Both biologically inspired scouting and deflection-based, bufferless routing rest on local decision making. In biologically inspired schemes, scouting packets "discover" or "reinforce" certain paths. In deflection routing, each router makes a quick (local) decision when a preferred port is blocked, forcing packets to keep moving.
- Complementary Mechanics: Because biologically inspired "pheromone" updates naturally reflect congestion and path usage, they integrate well with a bufferless or deflection style—turning forced misroutes into valuable "exploration" signals that feed back into local heuristics.
- Directional Routing: The presence of N, S, E, W (plus diagonals) simply defines how many possible local moves each node (router) can attempt. In 2D meshes or tori, these directions make for a convenient coordinate system that parallels how ants (or other scouts) might sense local gradients or pheromone intensities in each of eight compass directions.

Overall, if we combine a scouting mechanism (to adaptively find neighbors and good routes) with deflection routing (to handle buffer constraints or high contention), we get a dynamic, emergent routing system in which packets flow continuously and local updates shape global traffic patterns in a self-organizing fashion.

All this happens without the need for Source/Destination Addresses, which present severe security problems by exposing the “identity" of nodes making them vulnerable to attack.

## 1.11 A Blueprint for Æthernet Protocol

## 1.12 Foundations

We begin with Bartlett, Scantlebury and Wilkinson's (**?**) Alternating Bit Protocol (ABP).

From ./AE-Specifications/sections/BSW.tex

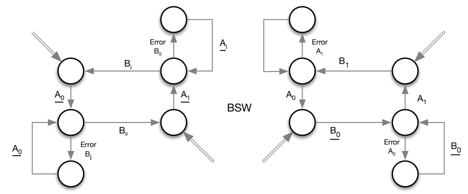


Figure 1.15: Aternating Bit Protocol(ABP)

Alternating messages are implemented as a single snake (a longitudinal set of bits traveling through the wire and FPGA's). Wrapping its way through the SerDes Registers of Alice and Bob on both ends.

First imagine a zero length wire connecting the chiplets for Alice an Bob (they are as adjacent as they can be on a module or motherboard). Two SerDes channels are directly connected.

Where in BSW[1], the edges of the automata are labeled with the 'alternation' bit, Lynch expands the single bit to multiple bits.

[1] A note on reliable full-duplex transmission over half-duplex links

A single bit alternates, but it does not have a direction.

**Alternating 2-Bit Protocol (from Lynch):**

Two bits at least have a direction in their evolution through their set.

`{00 01 11 11}`

OR

`{11 10 01 00}`

Where the next item establishes the direction of evolution of the set.

We use Ternary logic `-1,0, +1` in our model of the protocol.

0 simply means 'equilibrium', which gives us the +1 and -1 complementary states that are expected of Ternary Logic. But there is a mathematical subtlety: we need both two versions of Zero, one approaching from the negative side, and one approaching from the positive side.

This provides a mechanism for *forward-propagation* and , of the state.

### 1.12.1 Proof of Fallibility

> "the alternating validation bit becoming the additionally the *alternation* bit for message transmission in one direction, while the alternation bot for the reverse directions serves additionally as a validation bit" SBW **?**

Lynch's scheme is constructed from independent simplex procedures

## 1.13 Alternating Causality

QUESTION: What is the difference between reliable and fallible?

## 1.14 Framing the Vocabulary

In 1967 *Bartlett, Scantlebury & Wilkinson* (BSW) sketched the *alternating-bit protocol* (ABP): add one history bit to every frame, wait for an ACK that echoes it, and retransmit until the right ACK appears. ABP wraps an *unreliable* medium and presents a service that looks reliable—even *infallible* in steady state.

