1. Addressing and Routing

1.1 Addressing and Routing in Chiplet Ethernet

Adapted from [PLB][WIP] DAE-Spec/source-destination-routing.tex

In a Chiplet Ethernet environment, routing mechanisms determine how packets traverse the on-chip network from source to destination. Depending on the architectural goals and design constraints, various routing schemes may be employed. The three primary approaches considered here are:

Source Routing The sender encodes the entire route in the packet.

Destination-Based Routing Each hop examines the destination address and forwards accordingly.

Name-Based Routing The routing is based on service or data identifiers, rather than physical addresses.

1.1.1 Source Routing

In source routing, the source node determines and encodes the full path the packet should follow.

- **Mechanism:** Routing instructions are embedded in the packet header, specifying port transitions at each node.
- **Advantages:** Low per-hop complexity; ideal when the source has full network visibility.
- **Drawbacks:** Overhead due to path encoding; lacks flexibility under topology changes.

1.1.2 Destination-Based Routing

Here, packets carry a destination address and each node independently determines the next hop.

- **Mechanism:** Intermediate nodes use forwarding tables to route toward the destination.
- **Advantages:** Familiar, scalable, and adaptive; supports route recalculation during faults.
- **Drawbacks:** Per-hop table lookups increase logic complexity and memory footprint.

1.1.3 Name-Based Routing

Packets are routed based on service names or abstract identifiers rather than fixed addresses.

- **Mechanism:** Routers resolve names via distributed directories or longest-prefix matching.

- **Advantages:** Enables service-level decoupling, dynamic mapping, and resource abstraction.
- **Drawbacks:** Requires complex lookup mechanisms and incurs packet header overhead.

1.1.4 Routing Paradigm Summary

Routing Type	Header Overhead	Per-Hop Logic	Flexibility
Source Routing	High	Low	Low
Destination-Based	Low	Medium	Medium
Name-Based	Medium/High	High	High

1.2 Ants, Bees, Snakes, Spiders, and Worms: Biologically Inspired Topology Learning

To support adaptive and scalable routing in chiplet interconnects, we propose several biologically inspired methods for topology discovery and maintenance, especially for networks with nodes of valency 4, 6, or 8.

1.2.1 Ant-Based Discovery

- **Scouts:** Explore the network, collecting path metrics.
- **Pheromone Trails:** Nodes cache recent path qualities.
- **Reinforcement:** Successful paths are promoted; old paths decay.
- **Usage:** Enables adaptive routing without global knowledge.

1.2.2 Bee-Inspired Exploration

- **Hive Nodes:** Aggregate partial topological information.
- **Scouts:** Sample paths and report back.
- **Dance Protocols:** Broadcast high-quality routes to other nodes.
- **Usage:** Useful for semi-centralized routing optimization.

1.2.3 Snake Traversals

- **Traversal Packet:** Systematically visits each reachable node.
- **Data Aggregation:** Builds complete network maps.
- **Return Path:** Reports findings back to the origin node.
- **Usage:** Suitable for diagnostics or rare full-network verification.

1.2.4 Spider Web Construction

- **Threads:** Discovery packets sent on all ports.
- **Local Web:** Maintains neighbor lists and multi-hop connections.
- **Tension Metric:** Indicates link quality (latency, congestion).

- **Usage:** Builds resilient, redundant local meshes.

1.2.5 Wormhole Routing: Worm Behavior as Forwarding Strategy

Wormhole routing segments packets into flits and forwards them in a pipeline manner.

1.2.6 Comparison with Other Techniques

- **Store-and-Forward:** Full packet buffered at each hop; simple but high-latency.
- **Cut-Through:** Begin forwarding upon header arrival; requires moderate buffer sizes.
- **Wormhole:** Forward flits immediately, with minimal buffers.

1.2.7 Mechanics of Wormhole Routing

- **Head Flit:** Reserves path.
- **Body Flits:** Stream through reserved path.
- **Tail Flit:** Releases resources.

