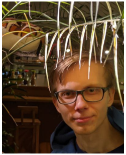
Polylog-Competitive Deterministic Local Routing and Scheduling


Bernhard[1,2] Haeupler


Shyamal[3] Patel


Antti[2] **Roeyskoe**


Cliff[3] Stein

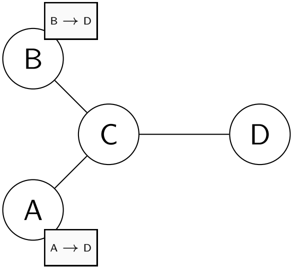

Goran[4] Zuzic

1: INSAIT, 2: ETH Zürich, 3: Columbia University, 4: Google Research

Packet Routing

Packet Routing Problem

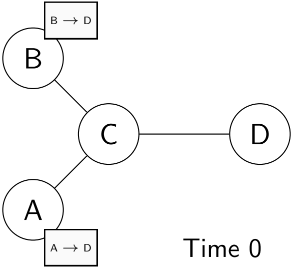
- Undirected graph
- Packets with set start and destination vertices



Packet Routing

Packet Routing Problem

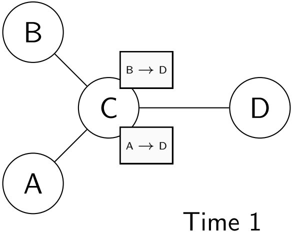
- Undirected graph
- Packets with set start and destination vertices
- Time step: each vertex can forward a packet over each incident edge



Packet Routing

Packet Routing Problem

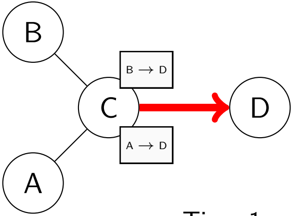
- Undirected graph
- Packets with set start and destination vertices
- Time step: each vertex can forward a packet over each incident edge



Packet Routing

Packet Routing Problem

- Undirected graph
- Packets with set start and destination vertices
- Time step: each vertex can forward a packet over each incident edge

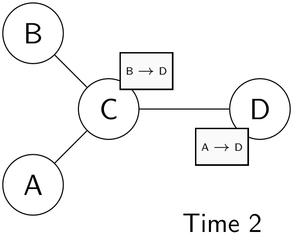


Time 1

Packet Routing

Packet Routing Problem

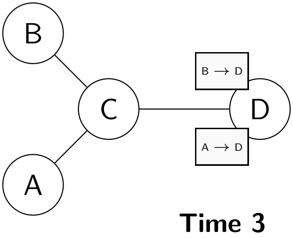
- Undirected graph
- Packets with set start and destination vertices
- Time step: each vertex can forward a packet over each incident edge



Packet Routing

Packet Routing Problem

- Undirected graph
- Packets with set start and destination vertices
- Time step: each vertex can forward a packet over each incident edge
- Goal: minimize time to *deliver* all packets (*completion time*)



Routing Tables

Routing Table Problem

Given undirected graph, design *routing tables* that solve the packet routing problem *competitively*

Routing Tables

Routing Table Problem

Given undirected graph, design *routing tables* that solve the packet routing problem *competitively*

Local forwarding rules for each vertex

- Which packets to forward over which edges
- Based only on the packets at **that vertex**, at **that time**

Routing Tables

Routing Table Problem

Given undirected graph, design *routing tables* that solve the packet routing problem *competitively*

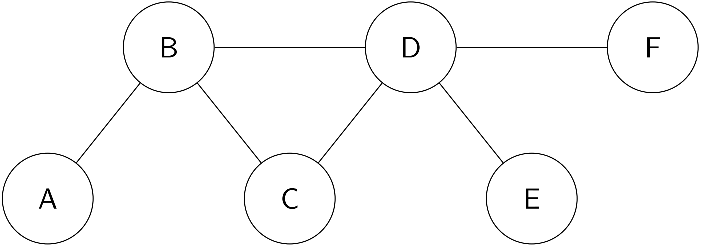
Local forwarding rules for each vertex

- Which packets to forward over which edges
- Based only on the packets at **that vertex**, at **that time**

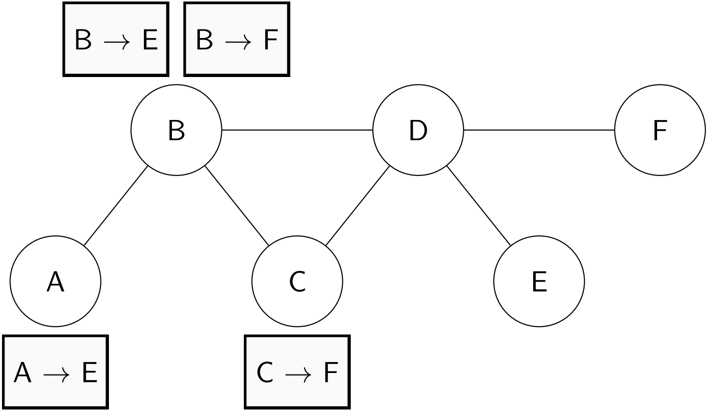
That solve packet routing *competitively*

- C-competitive: for **every** packet routing instance, the forwarding rules achieve completion time $C \cdot \mathrm{OPT}_{\mathrm{global}}$