Nancy Lynch later formalised these ideas with the *I/O-automaton*: a fallible channel is one whose execution may deviate from the specification, subject only to fairness.

| Term | Meaning |
|---|---|
| Unreliable | Frames may be lost. |
| Fallible | Channel may violate any promise (drop, duplicate, reorder, corrupt). |
| Reliable | No drops in steady state; recovery still required. |
| Infallible | No violations ever; no recovery logic needed above the link. |

Figure 1.16: Taxonomy of link qualities.

## 1.15 Where Classical Ethernet Fits

Original 10 Mb/s Ethernet (and most "best-effort" variants since) offers a CRC to *detect* corruption but no link-level retransmission. Frames can be dropped by congestion, policing, or topology loops. Hence raw Ethernet is both *unreliable and fallible*; higher layers—typically TCP—supply ABP-like recovery.

## 1.16 How InfiniBand Raises the Game

InfiniBand *embeds* ABP into silicon:

- Per-hop *credit flow control* makes buffer overflow almost impossible.
- Link-level CRC plus optional *link retransmission* retries any corrupted frame.
- Reliable Connection and Reliable Datagram queue pairs carry *ACK-/NACK* sequence numbers end-to-end, guaranteeing exactly-once, in-order delivery across multi-switch fabrics.

To software, the fabric appears nearly *infallible*; drops are rare and localized.

## 1.17 Reliable vs. Infallible, Unreliable vs, Fallible

Table 1.16 highlights the nuance. Priority Flow Control (PFC) can render an Ethernet link *loss-less* in steady state, but deadlock, misconfiguration, or burst congestion can still drop frames. Such a link is "reliable yet fallible." Infiniband's credit + retransmit pipeline, by contrast shifts real-world operation toward "reliable and almost infallible."

## 1.18 Why Ethernet Still Struggles

1. **Retrofitting**: inserting link retransmission into the IEEE 802 stack breaks long-standing timing and compatibility assumptions.

2. **Congestion domain**: shallow switch queues and ECMP paths leave more surfaces for loss than InfiniBand's strict hop-by-hop credits.
3. **Layering philosophy**: because TCP "already" ensures delivery, many operators accept occasional loss rather than pay silicon cost for hardware recovery.

## 1.19 Lessons from BSW and Lynch

- *Make reliability local*. ABP attaches one bit; InfiniBand embeds a few more. End-to-end recovery alone expands the failure scope.
- *Fail fast*. InfiniBand retransmits on explicit NACK within microseconds; Ethernet traditionally converts a microsecond drop into a millisecond TCP timeout.
- *Separate reliability from recovery*. Even a reliable link needs a failsafe plan; design that plan explicitly.

## 1.20 Conclusion

**CONCLUSION:**

BSW showed that a single alternating bit can tame a capricious wire; Lynch supplied the proof rules. InfiniBand adopted both insights in hardware and delivers a fabric whose normal behavior feels *infallible*. Classic Ethernet remains best-effort—unreliable and fallible—and relies on upper layers for recovery. Bridging that gap means absorbing more of the ABP playbook at the link: credits, link-level retransmission, and tight ACK/NACK loops that shrink recovery from milliseconds to microseconds.

## 1.21 Forward and Backpropagation Through the Lens of the Two-State Vector Formalism

The *Two-State Vector Formalism* (TSVF), developed by Aharonov and collaborators, describes a quantum system using both a forward-evolving state vector from the past and a backward-evolving vector from the future, forming a complete description of the system between two time boundaries.

This formalism maps intriguingly well onto the two-phase behavior of supervised learning:

- **Forward propagation:** evolving the input forward through the network to predict an output.
- **Backpropagation:** retroactively applying a loss function at the output and propagating error information backward through the network to adjust weights.

Forward propagation plays the role of the forward-evolving quantum state $|\psi(t)\rangle$, and backpropagation corresponds to the backward-evolving dual state $\langle\phi(t)|$. Together, they constrain the learning dynamics at each layer via a two-state viewpoint.

