Hardware Feasibility for (Semi-)Oblivious Reconfigurable Networks

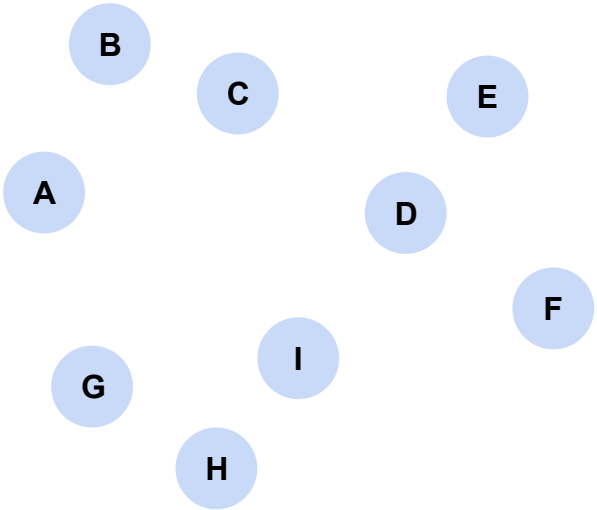
Speaker: Yunxi Shen

Based on joint work with Daniel Amir, Nitika Saran, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon

Challenges for ORN / SORN Hardware

- How can congestion control be implemented for ORNs/SORNs?
- Can (semi-)oblivious routing be implemented with low clock cycle overhead?
- How can on-chip memory consumption be minimized for better scalability?

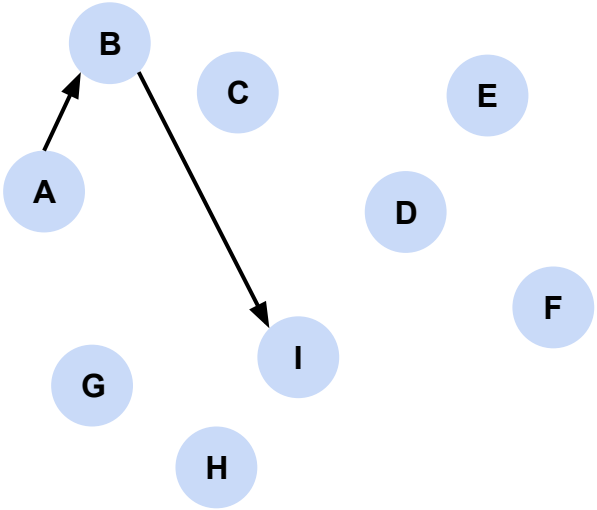
Oblivious Reconfigurable Networks



Shale

- Divides the datacenter into *h* dimensions
  - 1 round-robin for each dimension
- Valiant Load Balancing
  - 1 spraying hop + 1 direct hop for each dimension