1.2.8 Hardware Considerations

- **Buffers:** Minimal per-port buffering (1–2 flits).
- **Flow Control:** Credit-based or handshake-based.
- **Stall Propagation:** Congestion can block entire worm.
- **Virtual Channels:** Prevent deadlocks and increase concurrency.

1.2.9 Integration Strategies

- **Ant + Wormhole:** Explore paths via ants, forward data via wormhole flits.
- **Snake + Destination-Based:** Use full traversals to populate local forwarding tables.
- **Spider + Name-Based:** Webs retain service-to-node mappings.

1.3 Conclusion

Chiplet interconnects require a blend of scalable discovery protocols and efficient forwarding strategies. By combining biologically inspired mechanisms (ants, bees, snakes, spiders) with low-overhead routing paradigms (worms), the network gains resilience, adaptability, and performance.

These approaches can be tuned to the chiplet topology and application domain, forming a foundation for robust next-generation system-on-chip communication fabrics.

1.4 Biologically Inspired "Scouting" before "Routing"

Came from ./AE-Specifications-Eth/Biology.tex

1.4.1 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.

1.4.2 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.

1.4.3 Bufferless (Hot-Potato) Routing

1.4.4 Basics of Bufferless 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.

1.4.5 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.

1.4.6 Deflection Routing

1.4.7 How Deflection Routing Works

- 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.

1.4.8 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.

1.4.9 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."

1.4.10 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 in-

spired heuristics can be adapted to local, minimal-knowledge scouting decisions.

1.4.11 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.

1.5 Ants, Bees, Snakes, and Worms

Came from 2025-papers/Topology-Addressing/Topology-Addressing.tex

A sea in IPUs, within Rack-Scale and Chiplet Interconnects require additional levels of abstraction for configuration and reconfiguration than is provided in existing Standards. This paper Explores biologically inspired topology-learning and routing methods in networks of 8-bidirectional port nodes, with a special focus on wormhole routing and its variants. We then take a look at current state-of the art routing technologies, such as deflection forwarding combined with local compass point addressing and see how they match.

Modern system-on-chip designs increasingly rely on chiplet-based architectures, where multiple specialized chiplets—each dedicated to a particular function (e.g., CPU, GPU, accelerator)—are connected to

form a unified and scalable platform. Ensuring that these chiplets communicate efficiently poses a variety of challenges, especially as the number of chiplets grows and each chiplet (or on-chip router) may have multiple ports (valencies of 4, 6, or 8).

Existing solutions often rely on standard routing protocols or static configurations. However, as topologies become denser, more heterogeneous, and subject to partial reconfiguration, novel methods are needed to learn, update, and maintain the interconnect topology. Drawing inspiration from biological systems—where ants, bees, snakes, spiders, and now worms—each exhibit unique strategies for exploration, organization, and adaptation, promises fresh ideas for topology discovery, addressing, and low-level routing mechanisms.

1.5.1 Challenges in Chiplet Interconnects

1. Scalability: As the number of chiplets increases, the complexity of maintaining accurate global network knowledge grows exponentially. Conventional centralized or static approaches can struggle to stay current and efficient.
2. Partial Knowledge: On-chip routers typically have limited local visibility. They need to cooperate with neighbors to build a broader perspective of the network.
3. Adaptive Reconfiguration: Certain chiplets may power down, enter low-power states, or be repurposed. The routing and addressing approach must cope with these dynamic behaviors.
4. Latency Sensitivity: On-chip communications can be highly latency-sensitive. Any topology exploration or routing technique must have minimal overhead in time and power.
5. Physical Constraints: Depending on the routing mechanism (e.g., store-and-forward, cut-through, or wormhole routing), the buffer sizing, FIFOs, and local SRAM usage become critical design considerations.