## 1.22 Forward Propagation as Forward Evolution

In TSVF:

$$|\psi(t)\rangle = U(t,t_0)|\psi(t_0)\rangle,$$

where $U$ is the unitary time evolution operator from an initial preparation at $t_0$.

In machine learning:

$$a^{(l+1)} = f^{(l)}(W^{(l)}a^{(l)} + b^{(l)}),$$

where the activations $a^{(l)}$ propagate the input forward.

## 1.23 Backpropagation as Backward Evolution

In TSVF, a post-selected state $\langle\phi(t_1)|$ evolves backward:

$$\langle\phi(t)| = \langle\phi(t_1)|U(t_1,t),$$

complementing the forward-evolving $|\psi(t)\rangle$.

In neural nets:

$$\delta^{(l)} = (W^{(l+1)})^\top \delta^{(l+1)} \circ f'^{(l)}(z^{(l)}),$$

where $\delta^{(l)}$ encodes error at layer $l$ and propagates backward to adjust weights.

## 1.24 Two-State Update Rule

In TSVF, expectation values take the form:

$$\langle A\rangle_w = \frac{\langle\phi|A|\psi\rangle}{\langle\phi|\psi\rangle},$$

and represent weak values or amplitudes constrained by both past and future states.

In machine learning, the update rule:

$$\Delta W^{(l)} \propto \delta^{(l)}(a^{(l-1)})^\top$$

depends on the forward activations $a^{(l-1)}$ and backward error signal $\delta^{(l)}$. Together, they act like a sandwich operator:

$$\text{Update}^{(l)} \sim \langle\phi^{(l)}|\text{operator}|\psi^{(l)}\rangle.$$

## 1.25 Time-Symmetric Learning View

Rather than treating backpropagation as a mere computational trick, TSVF offers a time-symmetric interpretation:

- Both the input and the desired output state determine the intermediate learning dynamics.
- Each layer mediates between past input and future supervision, forming a time-bridging node.

## 1.26 Comparison Table

| Concept | Neural Network | TSVF QM |
|---|---|---|
| Initial input | $x$ | $\lvert\psi(t_0)\rangle$ |
| Prediction process | Forward prop | $U(t, t_0)$ evolution |
| Target supervision | Loss function | Post-selection $\langle\phi(t_1)\rvert$ |
| Error signal | $\delta^{(l)}$ | $\langle\phi(t)\rvert$ |
| Intermediate activity | $a^{(l)}$ | $\lvert\psi(t)\rangle$ |
| Weight update | $\delta^{(l)}(a^{(l-1)})^\top$ | $\langle\phi\lvert A\rvert\psi\rangle$ |

Figure 1.17: Analogies between supervised learning and TSVF.

## 1.27 FITO Thinking vs. Time Symmetry

Most machine learning frameworks assume *Forward-In-Time-Only (FITO)* causality: input causes output, and learning proceeds only by adjusting from past to future. TSVF suggests a richer model:

- Supervision from the future constrains the learning of the past.
- This bidirectional model aligns with concepts from goal-driven behavior and active inference.

## 1.28 Conclusion

The TSVF reframing of forward and backpropagation illuminates the deeper time-symmetric structure underlying learning. Far from being just a computational trick, backpropagation can be seen as a physical dual to forward propagation—both necessary to fully specify a learning system between two boundary conditions.

This is highly relevant to our model of the protocol, where we use reversible state machines to specify forward propagation, with whatever reversible glitches might be needed to handle failures are implemented as reversible steps, and the backpropagation as the credit-based flow control mechanism.

Because they are completely symmetric, packets being sent and packets being unsent are fully managed by the flow control system.

## 1.29 IP and Patent Implications

The core concept of credit-based flow control is now public domain, but a handful of post-2005 implementation patents remain enforceable through at least 2036. Modern designs should audit those families but can build freely on the expired foundational work.