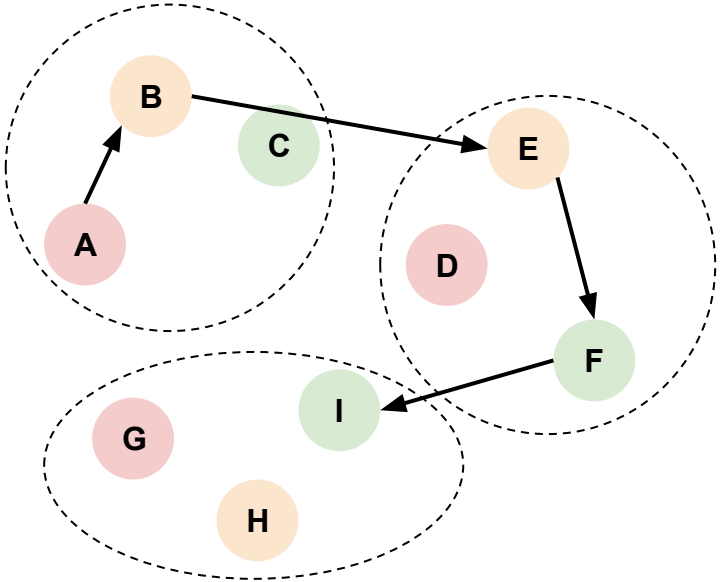
Oblivious Reconfigurable Networks



h=1

- 1 round-robin -> schedule length is 8
- 1 spraying hop + 1 direct hop

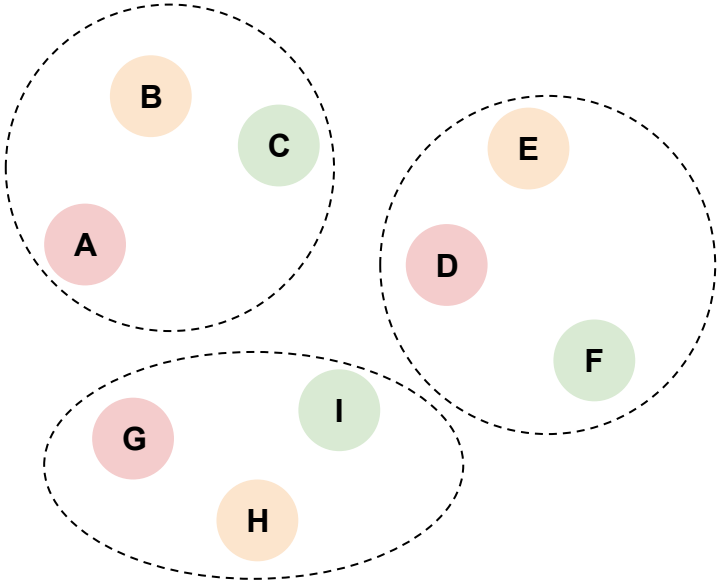
Oblivious Reconfigurable Networks



h=2

- 2 round-robins -> schedule length is 4
- 2 spraying hops + 2 direct hops

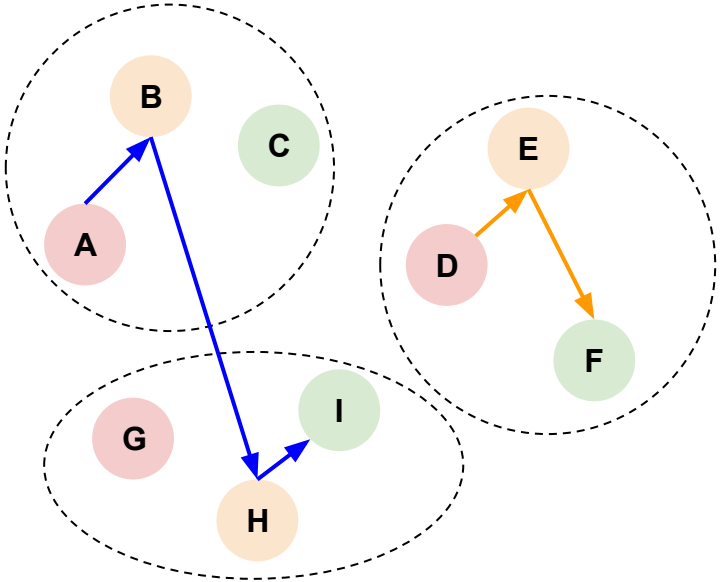
Oblivious Reconfigurable Networks



h↗

- Shorter schedule -> lower latency
- More hops -> lower throughput

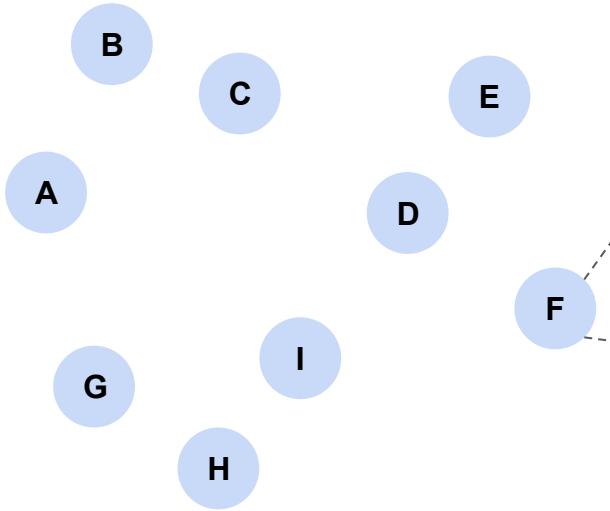
Semi-Oblivious Reconfigurable Networks

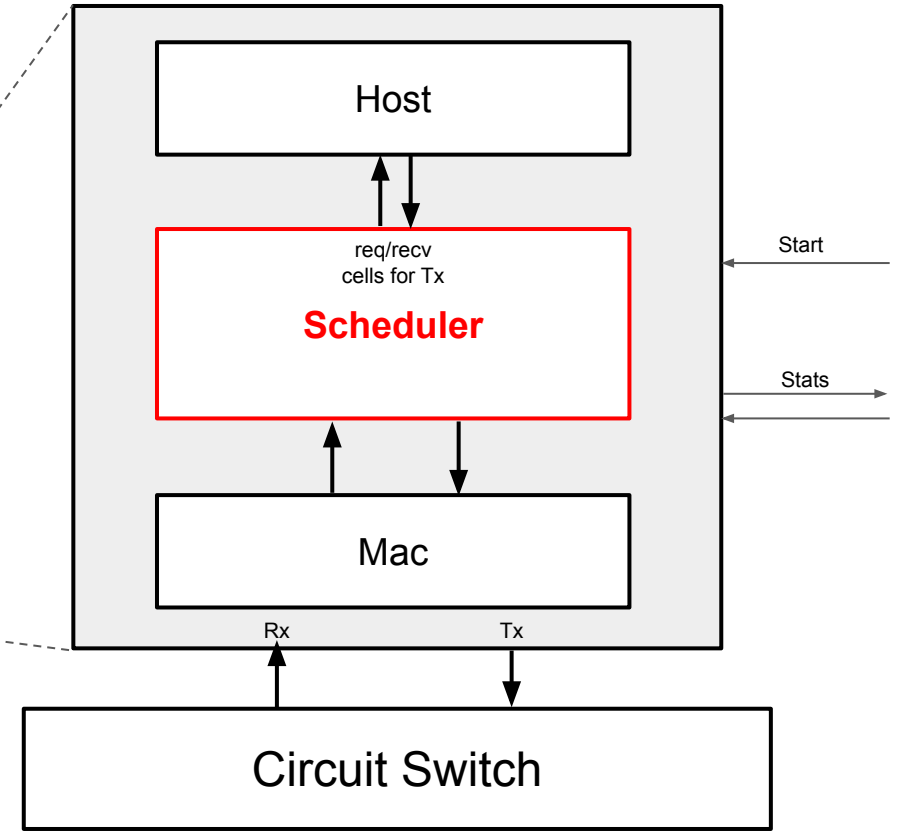


SORN

- Intra-cluster: 1 spray + 1 direct
- Inter-cluster: 1 intra-cluster spray + 1 inter-cluster direct, 1 intra-cluster direct

Hardware Design



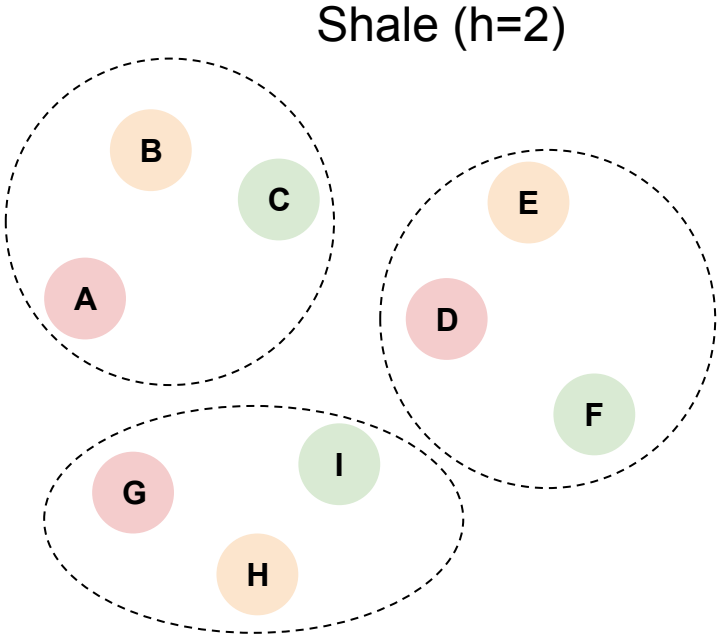


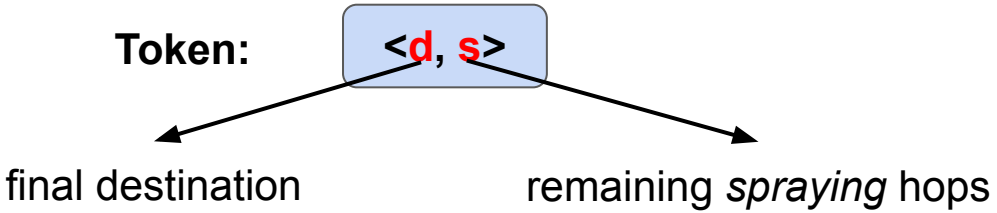
- **Terasic DE5-Net boards, Altera Stratix V FPGA**
- **Bluespec System Verilog**

Challenges for ORN / SORN Hardware

- How can congestion control be implemented for ORNs/SORNs?
- Can (semi-)oblivious routing be implemented with low clock cycle overhead?
- How can on-chip memory consumption be minimized for better scalability?