Biologically inspired approaches have shown promise in large-scale or dynamic networks because they emphasize local decisions and emergent global behaviors, while specialized routing methods at the flit or byte level can significantly impact network performance and resource usage.

1.5.2 1. Ants: Stigmergic Discovery & Adaptive Routing

In nature, ants communicate via pheromones—chemical trails used to discover and reinforce paths to resources. In a chiplet context, ant-inspired algorithms involve agents (packets) that:

1. Randomly Explore: “Scout ants” are periodically sent into the network to discover unknown or less-traveled routes.

2. Deposit "Pheromones": As these scouts move, they leave "digital pheromones" in local tables, indicating path quality or latency.
3. Positive Feedback: Over time, successful routes are reinforced, prompting data packets to favor paths with strong pheromone levels.
4. Decay Mechanism: Pheromones degrade, allowing the system to adapt if congestion or failures cause route performance to change.

1.5.3 Benefits:

- Adaptive to Congestion: Routes are continuously refined by real-time traffic patterns.
- Localized Decisions: Each node handles small amounts of metadata without needing a global map.
- Minimal Hardware Overhead: Requires storing pheromone metrics (e.g., counters, latency measures) in local tables.

1.5.4 2. Bees: Collaborative "Hive" and "Scout" Behavior

Honeybees manage tasks by scouting for resources and returning to the hive to share discoveries. A bee-inspired algorithm can leverage:

1. Hive Nodes: Certain nodes (management chiplets) aggregate topology or performance data.
2. Scout Bees: Packets leaving the hive to explore unknown or under-explored areas, returning with updated route metrics.
3. Recruitment: High-value or high-performance routes are "advertised," encouraging worker packets to follow them.
4. Dance Protocol: Returning scouts might broadcast partial routes or performance gains, influencing how future packets select paths.

1.5.5 Benefits:

- Semi-Centralized Knowledge: Hive nodes maintain or compile partial global knowledge for better load balancing.
- Dynamic Adaptation: Over time, good routes become well-known, while less efficient links see fewer packets.
- Diagnostic Aid: Central nodes can help with debugging or performance tuning.

1.5.6 3. Snakes: Sequential Path Traversals for Comprehensive Mapping

Snakes systematically traverse an environment. In a chiplet network, a snake-inspired approach might:

1. Slithering Packets: A special packet is launched that attempts to traverse every reachable node in a systematic pattern.

2. Topology Collection: As it hops, it collects neighbor lists, port connections, and node identifiers.
3. Loopback: Upon completing a traversal (or hitting boundaries), the packet returns to its origin with a consolidated network map.
4. Distributed Snapshots: Multiple vantage points or repeated "snake sweeps" yield an updated global view of connectivity.

Benefits:

- Guaranteed Coverage: Ensures every node and link is discovered periodically.
- Simplicity: Conceptually straightforward, though it can be heavier in overhead.
- Occasional Diagnostics: Particularly useful for network-wide verification or fault checks.

1.6 4. Spiders: Building and Maintaining "Webs" of Connectivity

Spiders create webs that dynamically adapt to external stresses or breaks. A "web-based" approach for chiplet networks focuses on building a resilient mesh:

1. Web Construction: Each router sends out "threads"—short discovery packets—on all ports. Neighbors respond, forming local connectivity data structures.
2. Local Weave: Threads intersect and overlap, letting routers learn about multi-hop neighbors.
3. Damage Repair: If a link fails, local threads are resent to repair or reroute around the break.
4. Tension Metrics: Each link in the web holds a "tension" (latency, throughput) that can be monitored and used to shift traffic if congestion or errors rise.

Benefits:

- Resilience Through Redundancy: Overlapping "threads" ensure multiple known paths.
- Incremental Updates: Each node refines its local web structure.
- Ease of Local Addressing: Short IDs can be assigned to neighbors, aggregated as the web extends outward.