## 1.30 Practical Take-Aways

1. **Freedom to Operate.** A straightforward "one credit = one buffer" design can rely on the expired Intel/Compaq and Brocade patents for prior-art cover. Avoid features identical to the still-active patents or license them.
2. **Design Around.** Active claims tend to be narrow. You can sidestep the Mellanox "macro credit" idea by limiting link span or by using rate-based pacing instead of credit aggregation.

## 1.31 Rethinking Datacenter Management

From ./AE-Specifications/AE-Specifications/sections/Topology.tex

Owners and operators of the network determine the relationships among distributed applications today. Minimum spanning trees, on which all routing is done, are built, and torn down, by *switches*; based on protocols standardized long ago when we first learned how our computers could communicate.

Today's datacenter architects build their infrastructures using two kinds of boxes: *switches* and *servers*.

We refer to all network devices as switches. Those that route at layer 3 are simply layer 3 switches. 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*topology, where the switches become hubs and servers become leaves.

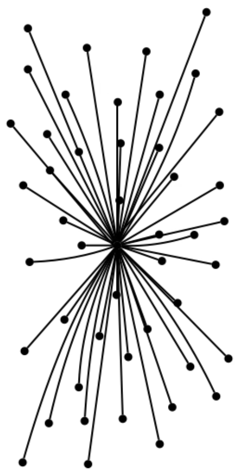


Figure 1.18: CENTRALIZED

Paul Baran's classification provides insight:
`CENTRALIZED (A)` shows 46 univalent nodes connected to a special high radix or *valency*

We use `valency` to denote the number of physical ports on a hyperconverged `cell`. A `cell` is a single type of node element (autonomous unit of compute, storage and packet processing).

A `link` is an individual, bidirectional, computation object (an autonomous communication entity between *two* `cells`) This is to distinguish us from radix, used in switches, but is equivalent to degree ($\delta$), in graph theory.

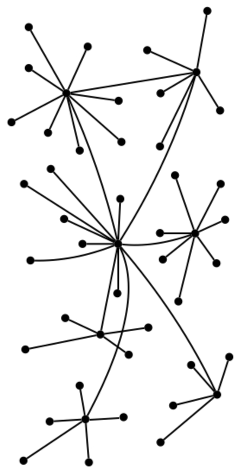


Figure 1.19: DECENTRALIZED

If the central hub dies, all nodes are cut off. `DECENTRALIZED (B)` shows 47 nodes and links, 7 nodes with a valency of 5-7 serve as switches, and 40 as univalent (leaf) nodes. If one of the switches fails, the network fractures into isolated partitions; and only nodes within the partitions can continue to communicate locally. `DISTRIBUTED (C)` shows 47 identical, multivalent (valency $\sim 5$) nodes, and 98 links. The network has better resilience: failed nodes are routed around, and many links must fail before *any* node is finally isolated.

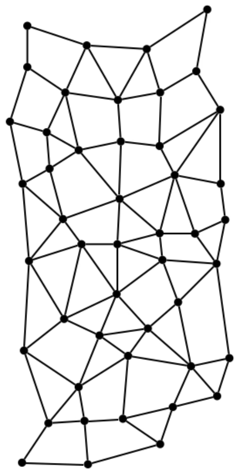


Figure 1.20: DISTRIBUTED

### Datacenter Topologies

`CENTRALIZED` topologies are avoided because they represent bottlenecks and have a single point of failure. `DECENTRALIZED` topologies are (hub & spoke) topologies may be economically necessary for large-scale geographically disparate systems like the interstate roadways and airline networks. are most prevalent, which is surprising given how *non-*

*optimal* they are when looked at from a perspective of distributed microservices container life-cycles

Distributed microservices have an overwhelming predominance of East-West Traffic. Containers can be created in microseconds and last only seconds or milliseconds.