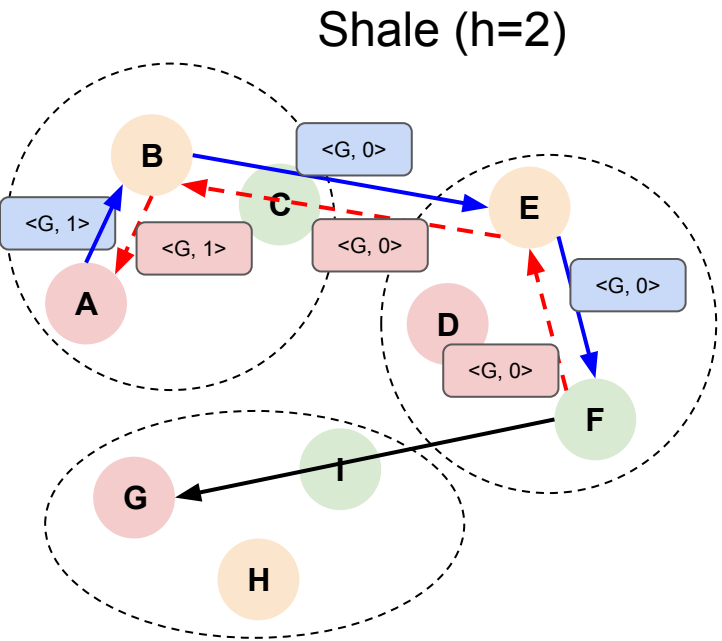
Token-based Congestion Control

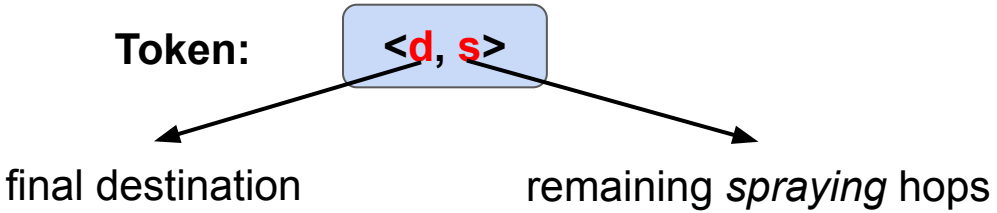




Flow: A->G

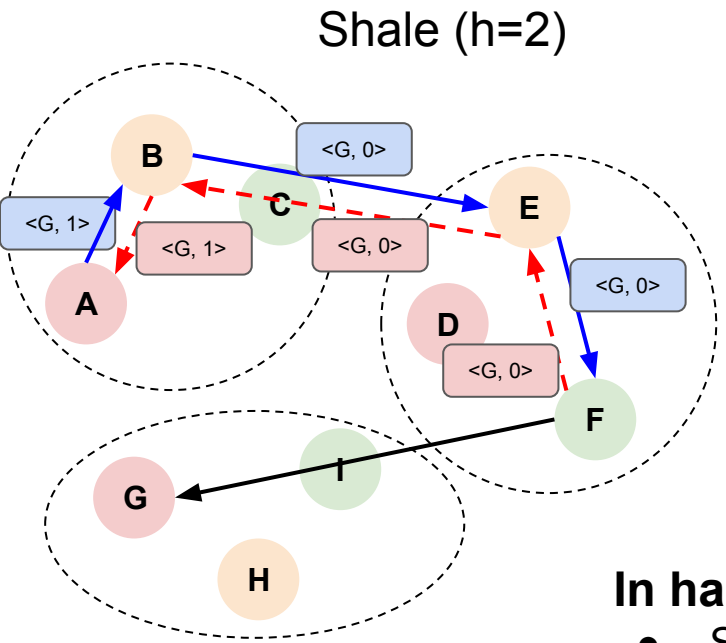
Token-based Congestion Control

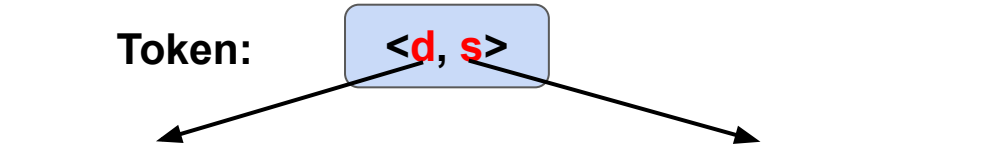




Flow: A->G

Token-based Congestion Control





Flow: A->G

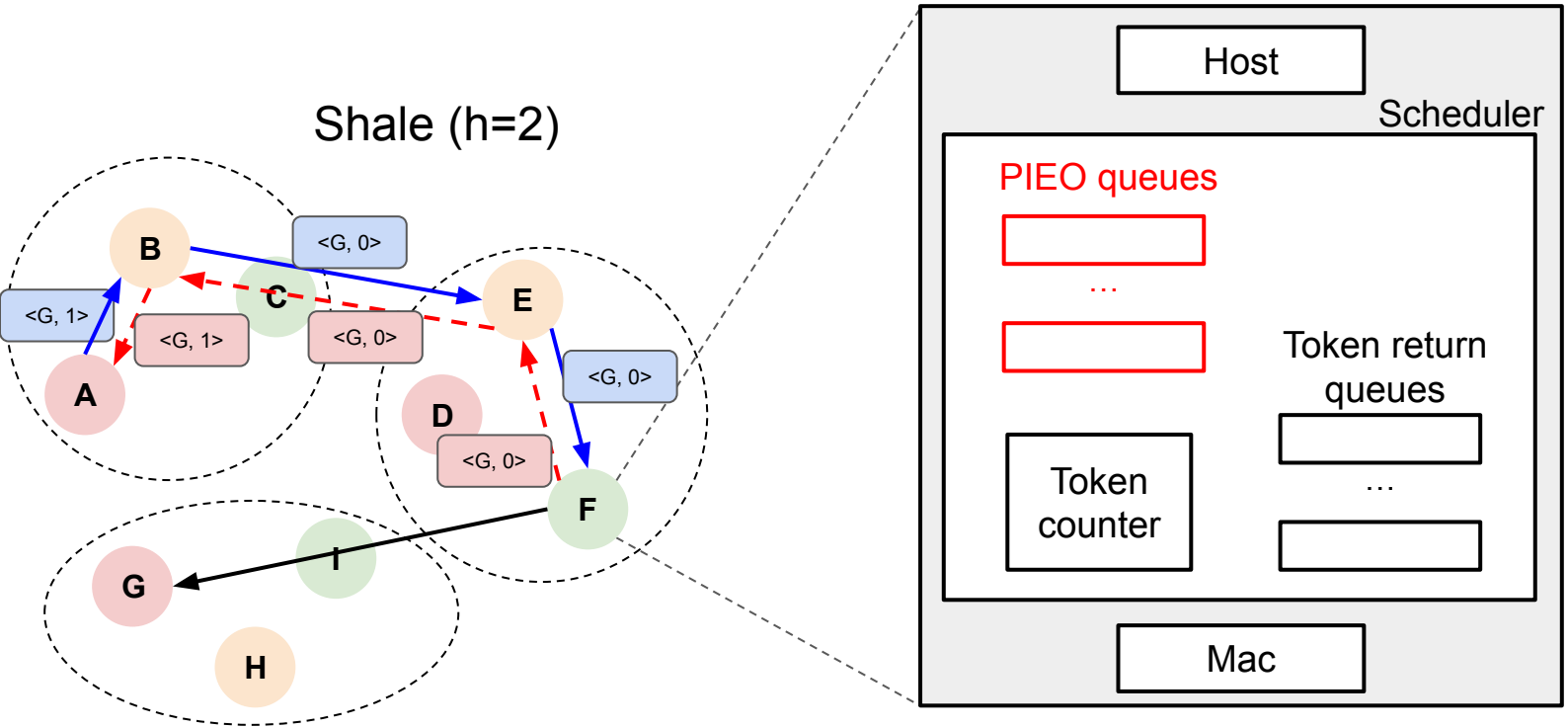
Expend token

Replenish token

In hardware

- Store # of remaining tokens (per-neighbor)
- Store which tokens to replenish (per-neighbor)
- Find first cell that can be sent given the current token count

Token-based CC in Hardware



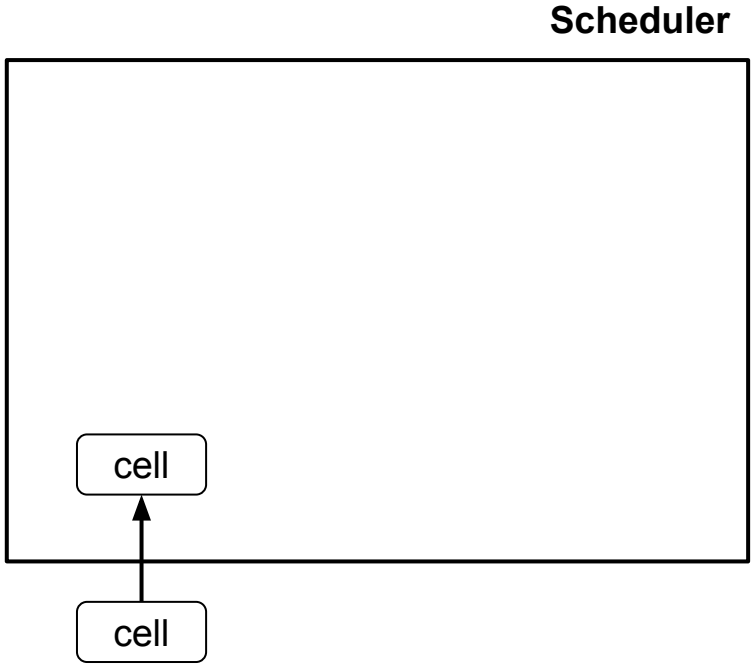
PIEO (Push-In, Extract-Out) Queues:

- Dequeue the first eligible cell in 3 clock cycles

Challenges for ORN / SORN Hardware

- How can congestion control be implemented for ORNs/SORNs?
- Can (semi-)oblivious routing be implemented with low clock cycle overhead?
- How can on-chip memory consumption be minimized for better scalability?

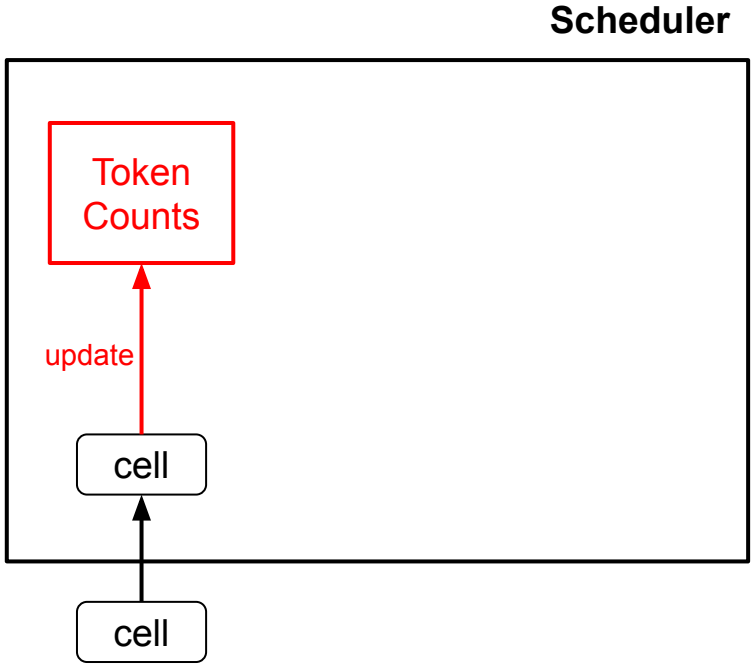
RX Path for ORNs / SORNs: 2 cycles



Cycle 1:

- Parse tokens in the received cell
- Determine if the cell should be sprayed, forwarded or received
- Calculate cell’s next hop