1.7 5. Worms: Segmenting and Traversing via Wormhole Routing

While ants, bees, snakes, and spiders mainly address topology learning and discovery, "worms" connect directly to how data flows once routes are known. In wormhole routing, the packet travels through the network in segments—often called flits ("flow control units")—that

form a pipeline from source to destination, much like a worm tunneling through soil.

1.7.1 5.1 Wormhole Routing Basics

In traditional store-and-forward routing, each node receives the entire packet, stores it in a buffer, then forwards it to the next node. This requires sizable FIFO or SRAM at each hop.

In cut-through routing, a node can begin forwarding the packet to the next hop as soon as the header is processed—without waiting for the entire packet to arrive—assuming the next link is available.

Wormhole routing takes cut-through a step further:

1. Head Flit Reservation: The first flit (the "head") reserves a path through each router as it progresses.
2. Body Flits: Subsequent flits follow in a pipeline—only minimal buffer space (a few flits) is needed at each router.
3. Tail Flit Release: Once the tail flit exits a router, the resources can be released and reused for another packet.
4. Blocking: If a busy link stops the head flit, the entire packet may stall, occupying small buffers across multiple nodes like a worm stretching through the network.

1.7.2 5.2 "Worm" Analogy

Much like a biological worm that elongates and contracts through tight spaces:

- Elongation: As the head flit moves forward, it effectively extends the pipeline.
- Contraction: Once the tail flit exits a router, that router is freed—much like the worm's tail leaving a tunnel behind.
- Sensitivity to Congestion: If a worm encounters a "block" (busy link), it must wait in place. A real worm might avoid or reroute around obstructions, suggesting that combining wormhole routing with "ant" or "bee" style dynamic route discovery could reduce blocking situations.

1.7.3 5.3 Distinguishing Packet-, Flit-, and Bit-Level Forwarding

- Store-and-Forward (Packet-Level): Each node stores the full packet in a local buffer before forwarding. This simplifies control logic but requires larger FIFO or local SRAM.
- Cut-Through (Often Byte/Flit-Level): Forwarding starts once enough of the packet (header) arrives and the downstream link is reserved. This reduces latency but still needs capacity to handle the largest packet if blocking occurs.

- Wormhole (Flit-Level): Often has the smallest per-node buffer requirement. Each node only needs space for one or a few flits. If the path is blocked, flits in-flight remain distributed across multiple routers.

1.7.4 5.4 Biologically Inspired Routing and Bufferless Design

Borrowing from the "worm" metaphor, one might imagine a system where:

- **Exploration "Head"**: Uses "ant-like" or "bee-like" exploration to find a path.
- **Data "Body"**: Follows in a wormhole fashion along that discovered path.
- **Adaptive "Segments"**: If blocked, the worm could "shed segments" or re-route partway (requires advanced partial re-routing logic) akin to how some worms can regenerate or re-segment.

Integration with Addressing and Routing

The biologically inspired discovery methods (ants, bees, snakes, spiders) can guide or complement the forwarding mechanism (worms via wormhole routing, or store-and-forward, or cut-through). For instance:

- **Name-Based Routing + Wormhole**: "Spider web" or "ant trails" might store service names or resource mappings, while "wormhole" flits pipeline across chosen paths once discovered.
- **Source Routing + Wormhole**: The source determines a path (potentially from an "ant" or "bee" exploration), then sends data flit by flit to the next node.
- **Destination-Based Routing + Wormhole**: Each node consults a local table (built by "snake" or "spider" sweeps) to direct the head flit.

Practical Considerations

1. **Overhead vs. Benefit**: While biologically inspired discovery can adapt well to changing conditions, each approach introduces control packets or additional logic.
2. **Hybrid Approaches**: Some systems run an "ant-like" algorithm for local path optimization but periodically launch "snakes" for global verification, then use wormhole routing for actual data transfer.
3. **Hardware Feasibility**: On-chip routers with 4, 6, or 8 ports must have carefully sized buffers. Wormhole routing can save SRAM but demands robust flow control to avoid deadlocks.
4. **Complex Resource Management**: Virtual channels, flit-level flow control, and dynamic reconfiguration add to design complexity,

though they allow for higher performance and flexibility.