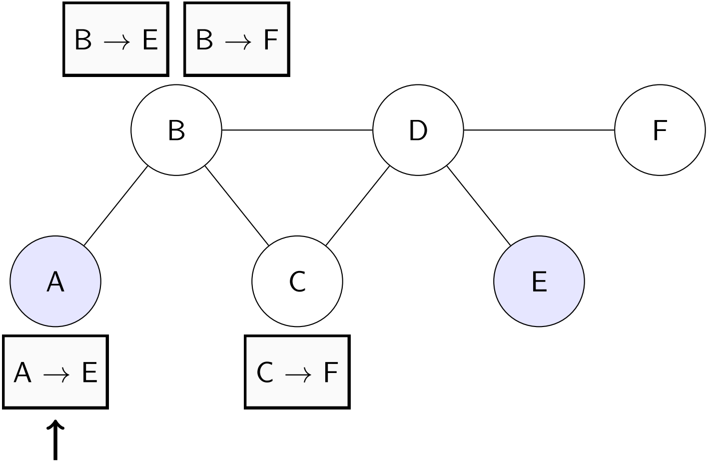
Example



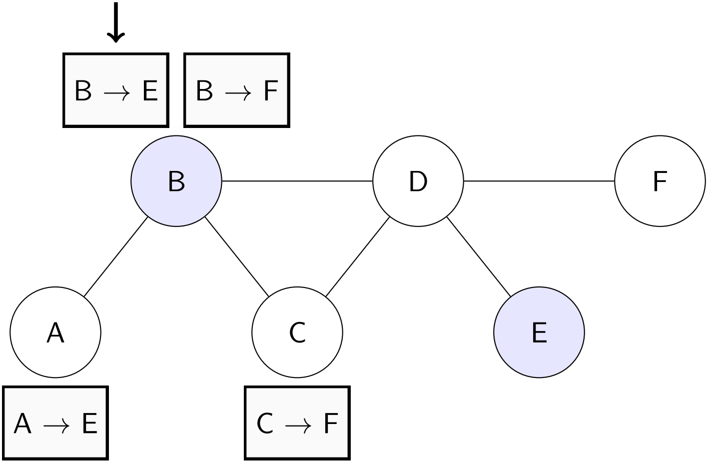
Example: Offline Optimum



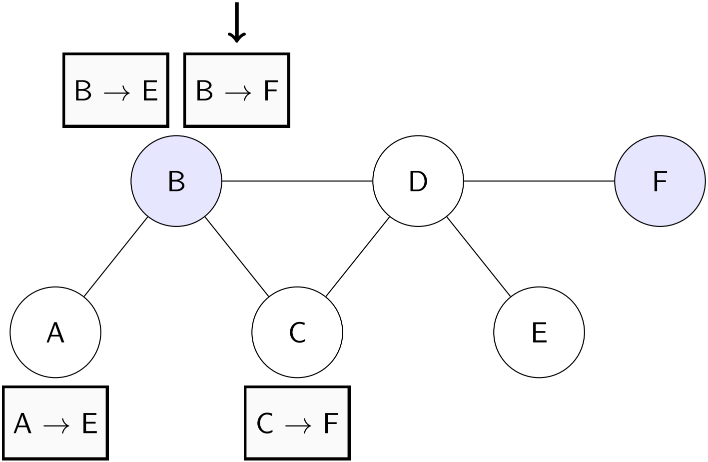
Example: Offline Optimum



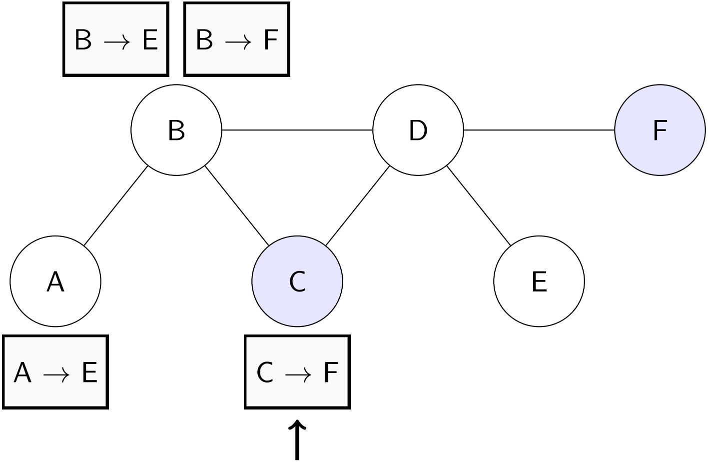
Example: Offline Optimum



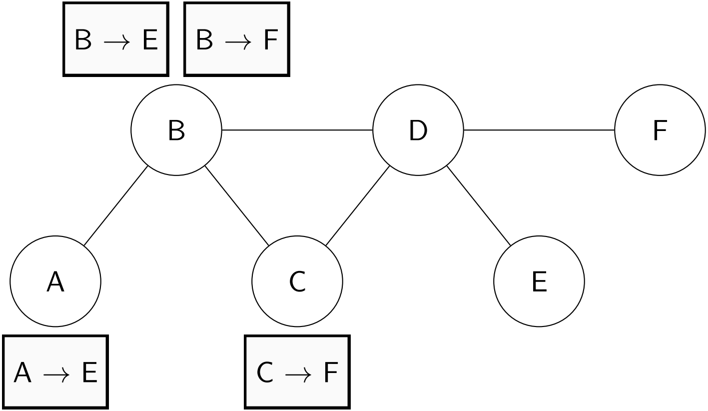
Example: Offline Optimum



Example: Offline Optimum

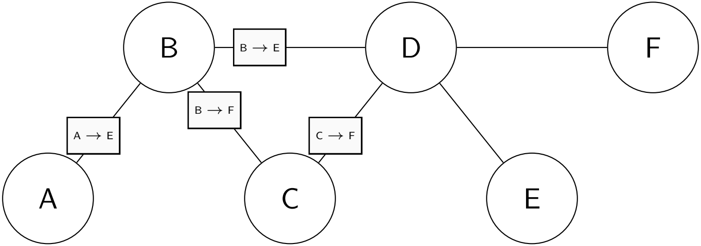


Example: Offline Optimum



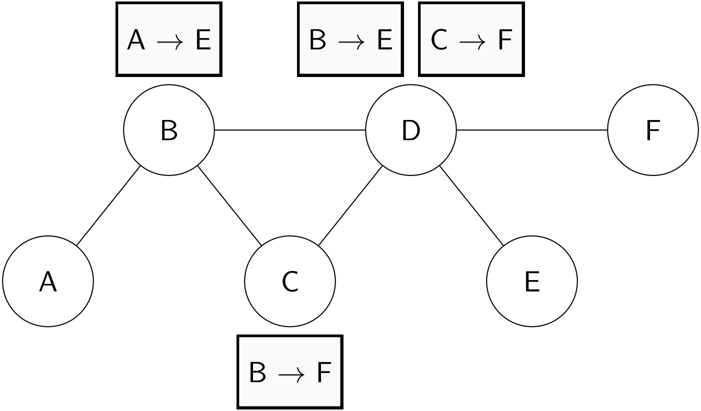
Time 0

Example: Offline Optimum



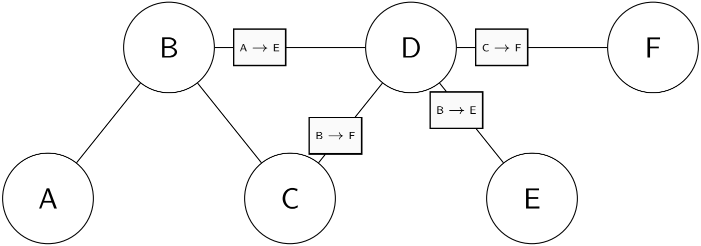
Time 0.1

Example: Offline Optimum



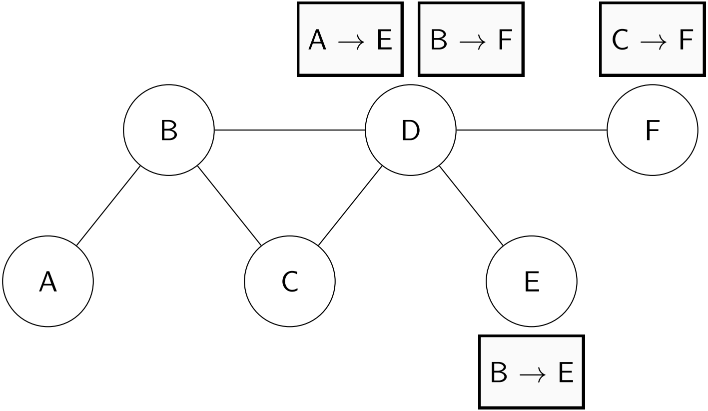