RX Path for ORNs / SORNs: 2 cycles



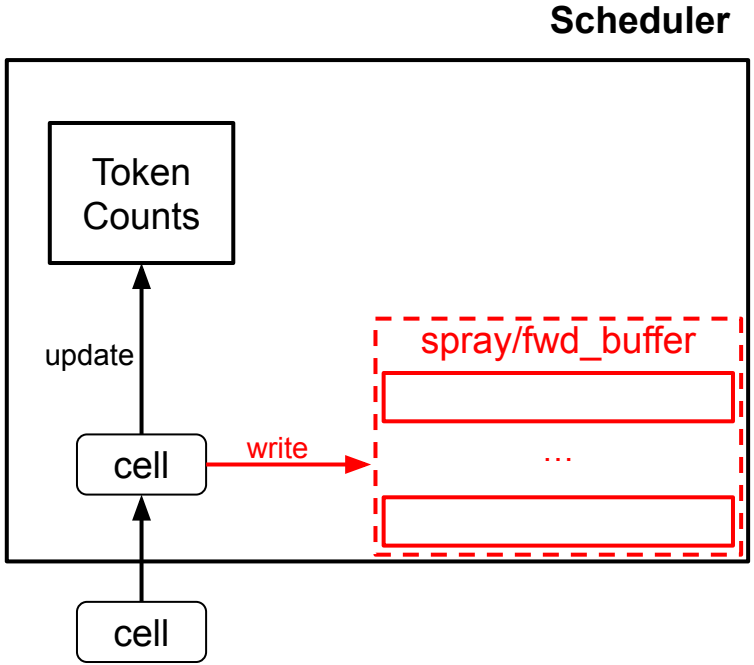
Cycle 1:

- Parse tokens in the received cell
- Determine if the cell should be sprayed, forwarded or received
- Calculate cell's next hop

Cycle 2:

- Update token counts

RX Path for ORNs / SORNs: 2 cycles



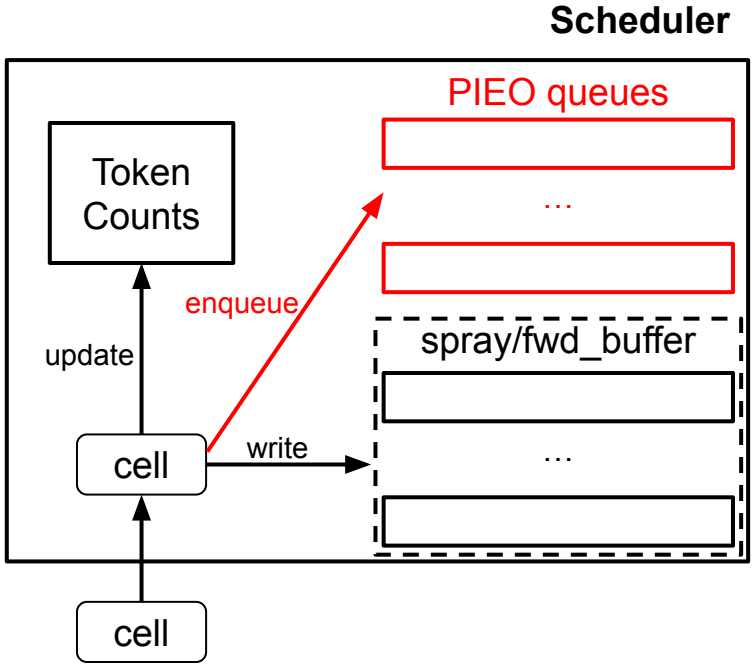
Cycle 1:

- Parse tokens in the received cell
- Determine if the cell should be sprayed, forwarded or received
- Calculate cell's next hop

Cycle 2:

- Update token counts
- Write cell data to buffers in DRAM

RX Path for ORNs / SORNs: 2 cycles



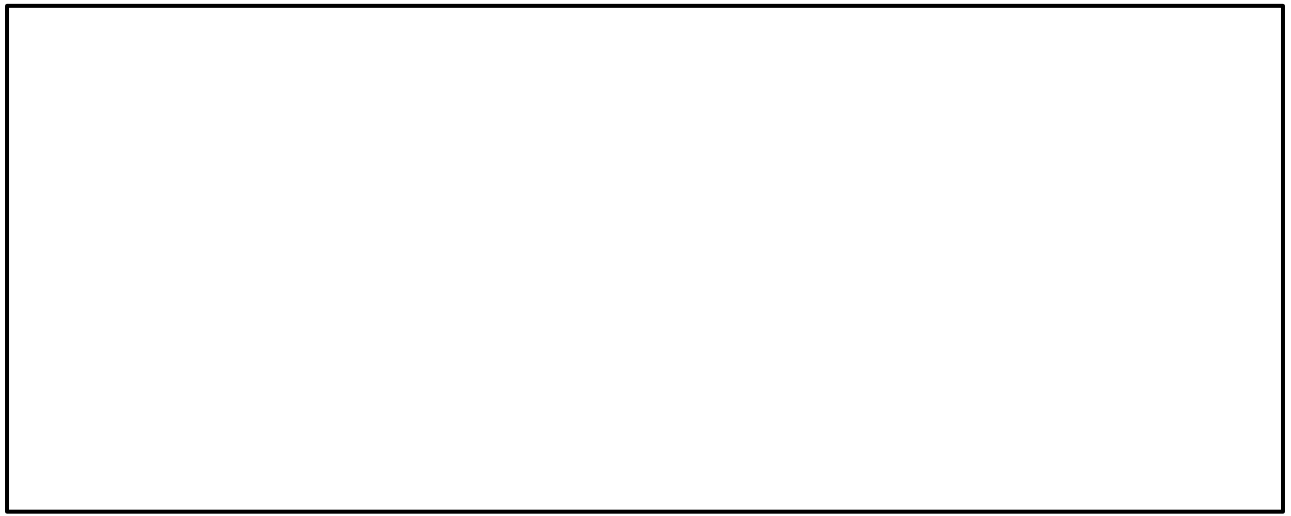
Cycle 1:

- Parse tokens in the received cell
- Determine if the cell should be sprayed, forwarded or received
- Calculate cell's next hop

Cycle 2:

- Update token counts
- Write cell data to buffers in DRAM
- Enqueue flow metadata to PIEO queue