Conclusion

As chiplet-based systems continue to evolve, combining unconventional, biologically inspired methods of topology discovery (ants, bees, snakes, spiders) with specialized data transfer techniques like wormhole routing ("worms") opens up promising new frontiers. This fusion can lead to adaptive, resilient, and efficient on-chip communication—crucial in an era of many-chiplet systems requiring rapid reconfiguration and minimal latency.

Each of these paradigms underscores a key principle: localized, iterative learning and distributed adaptation can collectively produce robust global outcomes. Layered on top of a suitable forwarding mechanism—whether store-and-forward, cut-through, or wormhole—these biologically inspired methods help chiplet interconnects gracefully handle complexity, partial failures, and reconfigurations, ultimately paving the way for faster and more reliable on-chip networks.

PART TWO: Connection with Bufferless and Deflection Routing

Below is an overview of how biologically inspired "scouting" or "discovery" mechanisms connect with key ideas in bufferless (hot-potato) routing and deflection routing, especially in mesh-like topologies where routers have ports arranged along the cardinal (N, E, S, W) and intercardinal (NE, SE, SW, NW) directions.

1. Biologically Inspired "Scouting" and "Routing"

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.

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).

2. Bufferless (Hot-Potato) Routing

Basics of Bufferless Routing:

- **No Packet Buffers (or Very Limited Buffers)**: Every router either immediately forwards or deflects each incoming packet.
- **Hot-Potato / Deflection Character**: If the preferred output port is unavailable, the packet is sent out another (less ideal) port.

Connection with Biologically Inspired Approaches:

- **Continuous Movement**: Biologically inspired scouts are designed to wander; in a bufferless system, "wandering" (via deflections) is also central.
- **Adaptive Reinforcement Over Time**: Pheromone or congestion metrics guide most packets down better ports, even if some must deflect.

3. Deflection Routing

How It Works:

- **Forced Misrouting / Deflection**: If the best output port is busy, the packet takes an alternative route.
- **Common in Low- or No-Buffer Architectures**: Used when buffering is not an option.

Compass Ports Integration:

- **Local Prioritization**: A packet heading NE may prefer N or E, deflect to NE, or in worst cases, NW/SE.
- **Biologically Inspired Ranking**: Use pheromone levels to rank output directions and pick the best available port.

4. Example Flow in an 8-Port Router

1. Receive a packet from the south port.
2. Look up destination (e.g., north-east direction).
3. Check local pheromone table: NE is preferred.
4. If NE port is free, forward the packet.
5. If NE is busy, try N or E or fallback ports.
6. If all preferred ports are busy, deflect to any open port (e.g., SW).
7. Update pheromone levels based on eventual delivery success.

5. Literature and Further Reading

- **Hot-Potato Routing / Deflection Routing:**

  - Baran, P. (1962). *On Distributed Communications Networks*. IEEE Transactions on Communications.
  - Dally, W., & Towles, B. (2004). *Principles and Practices of Interconnection Networks*.
- **Biologically Inspired / Ant-Based Routing:**
  - Di Caro, G. A., & Dorigo, M. (1997). *AntNet: Distributed stigmergetic control for communications networks*. JAIR.
  - Schoonderwoerd, R., et al. (1996). *Ant-based load balancing in telecommunications networks*. Adaptive Behavior.
- **NoC with Deflection / Bufferless Routing:**
  - Moraes, F. et al. (2004). *A Low Area Overhead Packet-switched Network on Chip*. SBCCI.
  - Fallin, C., et al. (2012). *CHIPPER: A Low-Complexity Bufferless Deflection Router*. HPCA.