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

The SPoF's in decentralized topologies are mitigated by redundancy in modern multi-slice Clos Networks. Modern Clos networks typically have two, three or four *slices* of spline and leaf switches, along with multiple sets of cables.. Switches networks that perpetuate this model, are embarrassingly complex, unreliable, arcane, and parochial. This results in very high operational costs, poor security/high vulnerability, and nothing close to five nines reliability [From Joe Howard, The perpetuation of complexity:

> "Ethernet and IP networking is embarrassingly complex, unreliable, arcane, and parochial. That results in very high operational costs, poor security/high vulnerability, and nothing close to five nines reliability. In almost any other product category this would be considered unacceptable. Network technology has changed very little since the late 1980s, with the exception of faster speeds/feeds and some additional protocols and features."

`DISTRIBUTED` topologies are rarely used (so far) in datacenters, except for a few HPC applications. Except for HPC applications, which have used various forms of hypercube routing, based on a cartesian coordinate system of destination addresses (a God's-Eye-View), which assumes that failures are rare, and uses complex band-aids to route around failed nodes.

When we use a *relative* addressing scheme instead, routing around failed nodes becomes far simpler, and we can build entirely new kinds of stacked graph covers to provide functionality not previously needed (or envisaged) on hypercube interconnects..

However, within the same *or lower* capital cost, `DISTRIBUTED` topologies provide: greater resilience, lower latencies, higher available bandwidth and far more flexibility; by connecting `cells`

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*

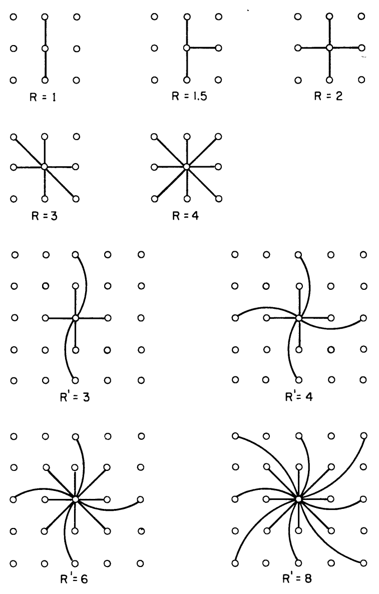


Figure 1.21: Baran: Definition of Redundancy Level

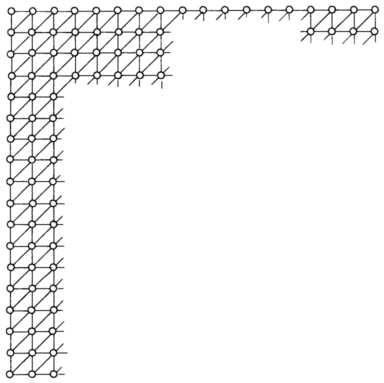


Figure 1.22: Baran: Array of Stations

*of the physical topology, afforded by the* `cell`*'s 'middle' range valency (5 to 9), enables far richer virtual topologies to be built.* directly with neighbor to neighbor (N2N) connections rather than through a switched or aggregated network. By not perpetuating the management complexity of switched networks, and introducing new, simpler, control/forwarding planes through `cells`, we can also dramatically lower operational costs.

Perhaps the time has come to recognize the genius of Paul Baran's insights, and ask why `DISTRIBUTED` topologies are not deployed in datacenters, where their resilience and security can be readily exploited?

## Datacenter Programmability

Two types of teams co-evolved to manage modern 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 infrastructure evolved slowly, and different business units could work within their own silo's. This is no longer a viable architecture in today's highly dynamic multi-tenant datacenters.

Distributed applications can no longer afford to be held back by the slow pace of networking innovation.

While programmable switches may be a promising approach to improve the performance and manageability 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 packet forwarding paradigm of today's network engineering. What is needed to complete this revolution is to include the cells (agents on servers) as first class members of this set of devices which are allowed to route packets as well as process them.

