Switch Scheduling in a Hybrid World

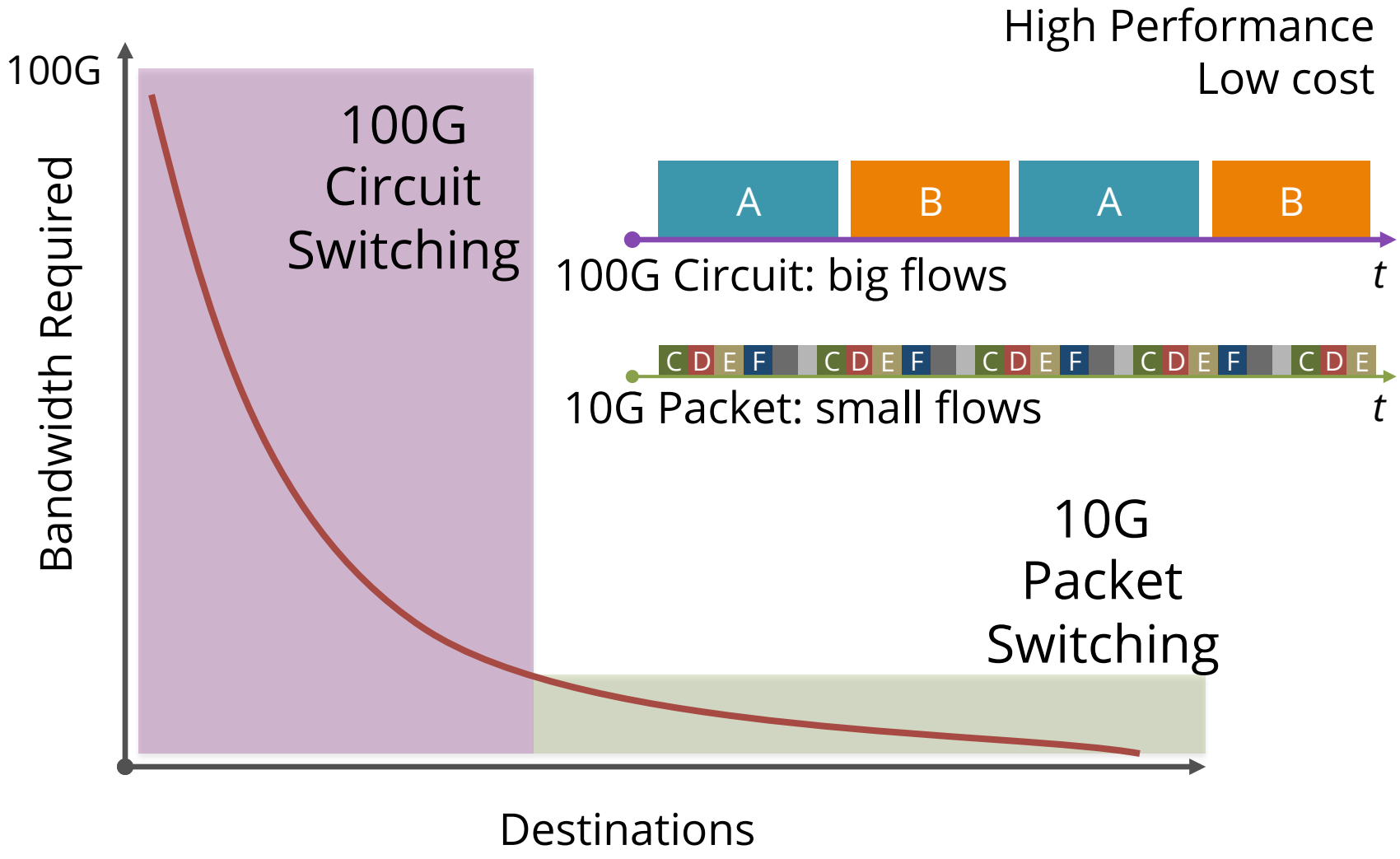
He “Lonnie” Liu, George Porter, George Papen
Stefan Savage, Geoffrey M. Voelker, Alex C. Snoeren