6. Concluding Remarks

- **Shared Tenets**: Both biologically inspired scouting and deflection routing use localized decision making to produce emergent behavior.
- **Complementary Mechanics**: Pheromone feedback integrates naturally with bufferless routing decisions.
- **Directional Routing**: Cardinal and intercardinal ports align well with the 2D mesh and natural directional behavior in biological metaphors.

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

1.8 From Hesham

I am more familiar with Ant Colony Optimization Algorithms (ACO) which are used in many applications e.g.

1. Energy and thermal aware mapping for mesh-based NoC architectures using multi-objective ant colony algorithm
2. Ant Colony Optimization-Based Adaptive Network-on-Chip Routing Framework Using Network Information Region

1.9 Design Principles

Came from KAO-DAE-Specifications/Sections/DesignPrinciples.tex

The Daedaelus Substructure (DS) is made of Cells (physical nodes, complete with processor, memory, and FPGA-Enabled SmartNICs),

and physical Links which run the DSPS Protocol, and are based on the following principles:

Every cell has a constrained 7 ± 2 *ports per cell* [1], which balances the need for enough path diversity between any pair of cells to make partitions extremely rare while bounding the complexity of the algorithms for routing, healing and recovery. However, the most significant advantage of a constrained number of ports (valency) is in the building of long-lived Treez and their extended paths (dendrites) which can be used for self-organization[2].

Local information only Each cell has information only about itself and the cells its ports are connected to. While this limitation rules out certain global optimizations, it enhances flexibility in that only neighbors of a cell need deal with changes made to that cell. For example, adding a link only affects the two cells it connects. Local Observer View (LOV).

Failure detection with DSPS provides Temporal Intimacy (The `TIKTYK TIK`, The I Know That You Know That I Know property). `TIKTYKTIK` simplifies the enforcement of desired properties, such as in-order message delivery (even when a failure forces subsequent messages to take a different route to their destination), and Atomic Information Transfer (AIT).

Event driven The network fabric is *event driven*, *i.e.*, there are no network-specific timeouts. Self-sustaining events are created in the `links` using the ENTL mechanism. This is in contrast to conventional computing, where all events are driven by processes running on one or more cores of a processor.

Every cell builds a spanning tree with itself as root A spanning tree provides certain guarantees. Each root can reach all other cells in the fabric with a *tree cast*, and each cell can reach the root with a *root cast*. In addition, it is easy to enforce in-order delivery of messages between any pair of cells, simplifying many distributed computing algorithms.

Each link keeps state needed to recover from routing around its failure Link state is maintained by the two ports it connects to. This information is used to reestablish the ordering guarantees when routing around the link if it fails, to maintain complimentary AIT state on each end of the link, and to silently maintain "presence" for the two parties.

Each cell keeps state to aid in routing around failures The spanning trees (black trees) are built on the `groundfabric` (the physical connectivity), and all cell trees together represent the `groundplane` which is a

[1] Typical servers today have from 6-8 ports; with an additional 2 ports are used for connecting to Top of Rack (ToR) switches

[2] For example, for migrating data and computations automatically, based on local only information, to create chains or gradients of, for example, data temperature, or the automatic layout of graph computations, or evolving topologies for deep learning applications.

connected undirected logical graph that matches the underlying `groundfabric` exactly. When building the spanning trees, each cell records the state of each port for each tree in the fabric. For each tree one port points to a parent, others point to zero or more children, and the rest are not part of the tree. These ports have been *pruned*. We call this state a *TRAPH*, for Tree-Graph. When the link connected to the port pointing to a parent fails, the cell uses the TRAPH data to find an alternate port to reach the new parent.

`groundplane` is the lowest sub-level of the FPGA Substructure : the physical layer. All TRAPHs (selected graph covers) are built as subsets of the `groundplane`. Each TRAPH has a controlling tree (the owner), based on the cell which created and named the TRAPH. This owner cell may keep itself as the sole entity which controls the TRAPH, but more typically, it will select one or more other cells to provide failover should it fail. The failover protocols (Tree Paxos) are described in a separate section of this document.