## TRAPHs: Programmable Application Topologies

Critical layers are missing between applications and infrastructure: a layer which contains the evolving *graph* relationships of modern microservices. A substrate that programmers can own and manage themselves[2]. This provides the missing abstraction for a programmable, and deterministic-when-needed, topologies as tools and resources to the application architect. For example, application programmers can program these TRAPHs (Tree gRAPHs) using a Graph Virtual Machine (GVM) to provide services such as distributed consensus, atomic broadcast, and presence management among members of a cluster or microservice set.

[2] A substrate that can work in conjunction with the simpler forwarding functions within NICs and switches

TRAPHs enable datacenter operators to organize *graphs* of resources, managing them on trees, enabling computing on graphs. From the perspective of different vantage points, each with least-privilage[3]. They also provide developers of microservices complete freedom (within the nodes assigned to them), to programmatically determine their sub-relationships, and the protocol characteristics most needed for their applications.

[3] E.g. Managing realms, jurisdictions, tenants and sub-tenants as graphs instead of lists.

## Examples

The advantage or using TRAPHS over a distributed network 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:

*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 applications: erasure coding, machine learning, etc.

*Coherent graph overlays* where cache *heterarchies* co-exist to automatically manage the placement and eviction of caches based on request patterns. One use case would be a coherent configuration file system, which provide a unified mechanism to keep configuration files synchronized, for Docker, etc. Another would be a *coherent* `memcached`. Eliminating cascade failure incidents, like Google saw recently spread to all regions of their Cloud Platform due race conditions to update configuration files.

*Managing Infrastructure as Sets, Graphs & Tensors,* instead of *Boxes, 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 `cells` and `links` allocated to a particular tenant, may be used by that tenant for any topology whatsoever, including allocation to sub-tenants. *Graph computing* on TRAPHs (TreegRAPHs) enable automatic mapping of the natural DAG relationships of distributed applications on nested datacenter infrastructure resources.

**Conclusion:** 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.

## 1.32 The Evolution of Baran to Chiplets

## 1.33 Baran Distributed

## 1.34 Baran Chiplet

## 1.35 Transition to Layer 3-Centric Datacenter Design

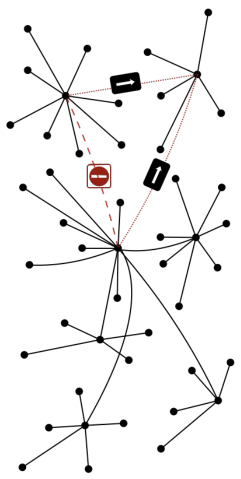


Figure 1.23: Partial Network Partitioning

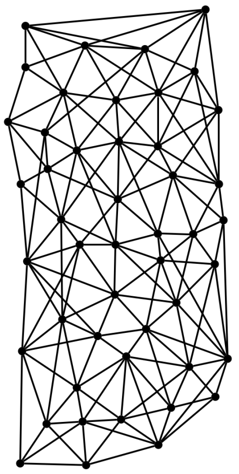


Figure 1.24: Distributed (valency 8)

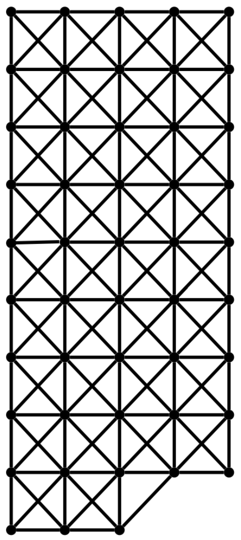


Figure 1.25: Baran Chiplet

Modern datacenter network architectures are increasingly gravitating toward Layer 3 (L3) routing as the default mode of operation. While Layer 2 (L2) bridging continues to underpin legacy assumptions in application design, new trends in silicon, open networking, and container orchestration are enabling a scalable, routed foundation across the fabric.

### 1.35.1 The Fall of Bridging: Application-Layer Assumptions

While the network core transitions to L3, many application-level constructs still implicitly assume bridging:

- **Service and Node Discovery:** Often rely on broadcast, assuming a shared subnet.
- **Cluster Heartbeats:** Use multicast within a broadcast domain.
- **VM Mobility:** Presumes persistent MAC/IP identity and L2 reachability.