UCSDCSE
Computer Science and Engineering

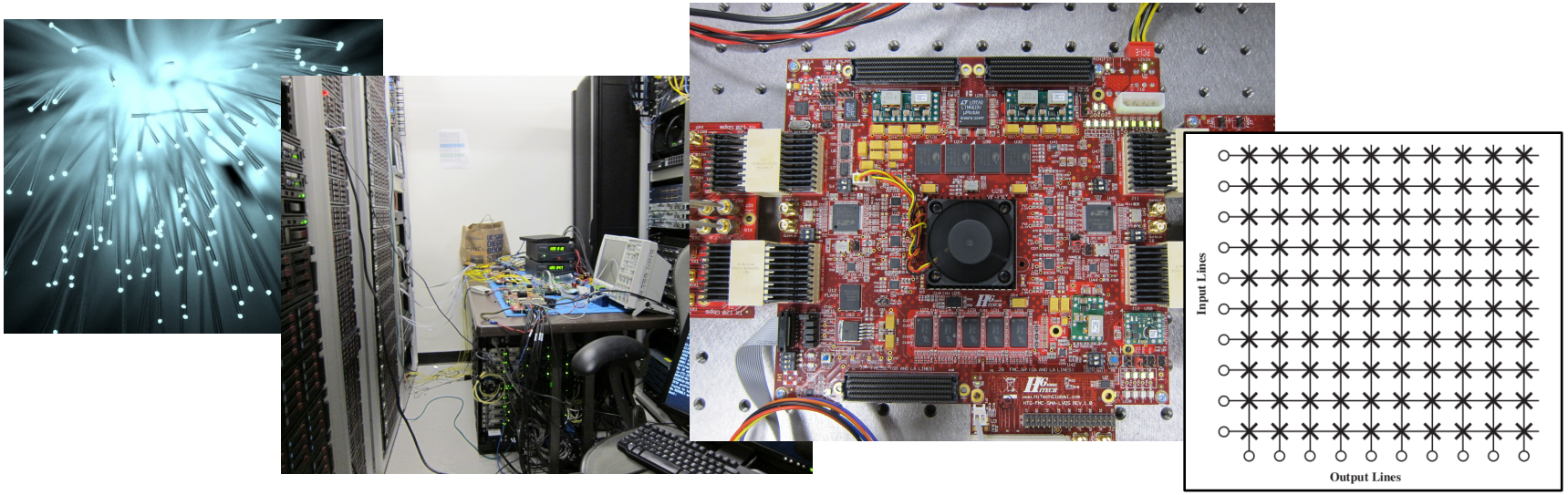
Design Switches for 100G Datacenters

[NSDI 2014]