Time 1

Example: Offline Optimum



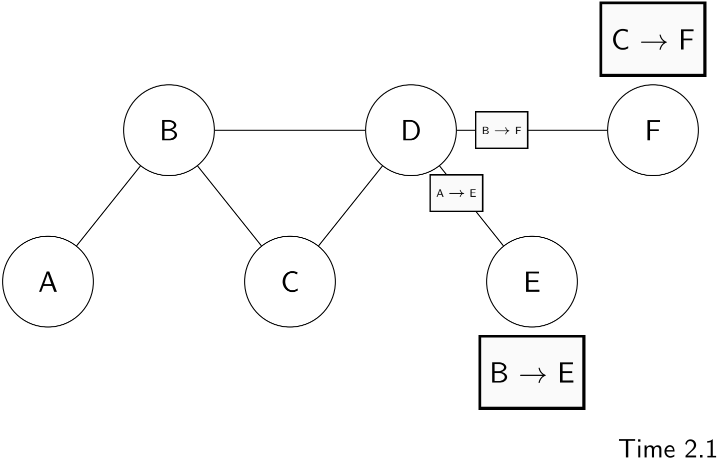
Time 1.1

Example: Offline Optimum

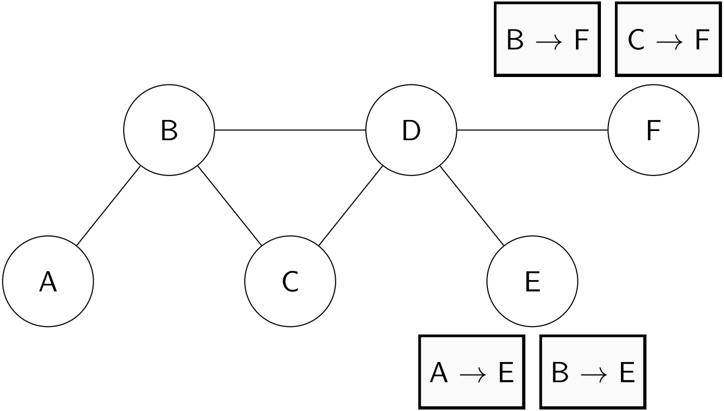


Time 2

Example: Offline Optimum

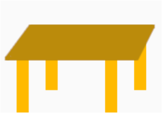


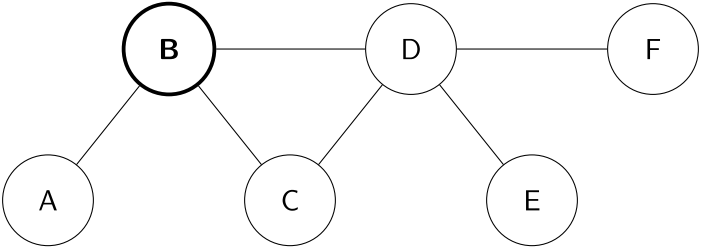
Example: Offline Optimum



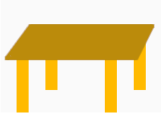
Time 3

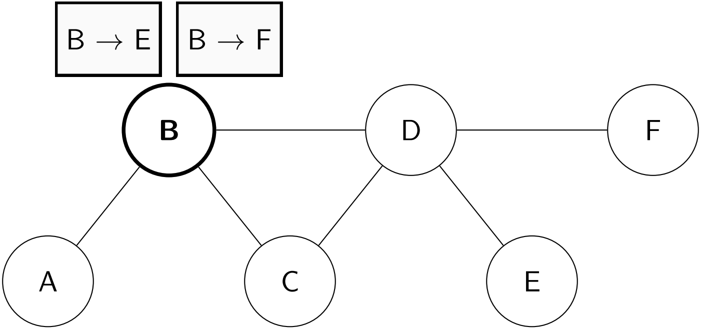
Example: B's Perspective





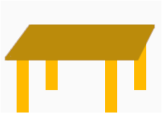
Example: B's Perspective



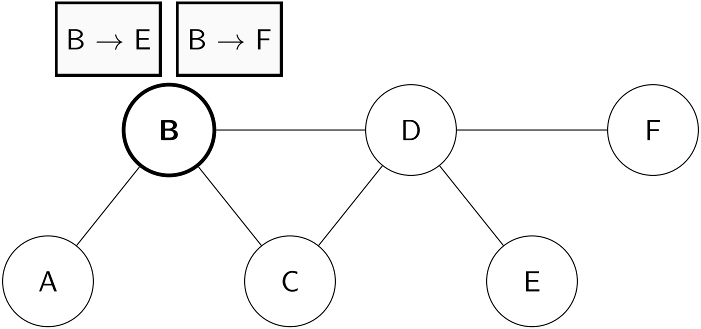


Time 0

Example: B's Perspective

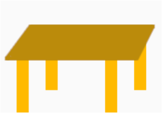


Time 0