TX Path for ORNs / SORNs: 7 cycles

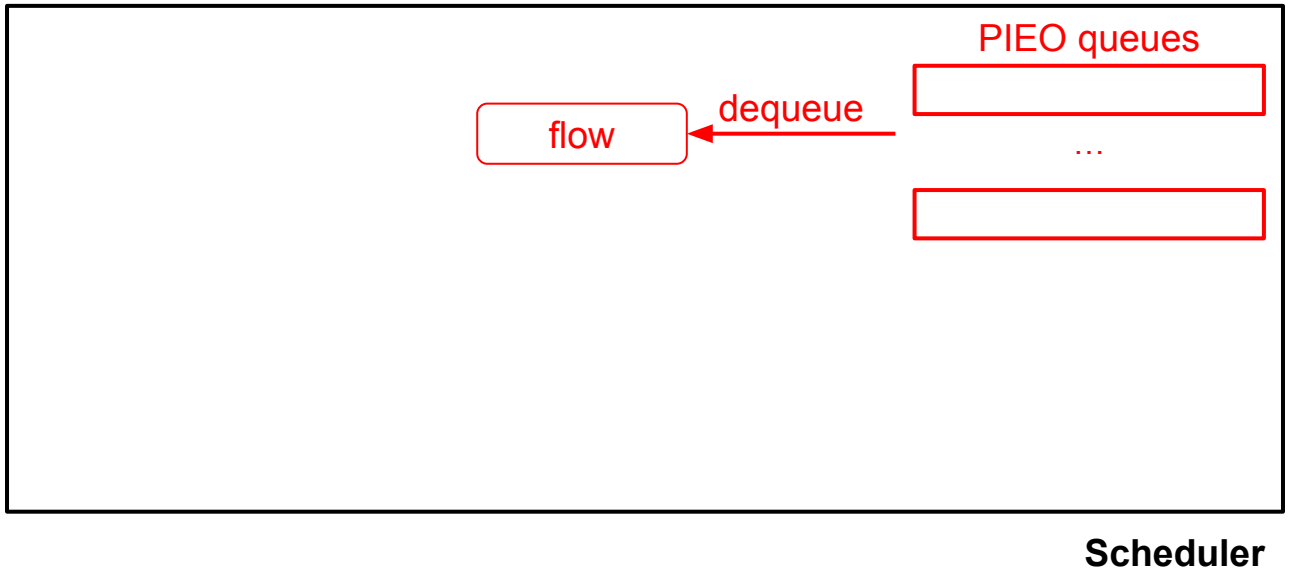


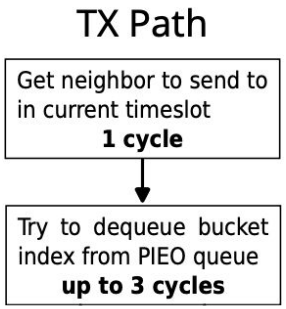
Scheduler

TX Path

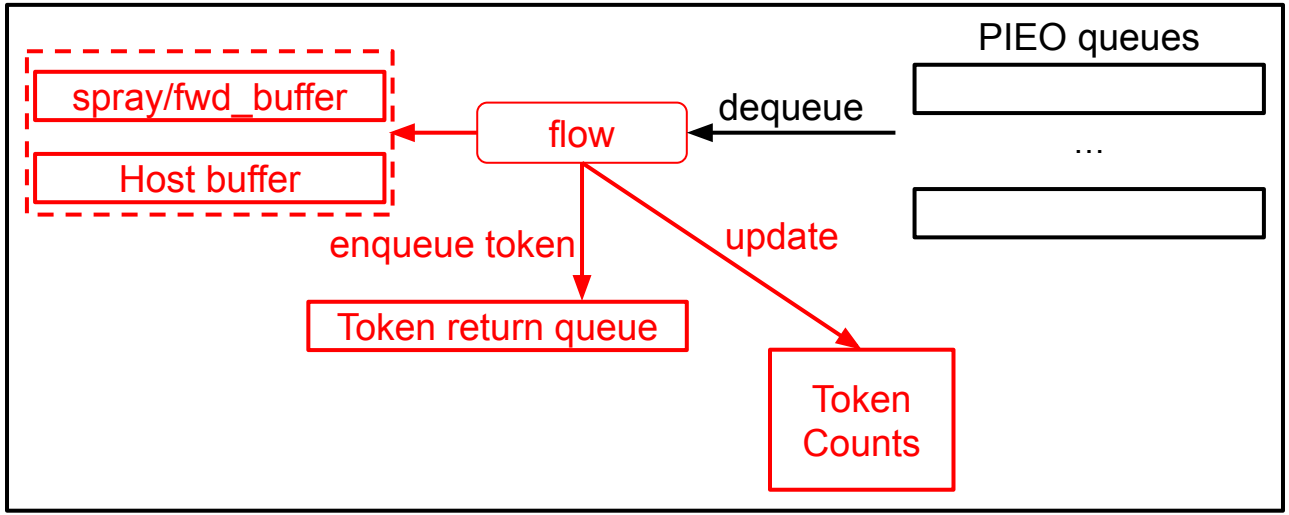
Get neighbor to send to in current timeslot
1 cycle