Our Hybrid Approach: REACToR



Core of Big Flows: Optical Circuit Switch

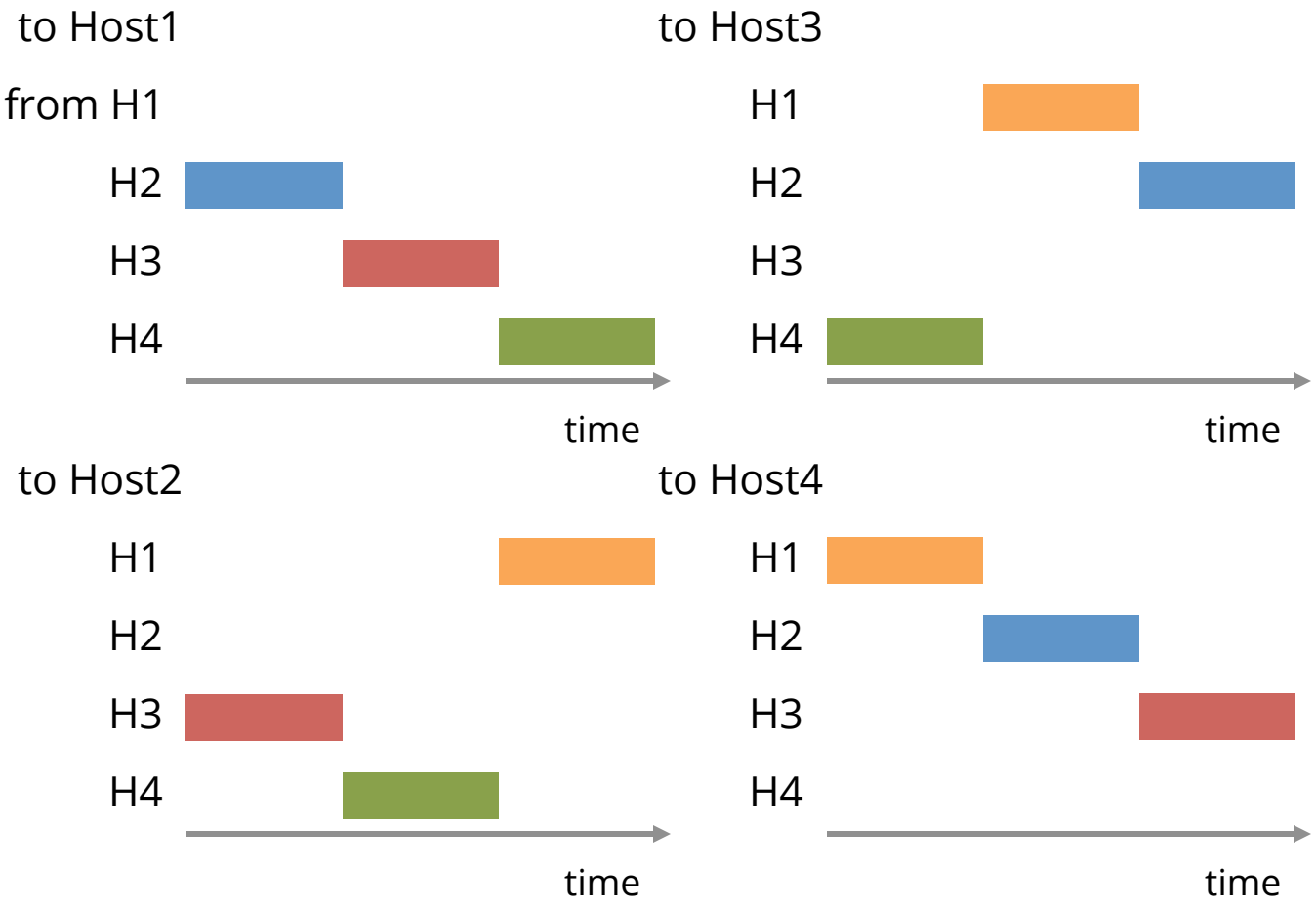


- 100G crossbar of mirror arrays (MEMS)
- Buffer-less, end-to-end optical, saves transceiver cost