([B $\to$ F], [B $\to$ E])



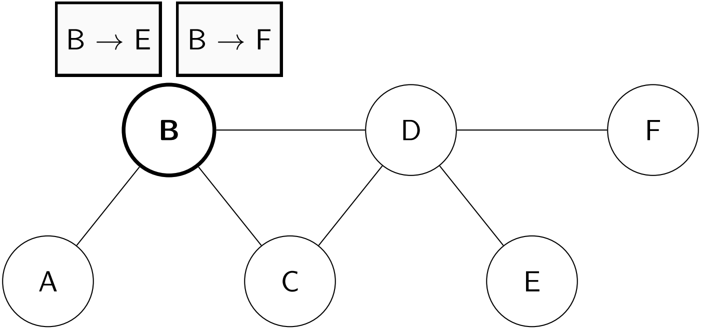
Time 0

Example: B's Perspective



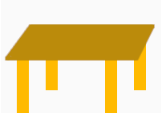
→

[B → F]: Forward to D
[B → E]: Wait

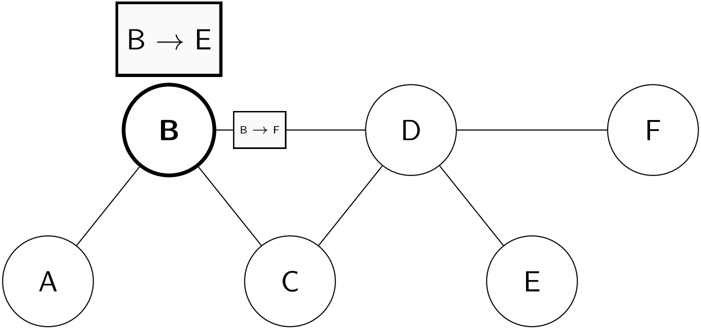


Time 0

Example: B's Perspective

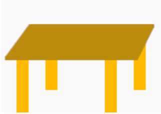


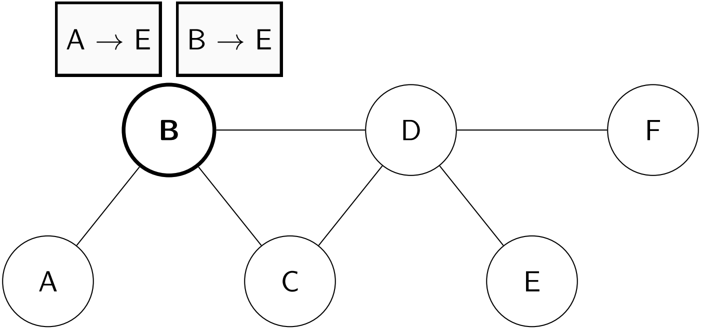
[B → F]: Forward to D
[B → E]: Wait



Time 0.1

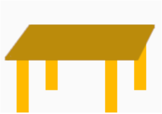
Example: B's Perspective





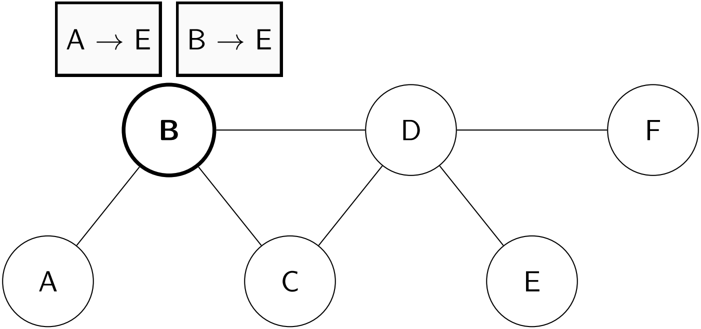
Time 1

Example: B's Perspective



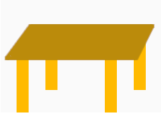
Time 1

([A → E], [B → E])