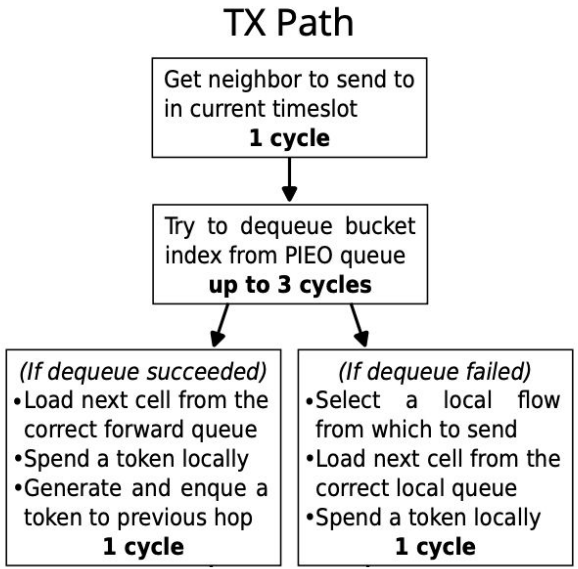
TX Path for ORNs / SORNs: 7 cycles



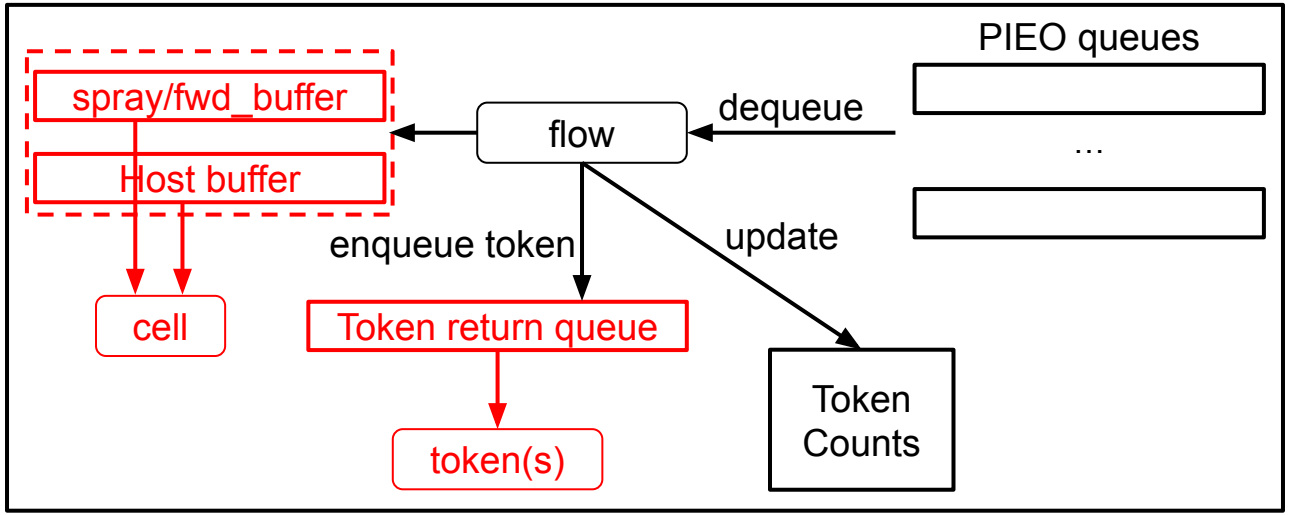


TX Path for ORNs / SORNs: 7 cycles

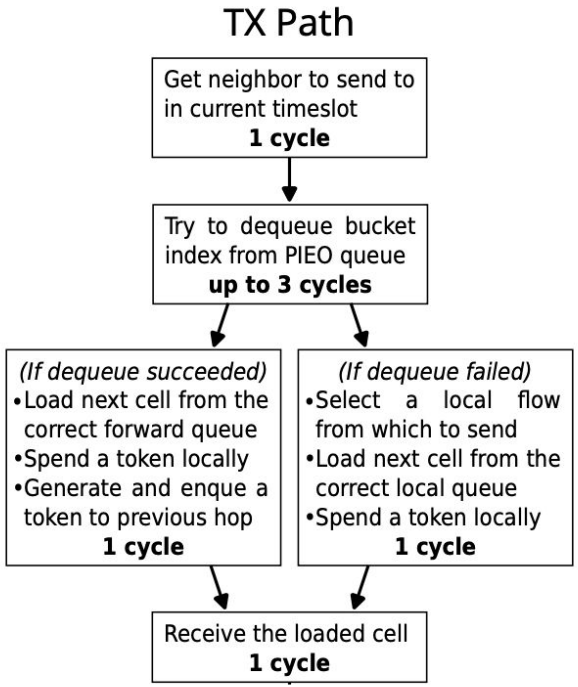




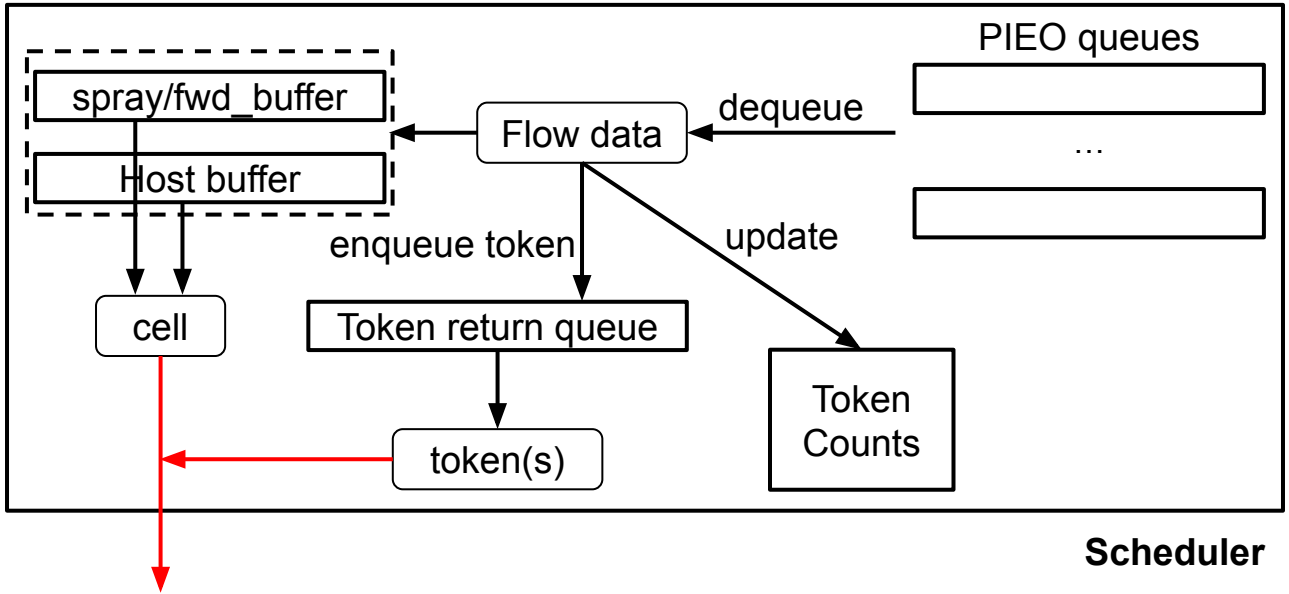
TX Path for ORNs / SORNs: 7 cycles

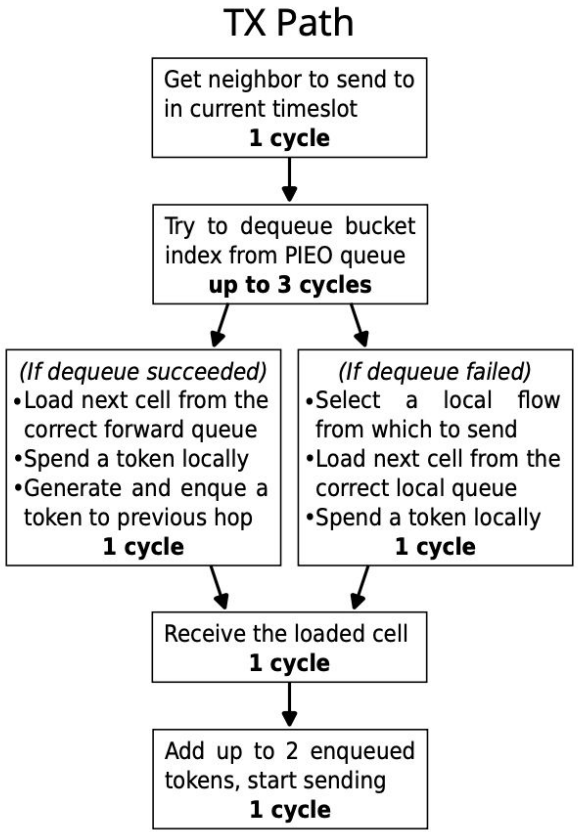


Scheduler



TX Path for ORNs / SORNs: 7 cycles





Challenges for ORN / SORN Hardware

- How can congestion control be implemented for ORNs/SORNs?
- Can (semi-)oblivious routing be implemented with low clock cycle overhead?
- How can on-chip memory consumption be minimized for better scalability?

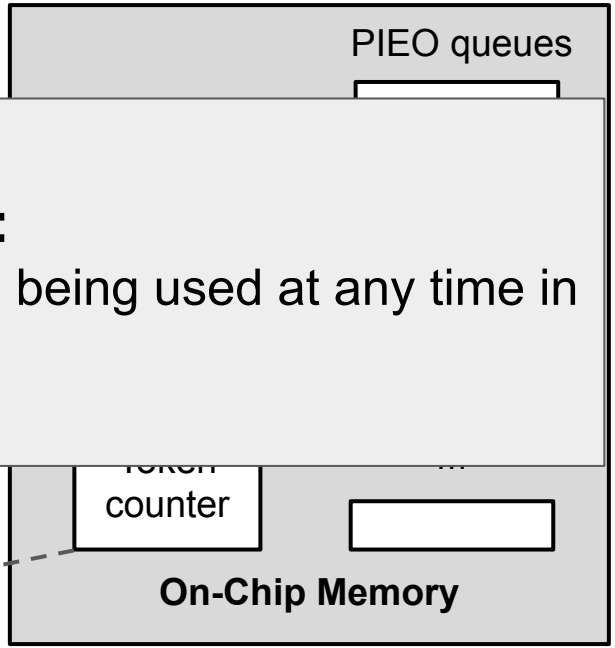
Unoptimized On-chip Memory Consumption

Shal
Mem
$\mathcal{O}(h^2$

SOR
Mem

$$\mathcal{O}(N(\frac{N}{k}+k-2)\log(N)) \approx \mathcal{O}(N^{\frac{3}{2}}\log(N))$$

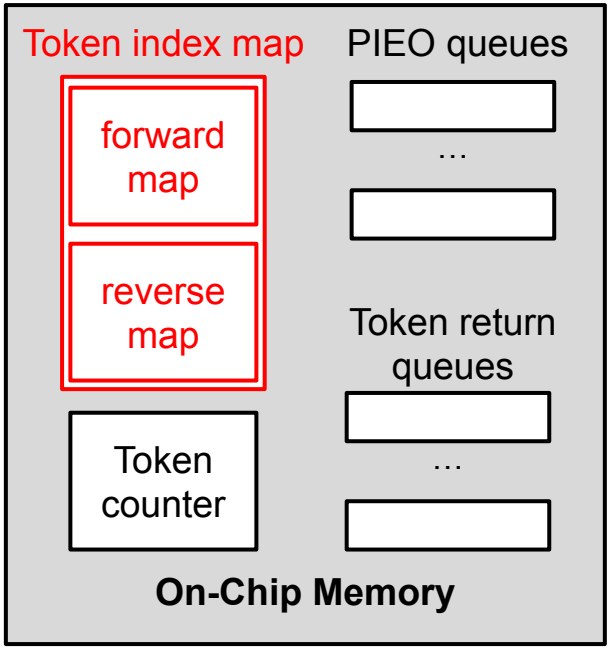
Observation:
There are only a small number of tokens being used at any time in any node



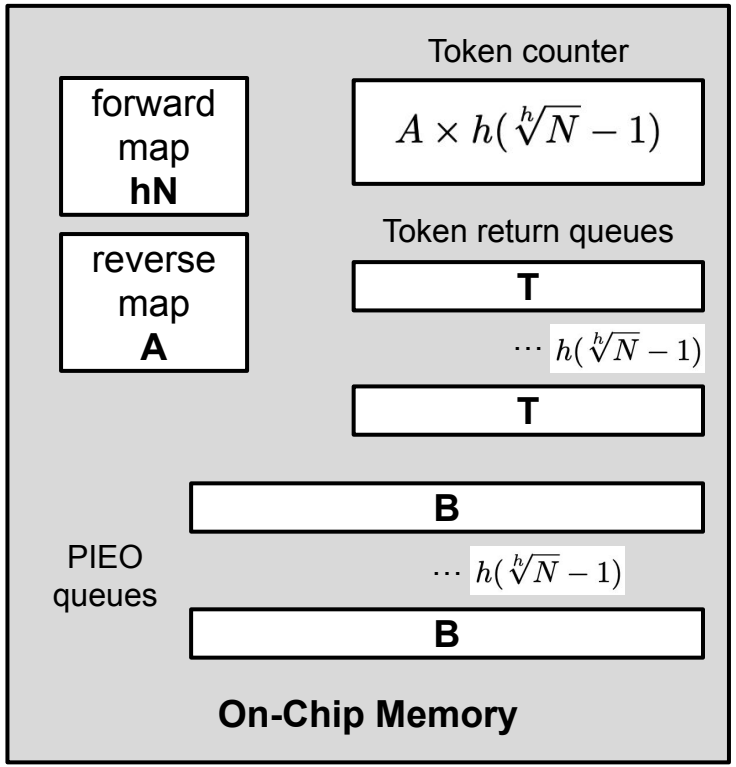
Optimization for On-chip Memory Consumption

Keep track of **only** the tokens currently being used

- Limit the number of *active* tokens to ***A***
- Store the mapping and reverse mapping of token IDs to active token indices