- Fast: ~10μs reconfiguration, full speed TCP with TDMA
- Real-time control plane
- Needs a good crossbar scheduler

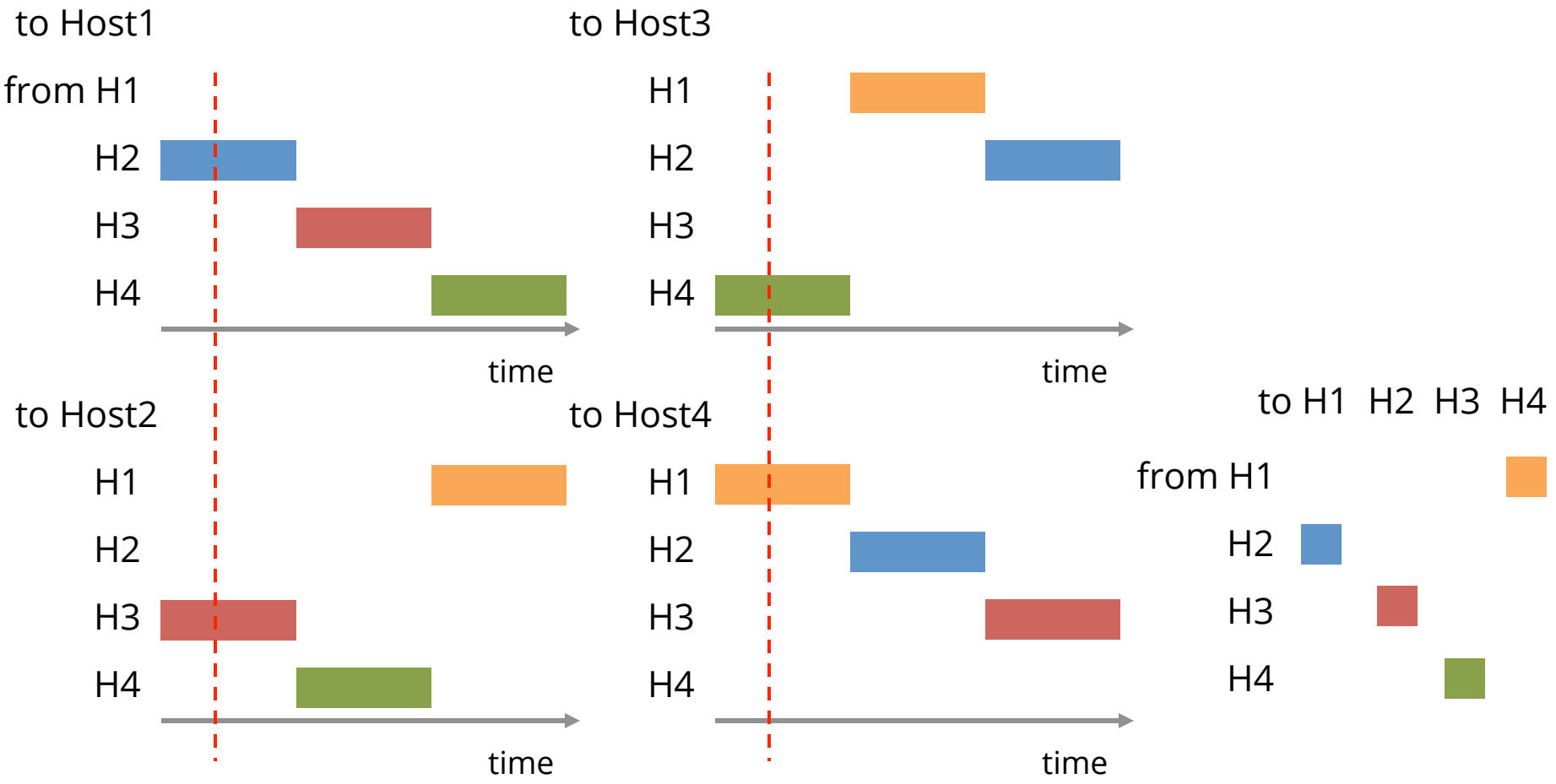
What is a Crossbar Schedule?