These patterns constrain scalability and impede network abstraction.

### 1.35.2 Historical Impediments to L3 Adoption

Legacy resistance to L3 stemmed from:

- Cost: Vendors historically charged premiums for L3 features and protocols (e.g., BGP).
- Complexity: L3 routing was perceived as difficult to configure.
- Lack of Host Support: Quality routing protocol stacks were unavailable on servers.

These constraints are no longer valid.

### 1.35.3 New Enablers for Routed Datacenters

- **Merchant Silicon:** Broadcom, Cavium, and Mellanox now offer chips with bridging and routing parity in performance and cost.

- **Open Networking OSes:** Solutions like Cumulus Linux support native routing (e.g., Quagga, BIRD, ExaBGP).
- **Windows Server Support:** Native BGP support since 2012.
- **Routing Protocol Advances:** BGP and OSPF unnumbered simplify configuration.

### 1.35.4 Kubernetes and the L3 Application Model

Kubernetes abstracts applications from machines, embracing declarative scheduling and management. Key primitives:

- **Pods:** Collections of containers with shared network/storage context.
- **Services:** Logical groups of pods, accessed via virtual IPs.
- **Namespaces:** Logical partitioning for policy and resource isolation.
- **Replication Controllers / Deployments:** Maintain pod count and manage rolling updates.

#### Networking Model

Kubernetes defines a simple yet powerful networking model:

- Every pod receives a routable IP address.
- Pods can communicate without NAT, even across nodes.
- Eliminates port mapping and collisions.
- Designed to operate over routed L3 fabrics (underlay or overlay).

*Overlay Concerns:* While overlays (e.g., VXLAN) enable flexibility, they often incur significant performance overhead. Thus, L3-underlay models are gaining traction.

### 1.35.5 Security and Isolation

Network policy engines (e.g., Calico) leverage namespaces and DAG-style app definitions to:

- Restrict Pod-to-Pod traffic
- Isolate workloads by namespace
- Enforce policy using CNI plugins

### 1.35.6 Routing on the Host

Hosts can participate in the routed fabric via:

- **BGP Config:**
  ```
  router bgp 65534
    bgp router-id 10.10.1.1
    neighbor eth0 interface remote-as external
    redistribute connected
  ```
- **Dynamic Peering:** Using BGP unnumbered or listen range.
- **Stub Behavior:** Hosts are always endpoints; never transit.

### 1.35.7 Services and Load Balancing

- Kube-proxy programs iptables to load balance across pod backends.
- Services expose a stable VIP, accessed via DNS.
- External access enabled via `NodePort` or external load balancers.

### 1.35.8 Conclusion

The shift from L2 bridging to L3 routing in datacenters is well underway, driven by better silicon, open software, and container-native thinking. Kubernetes exemplifies how scalable infrastructure is no longer tethered to subnet-level assumptions, and instead embraces IP-based identity, network policy, and declarative automation.

## 1.36 Hybrid Clocks and Common Knowledge

From ./AE-Specifications-ETH/standalone/hlc$_c$*ommon*$_k$*nowledge.tex*

Hybrid clock research, led by S. S. Kulkarni and collaborators, welds a physical-time estimate to a Lamport-style logical counter. The result is a timestamp that (i) respects causality, (ii) stays within a known skew $\varepsilon$ of wall time, and (iii) avoids the commit-wait penalties of systems such as Google Spanner. This handout summarizes the key papers, explains how Hybrid Logical Clocks (HLC) operate, and analyzes their impact on the epistemic concept of *common knowledge*.