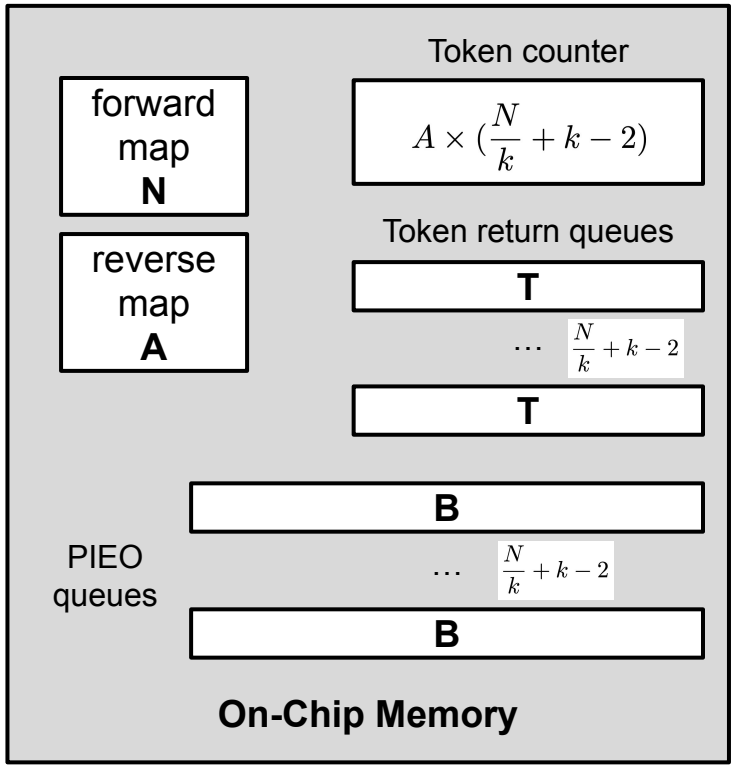
Memory Layout for Shale



Assuming each node has at most **A** active tokens, at most **P** token IDs in each PIEO queue, and at most **T** tokens to return to neighbors, the total on-chip memory requirement is:

$$\mathcal{O}(h(\sqrt[h]{N} - 1)(A\log(A) + P + T) + hN + A)$$

Memory Layout for SORN



Assuming each node has at most **A** active tokens, at most **P** token IDs in each PIEO queue, and at most **T** tokens to return to neighbors, the total on-chip memory requirement is:

$$\mathcal{O}((\frac{N}{k} + k - 2)(A\log(A) + P + T) + N + A)$$

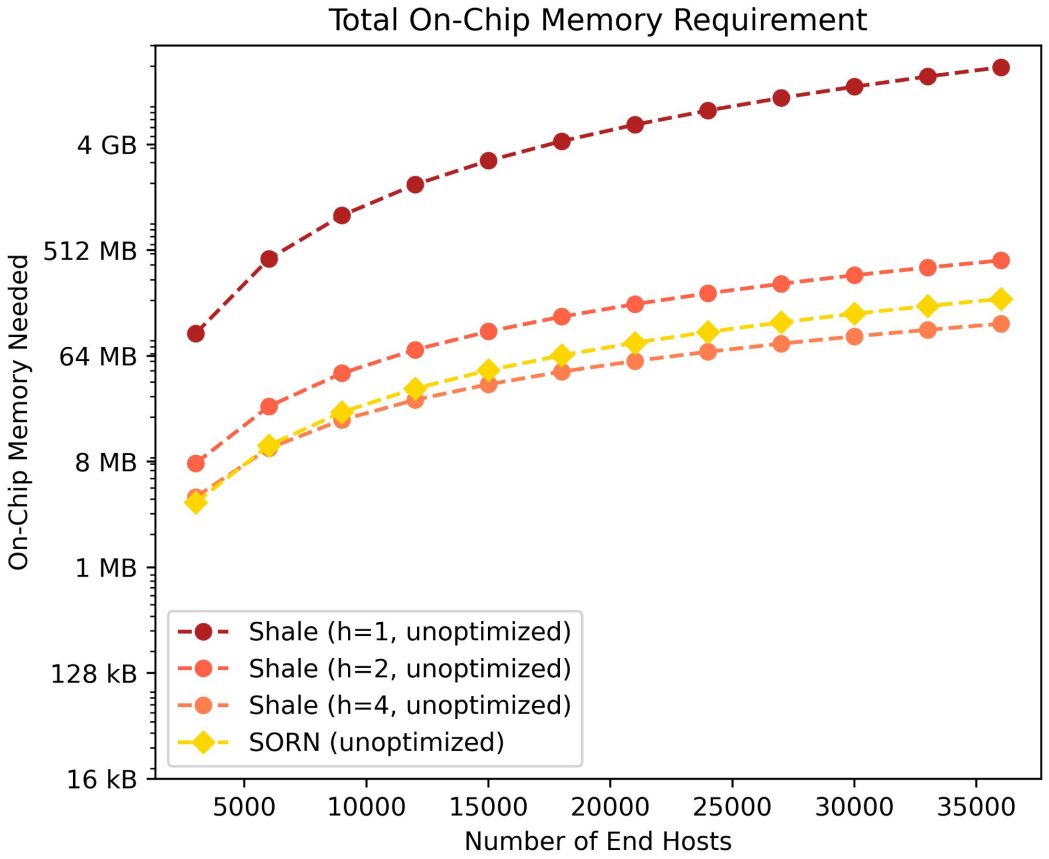
On-chip memory scalability

Shale (unoptimized):

$$\mathcal{O}(h(\sqrt[h]{N}-1)(hN\log(hN)+P+T))$$

SORN (unoptimized):

$$\mathcal{O}((\frac{N}{k}+k-2)(N\log(N)+P+T))$$



On-chip memory scalability

Shale (unoptimized):

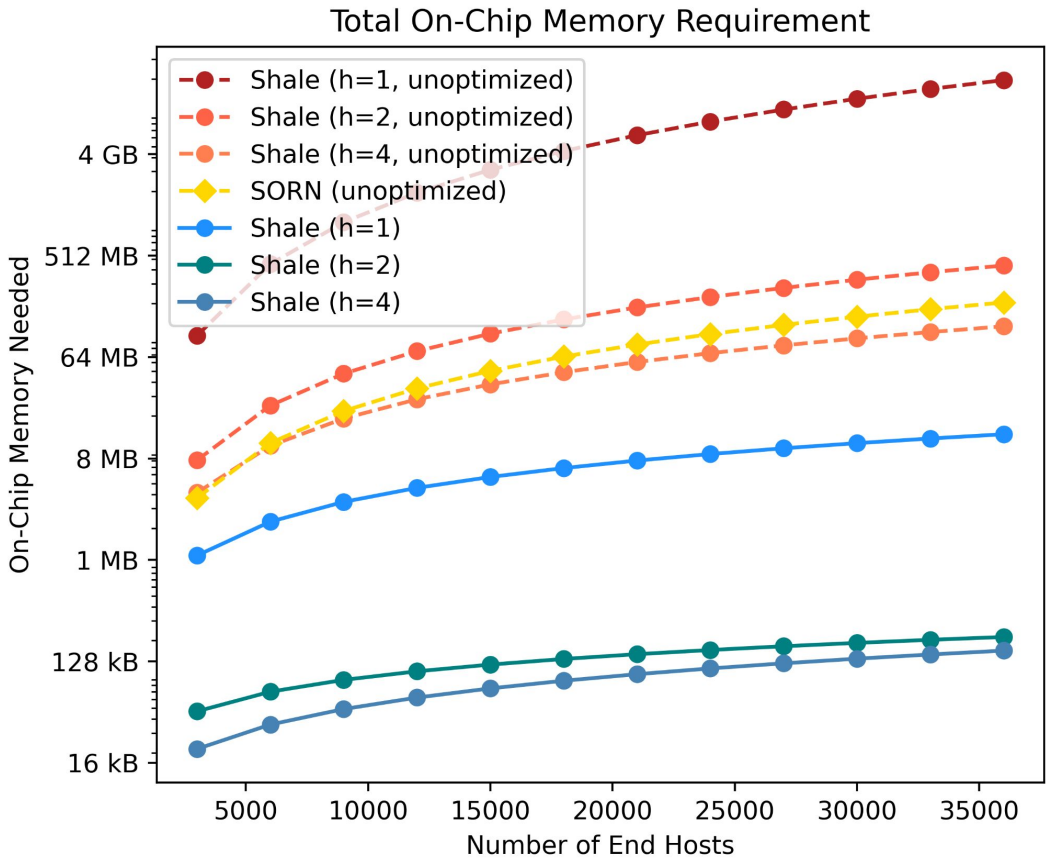
$$\mathcal{O}(h(\sqrt[h]{N}-1)(hN\log(hN)+P+T))$$