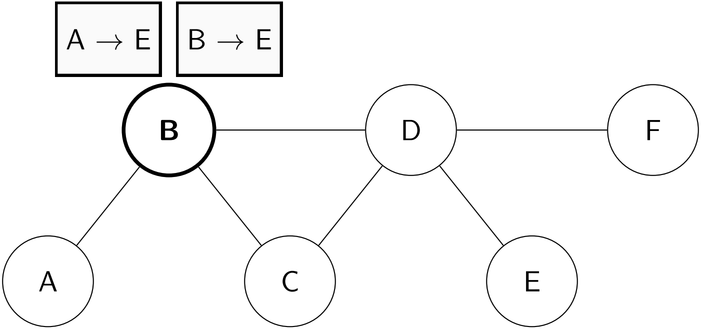


Time 1

Example: B's Perspective

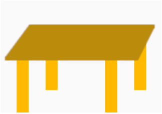


[A → E]: Forward to D
[B → E]: Wait

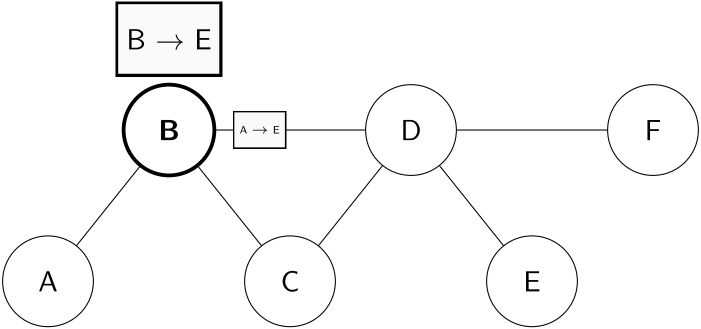


Time 1

Example: B's Perspective

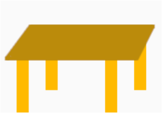


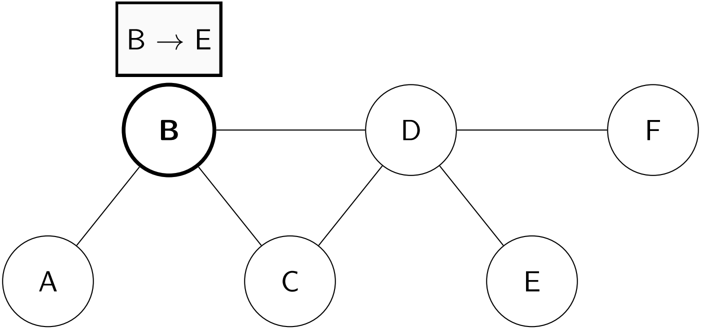
[A $\rightarrow$ E]: Forward to D
[B $\rightarrow$ E]: Wait



Time 1.1

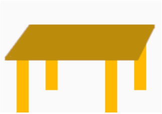
Example: B's Perspective



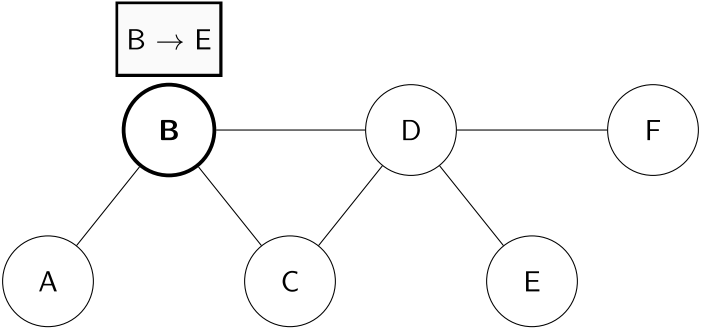


Time 2

Example: B's Perspective

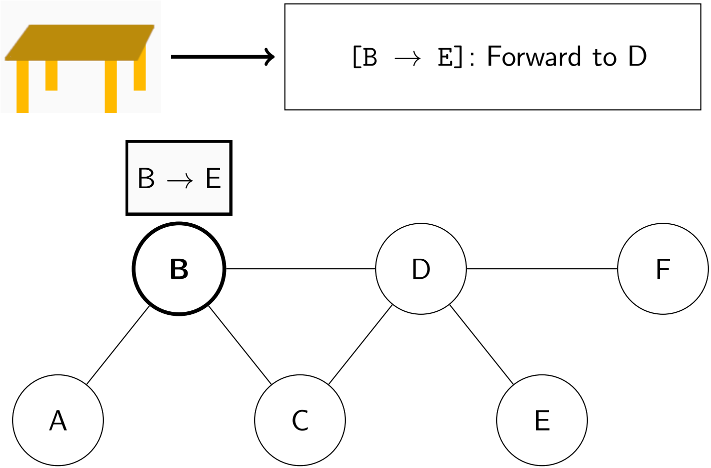


Time 2
([B $\rightarrow$ E])



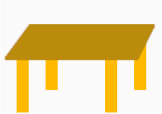
Time 2

Example: B's Perspective

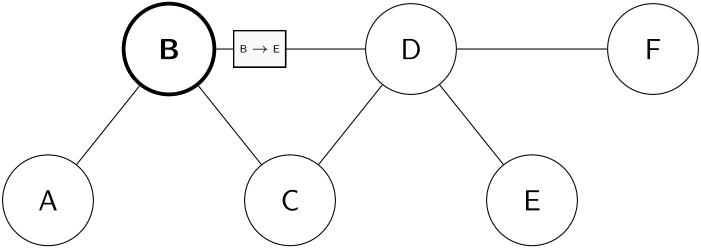


Time 2

Example: B's Perspective

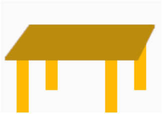


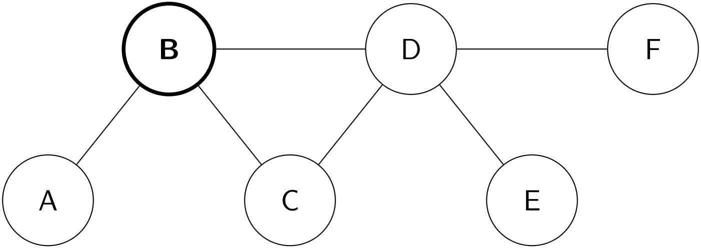
[B → E]: Forward to D



Time 2.1

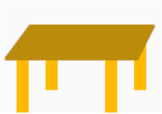
Example: B's Perspective

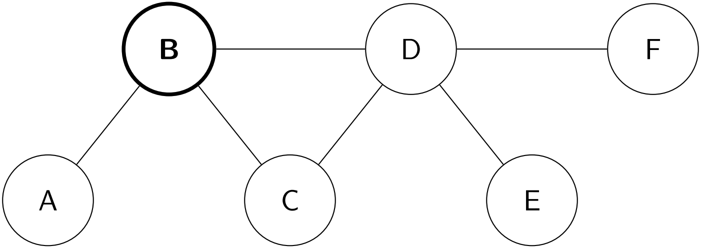




Time 3

Example: B's Perspective





Time 4

Our Results

Main Result

For every graph, there exists *deterministic* $\mathrm{poly}(\log n)$-competitive routing tables.