A recursively stacked set of logical TRAPHs may be built on the `groundplane` to provide the logical segregation layers for secure provisioning and confinement for management entities, such jurisdictions, tenants and sub-tenants. These will often be referred to as the *grey* layers, because they are in the substructure, and are hidden from the hypercontainers above (which run on the other side of the PCIe bus in the hose processor).

Virtual (Colored) TRAPHs may be stacked on top of any of the logical gray layer TRAPHs. Virtual TRAPHs are available under API control for the operating systems and applications above.

1.9.1 Relative Addressing

A major distinction from conventional data centers is that ECNF addressing is based on cell-to-cell (C2C) direct connections. Every communication from an agent on a cell is to one or more ports. The message is sent to the port on the other side of the link, which can forward the message on one of that cell's ports just as switches do.

This addressing scheme has a number of advantages. For example, each cell needs to maintain limited routing information. Should the path to some target agent change, perhaps because it migrated to another cell or due to a failover, only a limited number of cells need to update their routing information. Additionally, an attacker who penetrates a cell can only address its direct neighbors, limiting any potential damage

1.9.2 Port Labelling

Some failover algorithms use path information in the form of a sequence of port identifiers between some cell and the root. Each cell is free to assign arbitrary labels to its ports as long as each of its ports is assigned a label unique to the cell. However, debugging and network management will be easier if a consistent convention is used, as shown in Figure **??** for a cell with 12 ports.

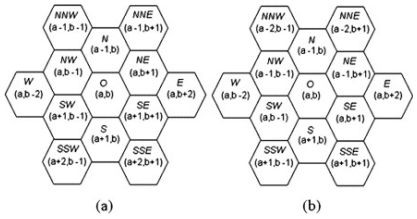

Figure 1.1: Port Labeling convention: 6–12 ports

1.9.3 Hop count

Knowing the number of hops from a given cell to the root can help in selecting shorter failover paths. However, there is a cost. When the path from a cell to the root changes, other cells in the TRAPH may need to be notified to update their hop counts.

1.9.4 Path information

When a link breaks, LW must find a new path to the root of every tree that uses the now broken link. Hop count helps guide the selection of which port to try next, but the new path must be tested all the way to the root. Keeping path information in the form of the each cell's port identifier on the path from the root can reduce that cost. As with hop count, other cells in the TRAPH must be notified to update their path information following a failover.

While messages carrying this information grow with the length of the path, the size is manageable. If we use four bits to label each port, and we use 1K of a 1,500-byte packet for path data, then we can manage 2,000 element paths with a single packet. With balanced trees in a datacenter with N cells, the longest path will be $\log_8 N = \frac{1}{3}\log_2 N$, which is far less than 2,000 for any datacenter envisioned today. Even with unbalanced trees grown stochastically, the longest path is likely to be $O(\sqrt{N})$. We can choose to truncate the path information at the cost of slightly somewhat more attempts before finding a new path.

Introducing Kleinberg links makes the problem tractable even with 64-byte packets, since the average hop count is around 5. Of course, some paths will be longer, but taking 10 bytes of the packet for path information allows us to handle up to 20 hops. We can truncate the path information of the few paths that are longer.

1.9.5 Packet-level Time Reversal

When a recipient of a packet associated with an AIT token cannot pass the packet to the next higher level in the stack, it implements local *time reversal*. The result is that the cells on the two sides of the link end up in the same state they were in before the packet in question was sent. This procedure can be extended to any number of intervening nodes.

This section is premised on the fact that we want to do this kind of time reversal when a link breaks, but I'm not sure why. It would seem that LW only needs to know if it should send the last packet sent before the failure to J, the join node responsible for reestablishing message ordering. No time reversal needed.