SORN (unoptimized):

$$\mathcal{O}((\frac{N}{k}+k-2)(N\log(N)+P+T))$$

Shale:

$$\mathcal{O}(h(\sqrt[h]{N}-1)(A\log(A)+P+T)+hN+A)$$



On-chip memory scalability

Shale (unoptimized):

$$\mathcal{O}(h(\sqrt[h]{N}-1)(hN\log(hN)+P+T))$$

SORN (unoptimized):

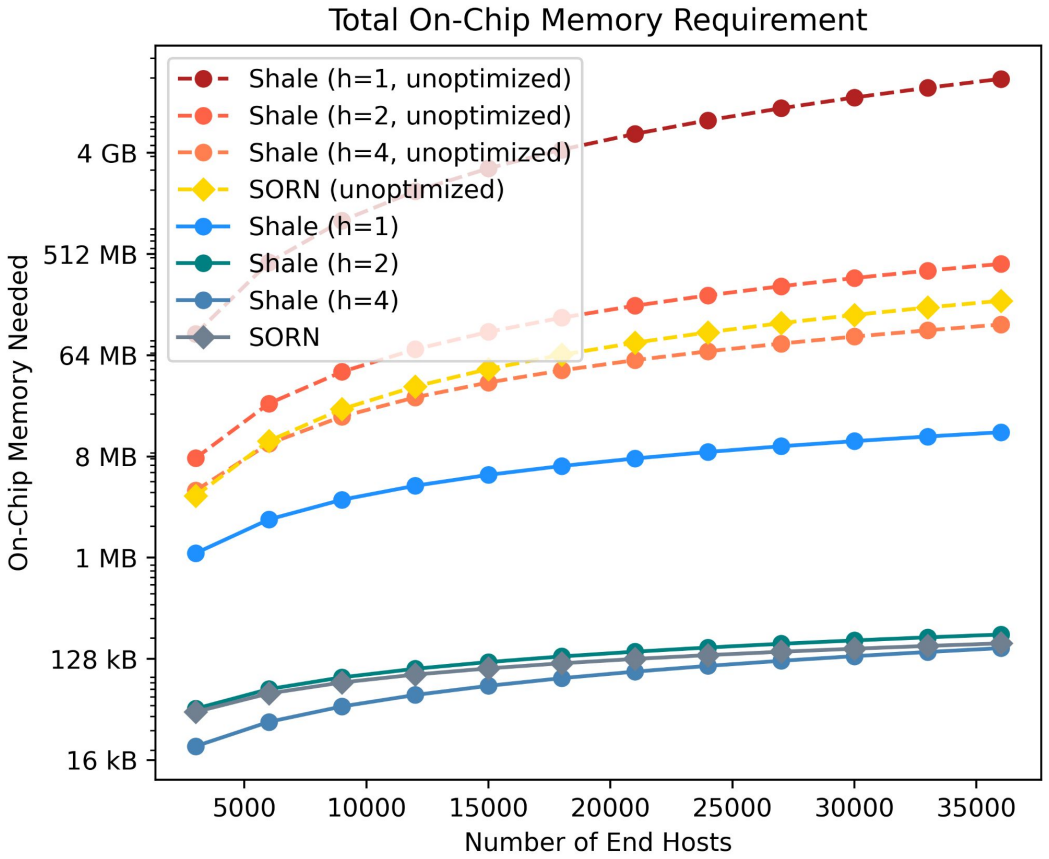
$$\mathcal{O}((\frac{N}{k}+k-2)(N\log(N)+P+T))$$

Shale:

$$\mathcal{O}(h(\sqrt[h]{N}-1)(A\log(A)+P+T)+hN+A)$$

SORN:

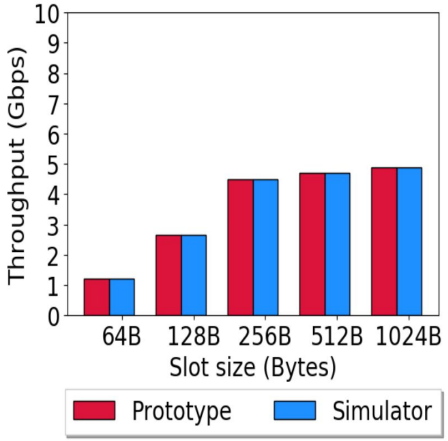
$$\mathcal{O}((\frac{N}{k}+k-2)(A\log(A)+P+T)+N+A)$$

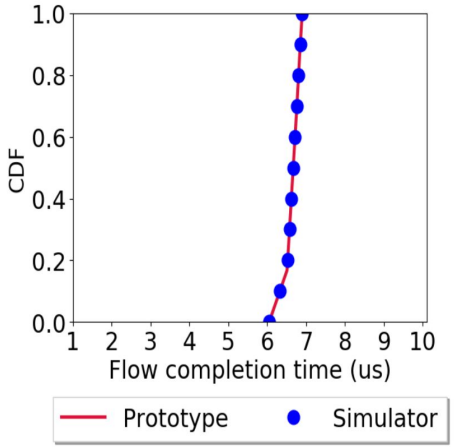


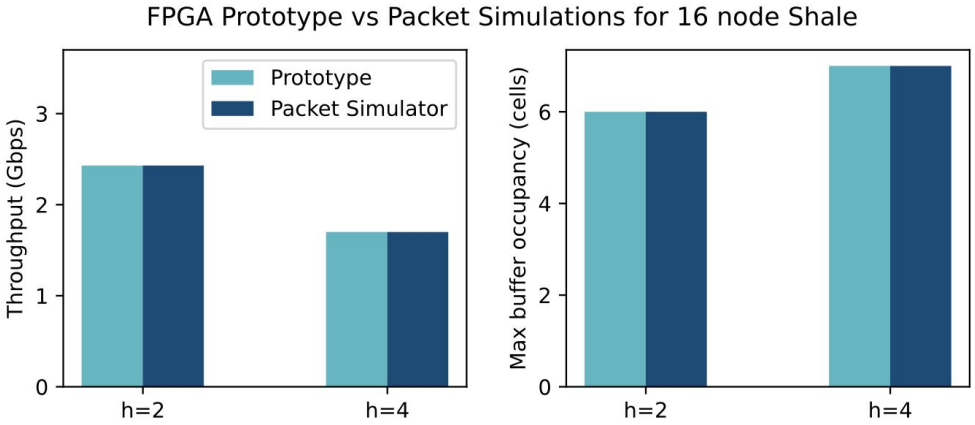
Hardware Prototype vs Packet Simulator

Shoal (Shale, h=1)

- Throughput (left)
- Flow completion time (right)







Shale(h=2, 4)

- Throughput (left)
- Queue occupancy (right)

Conclusion

- (S)ORNs can be prototyped in hardware using FPGAs.
- Routing and congestion control can be implemented and optimized in hardware.
- With optimizations, (S)ORN hardware achieves good on-chip memory scalability.
- Hardware prototypes achieve the same performance as their respective packet simulators.

References

[1] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. 2019. Shoal: a network architecture for disaggregated racks. In Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation (NSDI'19). USENIX Association, USA, 255–270.

[2] Daniel Amir, Nitika Saran, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. 2024. Shale: A Practical, Scalable Oblivious Reconfigurable Network. In Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24). Association for Computing Machinery, New York, NY, USA, 449–464.

[3] Tegan Wilson, Daniel Amir, Nitika Saran, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. 2024. Breaking the VLB Barrier for Oblivious Reconfigurable Networks. In Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC 2024). Association for Computing Machinery, New York, NY, USA, 1865–1876.

[4] Nitika Saran, Daniel Amir, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. 2024. Semi-Oblivious Reconfigurable Datacenter Networks. In Proceedings of the 23rd ACM Workshop on Hot Topics in Networks (HotNets '24). Association for Computing Machinery, New York, NY, USA, 150–158.

[5] Daniel Amir, Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, Robert Kleinberg, and Rachit Agarwal. 2022. Optimal oblivious reconfigurable networks. In Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022). Association for Computing Machinery, New York, NY, USA, 1339–1352.

[6] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. 2017. RotorNet: A Scalable, Low-complexity, Optical Datacenter Network. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17). Association for Computing Machinery, New York, NY, USA, 267–280.

[7] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, and Hugh Williams. 2020. Sirius: A Flat Datacenter Network with Nanosecond Optical Switching. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20). Association for Computing Machinery, New York, NY, USA, 782–797.