Determinism is Great!

Determinism is Great!

- Generating true randomness is slow and expensive
  - Requires specialized, *slow* hardware
  - Routers process *millions* of packets per second, and must be **fast**

Determinism is Great!

- Generating true randomness is slow and expensive
  - Requires specialized, *slow* hardware
  - Routers process *millions* of packets per second, and must be **fast**
- **Guaranteed** not to fail!
  - Even if the chance is low, the Internet going down is **extremely bad**

Talk Overview

So far:

- Problem definition
- Our result

Next:

Talk Overview

So far:

- Problem definition
- Our result

Next:

- Previous state of the art
  - $\rightarrow$ Competitive **randomized** local routing rules

Talk Overview

So far:

- Problem definition
- Our result

Next:

- Previous state of the art
  - → Competitive **randomized** local routing rules
  - . . . and why the approach seems inherently random

Talk Overview

So far:

- Problem definition
- Our result

Next:

- Previous state of the art
  - $\rightarrow$ Competitive **randomized** local routing rules
  - ... and why the approach seems inherently random
- Our *deterministic* approach

Two Subproblems

- Vertices must select

Two Subproblems

- Vertices must select
  - Where to next forward each packet, and

Two Subproblems

- Vertices must select
  - Where to next forward each packet, and
  - Which packet to forward over each edge

Two Subproblems

$$\text{Packet Routing} = \text{Path Selection} \\ + \text{Scheduling}$$

- Vertices must select
  - Where to next forward each packet, and
  - Which packet to forward over each edge

Local Scheduling

Suppose a helpful person has written a path on each packet

Local Scheduling

Suppose a helpful person has written a path on each packet

- *Congestion* C: maximum number of times any edge is used
- *Dilation* D: length of longest path

Local Scheduling

Suppose a helpful person has written a path on each packet

- *Congestion* C: maximum number of times any edge is used
- *Dilation* D: length of longest path

If we forward packets following the fixed paths:

- Completion time $\geq \max(C, D)$

Local Scheduling

Suppose a helpful person has written a path on each packet

- *Congestion* C: maximum number of times any edge is used
- *Dilation* D: length of longest path

If we forward packets following the fixed paths:

- Completion time $\geq \max(C, D)$
- $\mathcal{O}(C + D)$ possible (offline) [LMR94]

Local Scheduling

Suppose a helpful person has written a path on each packet

- *Congestion* C: maximum number of times any edge is used
- *Dilation* D: length of longest path

If we forward packets following the fixed paths:

- Completion time $\geq \max(C, D)$
- $\mathcal{O}(C + D)$ possible (offline) [LMR94]
- Simple local randomized algorithm [LMR94]
  - Randomly delay each packet $\Rightarrow \log(n)$ overhead

Local Scheduling

Suppose a helpful person has written a path on each packet

- *Congestion* C: maximum number of times any edge is used
- *Dilation* D: length of longest path

If we forward packets following the fixed paths:

- Completion time $\geq \max(C, D)$
- $\mathcal{O}(C + D)$ possible (offline) [LMR94]
- Simple local randomized algorithm [LMR94]
  - Randomly delay each packet $\Rightarrow \log(n)$ overhead
- **Deterministic**: nothing $o(CD)$ known

Oblivious Path Selection

Paths cannot be selected based on the global packet set

Oblivious Path Selection

Paths cannot be selected based on the global packet set

$\rightarrow$ Oblivious Path Selection: select based only on source and destination

Oblivious Path Selection

Paths cannot be selected based on the global packet set

$\rightarrow$ Oblivious Path Selection: select based only on source and destination

Classic Result: Oblivious Routing/Räcke Trees [Räc02]

For any graph, there exists a distribution of paths between every pair of vertices, such that for any set of packets, sampling paths from the distribution achieves **congestion** at most $\mathcal{O}(\log n)$ times the global optimum congestion.

Oblivious Path Selection

Paths cannot be selected based on the global packet set

$\rightarrow$ Oblivious Path Selection: select based only on source and destination

Classic Result: Oblivious Routing/Räcke Trees [Räc02]

For any graph, there exists a distribution of paths between every pair of vertices, such that for any set of packets, sampling paths from the distribution achieves **congestion** at most $\mathcal{O}(\log n)$ times the global optimum congestion.

- But paths may be long! ($D \gg \mathrm{OPT}$)

Oblivious Path Selection

Paths cannot be selected based on the global packet set

→ Oblivious Path Selection: select based only on source and destination

Classic Result: Oblivious Routing/Räcke Trees [Räc02]

For any graph, there exists a distribution of paths between every pair of vertices, such that for any set of packets, sampling paths from the distribution achieves **congestion** at most $\mathcal{O}(\log n)$ times the global optimum congestion.

- But paths may be long! ($D \gg \mathrm{OPT}$)

Hop-Bounded Oblivious Routing [GHZ21]

For any graph, there exists a distribution of paths between every pair of vertices, such that for any set of packets, sampling paths from the distribution achieves **completion time** at most $\mathrm{poly}(\log n)$ times the global optimum completion time.

Oblivious Path Selection

Paths cannot be selected based on the global packet set

$\rightarrow$ Oblivious Path Selection: select based only on source and destination

Classic Result: Oblivious Routing/Räcke Trees [Räc02]

For any graph, there exists a distribution of paths between every pair of vertices, such that for any set of packets, sampling paths from the distribution achieves **congestion** at most $\mathcal{O}(\log n)$ times the global optimum congestion.

- But paths may be long! ($D \gg \mathrm{OPT}$)

Hop-Bounded Oblivious Routing [GHZ21]

For any graph, there exists a distribution of paths between every pair of vertices, such that for any set of packets, sampling paths from the distribution achieves **completion time** at most $\mathrm{poly}(\log n)$ times the global optimum completion time.