1.10 Introduction

Came from ./AE-Specification-ETH/standalone/Graph-Algorithms.tex

This document proposes a Layer 2 Ethernet routing protocol called *Scouting at Layer 2*, inspired by recent advances in edge coloring algorithms from graph theory. It aims to enable deterministic, loop-free path discovery and forwarding solely at the MAC layer, bypassing the need for Layer 3 mechanisms such as IP routing.

Ethernet is traditionally a broadcast-based Layer 2 protocol, relying on IP-based Layer 3 protocols for routing. However, modern data centers and specialized networks demand low-latency, deterministic, and topology-aware communication mechanisms without the full overhead of the IP stack. To this end, we introduce *Scouting at Layer 2*, a distributed routing protocol that operates entirely within the Data Link Layer.

The design is motivated by recent work on edge coloring of graphs in near-linear time **?**, which provides a scalable and collision-free scheme for link differentiation.

1.11 Design Goals

- **Layer 2 Only**: Operates exclusively using MAC addresses.
- **No Broadcast Storms**: Avoids STP/RSTP flooding and enables deterministic paths.
- **Loop-Free Forwarding**: Guarantees no cycles using path identifiers.
- **Dynamic Topology Support**: Accommodates changes in network structure.
- **Low Computational Overhead**: Efficient enough to run in SmartNICs or ASICs.

1.12 Graph-Theoretic Inspiration

In the protocol, each Ethernet node and its direct connections are modeled as a graph $G = (V, E)$, where:

- Vertices V are MAC-layer devices (bridges, switches, NICs).
- Edges E represent Ethernet links.
- Each edge is assigned a **color**, i.e., a unique local forwarding tag, such that no two edges incident to the same vertex share the same color.

Using the result of Li et al. **?**, we can perform this coloring with at most $\Delta + 1$ colors in $O(m \log \Delta)$ time, where $m = |E|$ and Δ is the maximum degree.

1.13 Protocol Description

1.13.1 Initialization Phase

Each node performs neighbor discovery and assigns temporary colors (tags) to its outgoing links, ensuring local uniqueness. Nodes then gossip their tag assignments to neighbors until a global stable coloring is reached.

1.13.2 Path Discovery Phase

To reach a given MAC address, a node constructs a sequence of tags (path identifiers) describing a color-consistent path through the network. These path sequences are constructed in a Dijkstra-like traversal with color-awareness to avoid collisions.

1.13.3 Frame Forwarding

Frames are modified to include a *Path Identifier Sequence (PIS)* field, which encodes the list of edge colors (tags) to follow. As the frame traverses the network:

1. The switch reads the next tag in the PIS.
2. It matches this tag to an outbound port.
3. It decrements the PIS and forwards the frame.

This process continues until the PIS is empty, and the destination MAC is reached.

1.14 Frame Format

Field	Length (Bytes)	Notes
Destination MAC	6	Standard MAC
Source MAC	6	Standard MAC
Type/PIS Identifier	2	Ethertype or custom
Path Identifier Sequence	Variable	Encoded tag list
Payload	Variable	As usual
FCS	4	Standard CRC

1.15 Advantages

- No need for Layer 3 routing tables.
- Supports programmable switching (e.g., in SmartNICs or eBPF).
- Scales to large networks with sparse connectivity.
- Deterministic pathing avoids congestion and loops.

1.16 Challenges

- Path sequence length is limited by MTU.
- Requires coordination to avoid inconsistent tag assignment.
- Topology changes require propagation of new path info.

1.17 Applications

- HPC clusters with low-latency mesh topologies.
- Edge compute zones with fixed link-layer infrastructure.
- Datacenter overlays where IP is inefficient or unavailable.

1.18 Conclusion

By leveraging edge-coloring strategies for deterministic path discovery and forwarding, Scouting at Layer 2 offers a novel approach to MAC-layer routing. Its foundation in recent algorithmic advances such as those by Li et al. **?** provides a robust and efficient scheme, particularly suited to programmable and high-performance environments.