Table 1.1: Kulkarni hybrid-clock lineage

| Year | Work | Key idea |
|---|---|---|
| 2012 | HybridTime | Physical time $\oplus$ counter; bounds skew. |
| 2014 | Hybrid Logical Clock | 64-bit HLC preserves $e \to f \Rightarrow \text{HLC}(e) < \text{HLC}(f)$. |
| 2015 | Hybrid Vector Clock | Vector form; prunes entries older than $\varepsilon$. |
| 16–20 | System integrations | HLC in GentleRain+, CausalSpartanX, NuKV. |

### 1.36.1 How Hybrid Logical Clocks Work

Each node maintains

$$\text{HLC} = (\text{pt}, \text{ctr})$$

where `pt` mirrors physical time and `ctr` counts logical ties.

1. **Local event**

```
pt ← now();
pt ← max(pt,pt);
ctr ← (pt==pt) ?  0 :  ctr+1;
```

2. **Message receive** ($m$.**HLC**)

```
pt ← now();
pt ← max(pt,m.pt);
ctr ← (pt==m.pt) ?  max(ctr,m.ctr) + 1 :  0;
```

Guarantees:

- If event $e$ happens before $f$, then $\text{HLC}(e) < \text{HLC}(f)$.
- Absolute error $|\text{pt} - \text{HLC}| \leq \varepsilon$ as long as the underlying clock discipline meets that bound.

### 1.36.2 Epistemic Implications

#### Classical limit

Halpern and Moses proved that in asynchronous systems absolute common knowledge is impossible; only an infinite tower $E\varphi$, $E\,E\varphi, \dots$ is attainable.

#### Hybrid-clock shift

Because every timestamp deviates from real time by at most $\varepsilon$, a node that observes $\text{HLC} \geq T$ can *deduce* that all events with timestamp $< T - \varepsilon$ are in its past. That turns the impossibility into a quantitative statement: facts can become *$\varepsilon$-common knowledge*.

*Practical upshot* Linearizable commits, causal session guarantees, and wait-free consistent snapshots become feasible after one round trip, provided the skew bound holds.

*Limitation* If $\varepsilon$ blows up (GPS loss, NTP outliers, relativistic links) the deduction fails and classical uncertainty returns. Hybrid clocks do not escape Forward-In-Time-Only thinking; they merely bound its error.

## 1.37 Relation to FITO Perspective

Your Forward-In-Time-Only critique argues that Newtonian time is a hidden axiom. Hybrid clocks expose—rather than erase—this axiom by making $\varepsilon$ explicit. They therefore align with FITO analysis: progress requires either

- shrinking $\varepsilon$ (better hardware sync), or
- replacing deterministic bounds with probabilistic or reversible notions of order.

## 1.38 Open Problems

1. **Dynamic $\varepsilon$.** Adapt clocks when skew drifts.
2. **Probabilistic knowledge.** Treat timestamps as confidence intervals.
3. **ICO-aware clocks.** Design schemes that tolerate indefinite causal order and reversible transactions.
4. **Eventual common knowledge.** Combine HVC pruning with DAG gossip in partitioned networks.

## 1.39 Takeaways

Hybrid clocks bridge logical causality and imperfect wall time, achieving the effect of common knowledge after a bounded delay $\varepsilon$. They power modern geo-replicated stores without heavy coordination cost, but remain inside the FITO worldview. Future work will loosen or replace the global arrow of time.

## References

[1] S. S. Kulkarni and N. Mittal. HybridTime: A decoupling of coordination and time in distributed systems. TR, 2012.

[2] S. S. Kulkarni, et al. Logical Physical Clocks and Consistent Snapshots. 2014.

[3] V. Karmarkar and S. S. Kulkarni. Bounds on Hybrid Vector Clocks. IEEE SRDS, 2015.

[4] H. Wu, et al. CausalSpartanX: Causal consistency over HLC. Middleware, 2016.

[5] J. Su, et al. NuKV: Building a scalable and reliable KV store. SoCC, 2020.

[6] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in distributed environments. JACM 37(3), 1990.