Randomized Approach: Summary

$$\text{Packet Routing} = \text{Oblivious Path Selection} \\ + \text{Local Scheduling}$$

- Oblivious path selection: **sample** from hop-bounded oblivious routing
- Local scheduling: **sample** a random delay

Deterministic Local Scheduling / Oblivious Path Selection

Deterministic local scheduling:

- **any** strategy achieves $\mathcal{O}(C \cdot D)$
- Nothing better is known

Deterministic oblivious path selection:

- I.E. single fixed path between every vertex pair
- $\Omega(\sqrt{n} \cdot \mathrm{OPT})$ lower bound [KKT90]
- Even on hypercubes!

are deterministic routing tables doomed?

Our Results (again)

Main Result

For every graph, there exists *deterministic* $\mathrm{poly}(\log n)$-competitive routing tables.

But How?

$$\text{Packet Routing} = \text{Oblivious Path Selection} \\ + \text{Local Scheduling}$$

But How?

Packet Routing = ~~Oblivious Path Selection~~

\+ ~~Local Scheduling~~

But How?

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling

But How?

Packet Routing $=$ Sparse Semi-Oblivious Path Selection
$+$ Local Det. Noise-Tolerant Scheduling

[with polynomially many possible paths]

But How?

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling
[with polynomially many possible paths]

- Sparse Semi-Oblivious Path Selection: [ZHR23]

But How?

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling
[with polynomially many possible paths]

- Sparse Semi-Oblivious Path Selection: [ZHR23]
- Local Det. Noise-Tolerant Scheduling: [this paper]

But How?

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling

[with polynomially many possible paths]

- Sparse Semi-Oblivious Path Selection: [ZHR23]
- Local Det. Noise-Tolerant Scheduling: [this paper]
- "+": This talk [and paper]

Sparse Semi-Oblivious Path Selection

Sparse Semi-Oblivious Path Selection

Sparse Semi-Oblivious Path Selection

Sparse Semi-Oblivious Path Selection

- **Multiple** (up to α) fixed paths between each vertex pair

Sparse Semi-Oblivious Path Selection

Sparse Semi-Oblivious Path Selection

- **Multiple** (up to α) fixed paths between each vertex pair
- Guarantee: for any set of packets, can *globally* select one path for each packet among the α options,

Sparse Semi-Oblivious Path Selection

Sparse Semi-Oblivious Path Selection

- **Multiple** (up to α) fixed paths between each vertex pair
- Guarantee: for any set of packets, can *globally* select one path for each packet among the α options, s.t. $C + D \leq \beta \cdot \mathrm{OPT}_{C+D}$

Sparse Semi-Oblivious Path Selection

Sparse Semi-Oblivious Path Selection

- **Multiple** (up to α) fixed paths between each vertex pair
- Guarantee: for any set of packets, can *globally* select one path for each packet among the α options, s.t. $C + D \leq \beta \cdot \mathrm{OPT}_{C+D}$

Existence of Sparse Semi-Oblivious Routings [ZHR23]

For every graph, there exists a $\beta = \mathrm{poly}(\log n)$-competitive $\alpha = \mathcal{O}(\log^2 n)$-sparse semi-oblivious routing.

Sparse Semi-Oblivious Path Selection

Sparse Semi-Oblivious Path Selection

- **Multiple** (up to α) fixed paths between each vertex pair
- Guarantee: for any set of packets, can *globally* select one path for each packet among the α options, s.t. $C + D \leq \beta \cdot \mathrm{OPT}_{C+D}$

Existence of Sparse Semi-Oblivious Routings [ZHR23]

For every graph, there exists a $\beta = \mathrm{poly}(\log n)$-competitive $\alpha = \mathcal{O}(\log^2 n)$-sparse semi-oblivious routing.

- Key issue: how to *locally* select correct path

  ... amongst the $\alpha - 1$ paths of *noise*

Noise-Tolerant Scheduling

What can you hope to deliver in presence of noise?

- A subset $S \subseteq P$ is (α, T)-good, if

Noise-Tolerant Scheduling

What can you hope to deliver in presence of noise?

- A subset $S \subseteq P$ is (α, T)-good, if
  1. It is large: $|S| \geq \frac{1}{\alpha}|P|$
  2. If other packets didn't exist, it could be scheduled $C(S) + D(S) \leq T$

Noise-Tolerant Scheduling

What can you hope to deliver in presence of noise?

- A subset $S \subseteq P$ is (α, T)-good, if
  1. It is large: $|S| \geq \frac{1}{\alpha}|P|$
  2. If other packets didn't exist, it could be scheduled $C(S) + D(S) \leq T$

Noise-Tolerant Scheduling:

Noise-Tolerant Scheduling

What can you hope to deliver in presence of noise?

- A subset $S \subseteq P$ is (α, T)-good, if
  1. It is large: $|S| \geq \frac{1}{\alpha}|P|$
  2. If other packets didn't exist, it could be scheduled $C(S) + D(S) \leq T$

Noise-Tolerant Scheduling:

- Given (α, T) and packet scheduling instance

Noise-Tolerant Scheduling

What can you hope to deliver in presence of noise?

- A subset $S \subseteq P$ is (α, T)-good, if
  1. It is large: $|S| \geq \frac{1}{\alpha}|P|$
  2. If other packets didn't exist, it could be scheduled $C(S) + D(S) \leq T$

Noise-Tolerant Scheduling:

- Given (α, T) and packet scheduling instance
- Deliver **half** of **every** (α, T)-good subset

Local Deterministic Noise-Tolerant Scheduling

Noise-Tolerant Scheduling:

- Given (α, T),
- Deliver **half** of **every** (α, T)-good subset
- In time relative to αT

Local Deterministic Noise-Tolerant Scheduling

Noise-Tolerant Scheduling:

- Given (α, T),
- Deliver **half** of **every** (α, T)-good subset
- In time relative to αT

Harder problem than (regular) scheduling

Local Deterministic Noise-Tolerant Scheduling

Noise-Tolerant Scheduling:

- Given (α, T),
- Deliver **half** of **every** (α, T)-good subset
- In time relative to αT