- A plan of crossbar configurations and their time durations
  - Example: round-robin for symmetric all-to-all demand

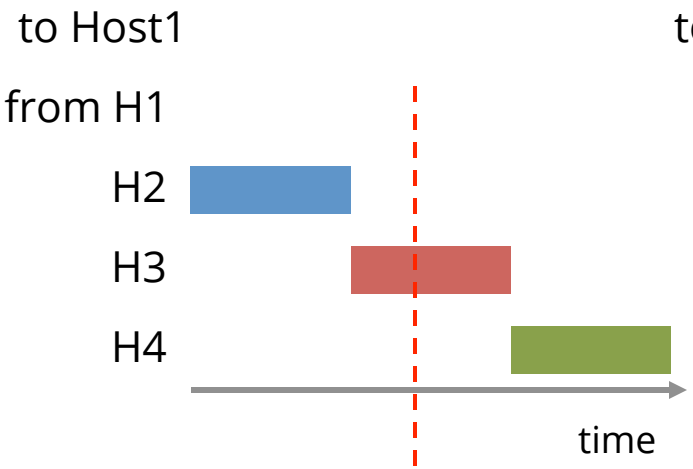
Round-robin Schedule: Visualized

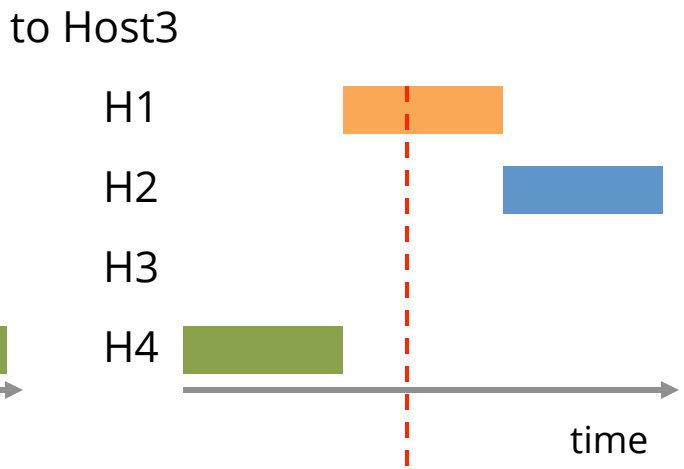


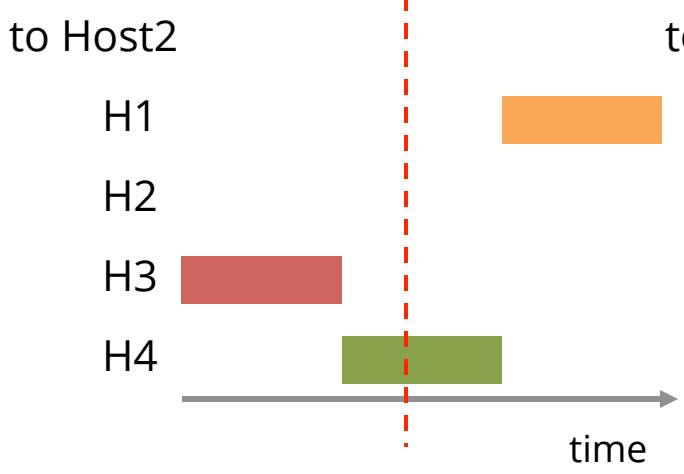
Round-robin Schedule: Visualized