Harder problem than (regular) scheduling

- Recall: nothing better than $\mathcal{O}(C \cdot D)$ known for local det. scheduling

Local Deterministic Noise-Tolerant Scheduling

Noise-Tolerant Scheduling:

- Given (α, T),
- Deliver **half** of **every** (α, T)-good subset
- In time relative to αT

Harder problem than (regular) scheduling

- Recall: nothing better than $\mathcal{O}(C \cdot D)$ known for local det. scheduling

We do not break this

Local Deterministic Noise-Tolerant Scheduling

Noise-Tolerant Scheduling:

- Given (α, T),
- Deliver **half** of **every** (α, T)-good subset
- In time relative to αT

Harder problem than (regular) scheduling

- Recall: nothing better than $\mathcal{O}(C \cdot D)$ known for local det. scheduling

We do not break this

- Restriction: poly-size *domain path set* $\mathcal{P}$

Local Deterministic Noise-Tolerant Scheduling

Noise-Tolerant Scheduling:

- Given (α, T),
- Deliver **half** of **every** (α, T)-good subset
- In time relative to αT

Harder problem than (regular) scheduling

- Recall: nothing better than $\mathcal{O}(C \cdot D)$ known for local det. scheduling

We do not break this

- Restriction: poly-size *domain path set* $\mathcal{P}$
- $\mathcal{P}$: given set guaranteed to contain all packet paths

Local Deterministic Noise-Tolerant Scheduling

Noise-Tolerant Scheduling:

- Given (α, T),
- Deliver **half** of **every** (α, T)-good subset
- In time relative to αT

Harder problem than (regular) scheduling

- Recall: nothing better than $\mathcal{O}(C \cdot D)$ known for local det. scheduling

We do not break this

- Restriction: poly-size *domain path set* $\mathcal{P}$
- $\mathcal{P}$: given set guaranteed to contain all packet paths
- Here: domain path set = semi-oblivious routing (αn^2 paths!)

Local Deterministic Noise-Tolerant Scheduling

Noise-Tolerant Scheduling:

- Given (α, T),
- Deliver **half** of **every** (α, T)-good subset
- In time relative to αT

Domain path set: given set guaranteed to contain all packet paths

Local Deterministic Noise-Tolerant Scheduling

Noise-Tolerant Scheduling:

- Given (α, T),
- Deliver **half** of **every** (α, T)-good subset
- In time relative to αT

Domain path set: given set guaranteed to contain all packet paths

Competitive Local Det. Noise-Tolerant Scheduling

For every graph and poly-size domain path set $\mathcal{P}$, there exists a *local* and *deterministic* Noise-Tolerant Scheduling algorithm that uses $\alpha T \cdot \mathrm{poly}(\log n)$ time steps.

The "+"

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling

The "+"

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling

The Algorithm:

The "+"

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling

The Algorithm:

1. $\mathcal{P} \leftarrow \alpha$-sparse β-competitive semi-oblivious routing

The "+"

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling

The Algorithm:

1. $\mathcal{P} \leftarrow \alpha$-sparse β-competitive semi-oblivious routing
2. Repeat $\mathcal{O}(\log n)$ times:

The "$+$"

Packet Routing $=$ Sparse Semi-Oblivious Path Selection
$+$ Local Det. Noise-Tolerant Scheduling

The Algorithm:

1. $\mathcal{P} \leftarrow \alpha$-sparse β-competitive semi-oblivious routing
2. Repeat $\mathcal{O}(\log n)$ times:
   a. For $i = 1, 2, \ldots, \alpha$:

The "+"

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling

The Algorithm:

1. $\mathcal{P} \leftarrow \alpha$-sparse β-competitive semi-oblivious routing
2. Repeat $\mathcal{O}(\log n)$ times:
   a. For $i = 1, 2, \ldots, \alpha$:
      i. For each (s, t), set (s, t)-packets' paths to $P(s, t)_i$

The "+"

Packet Routing = Sparse Semi-Oblivious Path Selection
+ Local Det. Noise-Tolerant Scheduling

The Algorithm:

1. $\mathcal{P} \leftarrow \alpha$-sparse β-competitive semi-oblivious routing
2. Repeat $\mathcal{O}(\log n)$ times:
    a. For $i = 1, 2, \ldots, \alpha$:
        i. For each (s, t), set (s, t)-packets' paths to $P(s, t)_i$
        ii. Run $(2\alpha, T)$-local det. noise-tolerant scheduling *with return*

End of Talk

Questions?

References

[GHZ21] Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. "Hop-constrained oblivious routing". In: *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM, 2021, pp. 1208–1220. DOI: 10.1145/3406325.3451098. URL: https://doi.org/10.1145/3406325.3451098.

[KKT90] Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas. "Tight Bounds for Oblivious Routing in the Hypercube". In: *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '90, Island of Crete, Greece, July 2-6, 1990*. Ed. by Frank Thomson Leighton. ACM, 1990, pp. 31–36. DOI: 10.1145/97444.97453. URL: https://doi.org/10.1145/97444.97453.

[LMR94] Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. "Packet Routing and Job-Shop Scheduling in O(Congestion + Dilation) Steps". In: *Comb.* 14.2 (1994), pp. 167–186. DOI: 10.1007/BF01215349. URL: https://doi.org/10.1007/BF01215349.

[Räc02] Harald Räcke. "Minimizing Congestion in General Networks". In: *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*. IEEE Computer Society, 2002, pp. 43–52. DOI: 10.1109/SFCS.2002.1181881. URL: https://doi.org/10.1109/SFCS.2002.1181881.

[ZHR23] Goran Zuzic, Bernhard Haeupler, and Antti Roeyskoe. "Sparse Semi-Oblivious Routing: Few Random Paths Suffice". In: *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*. Ed. by Rotem Oshman et al. ACM, 2023, pp. 222–232. DOI: 10.1145/3583668.3594585. URL: https://doi.org/10.1145/3583668.3594585.