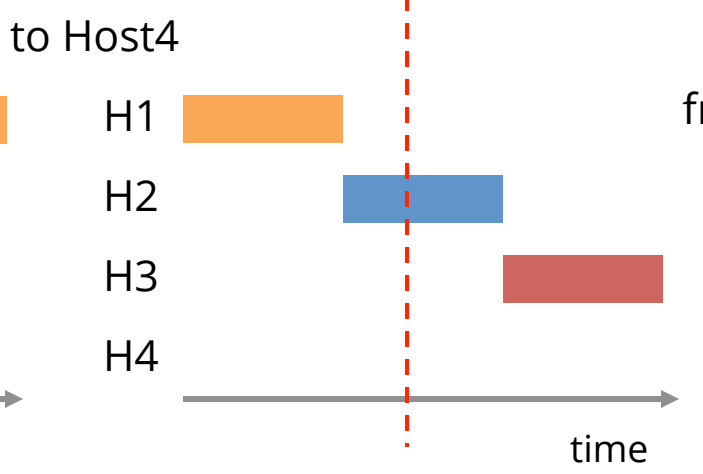


Round-robin Schedule: Visualized

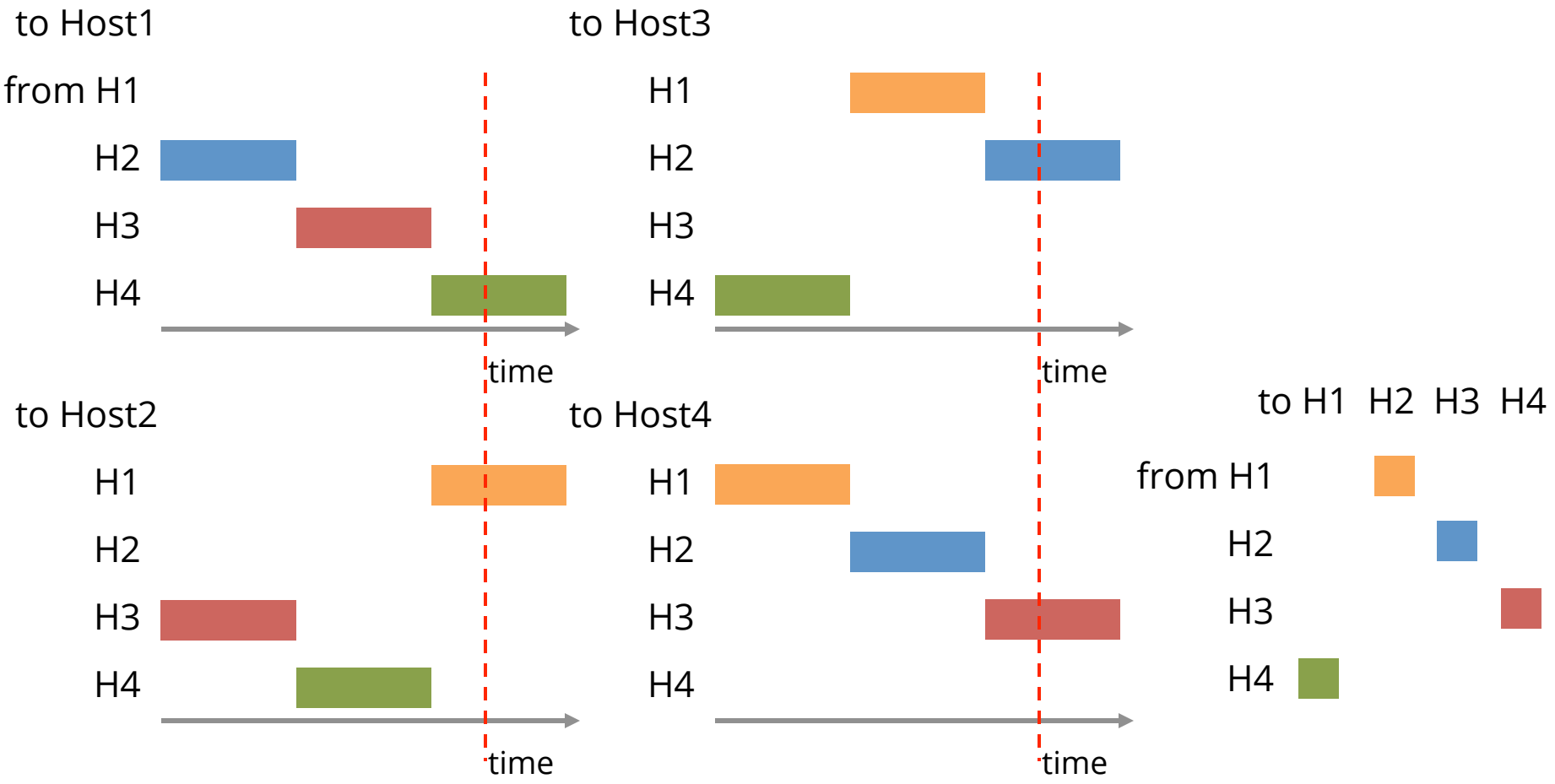




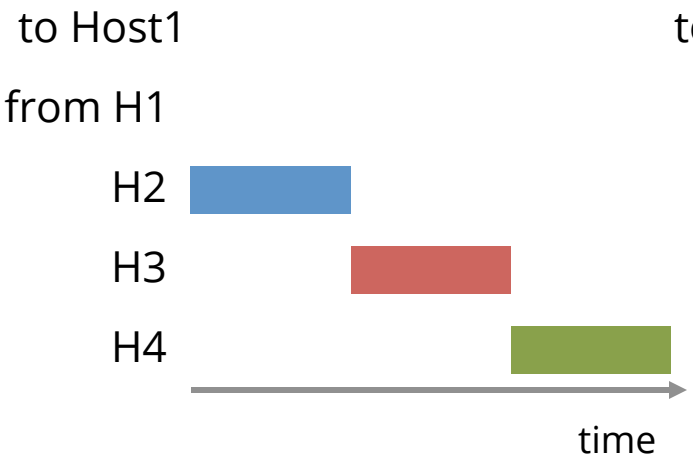


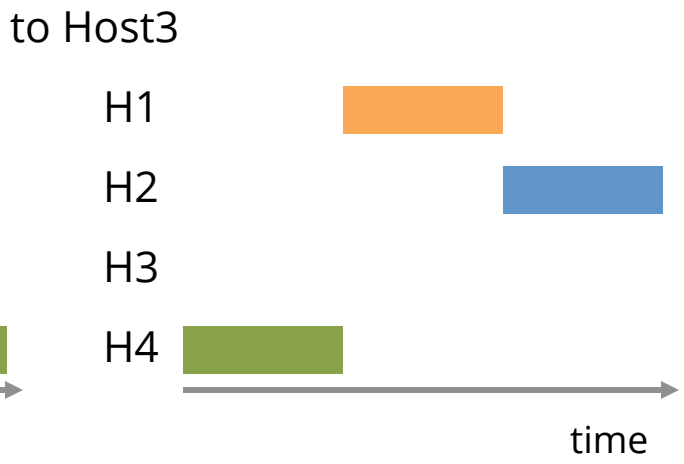


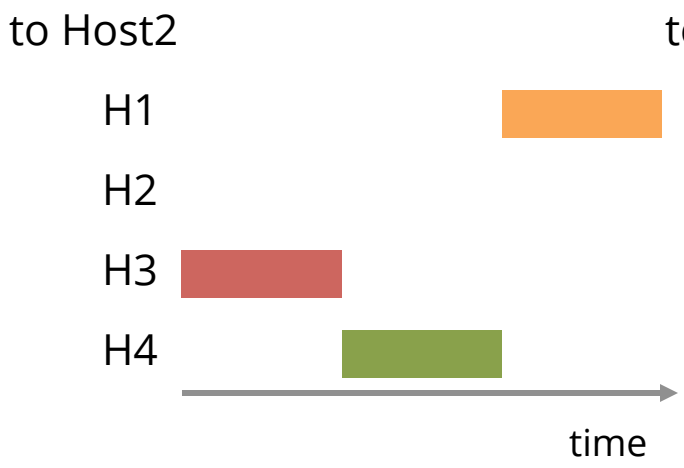
Round-robin Schedule: Visualized

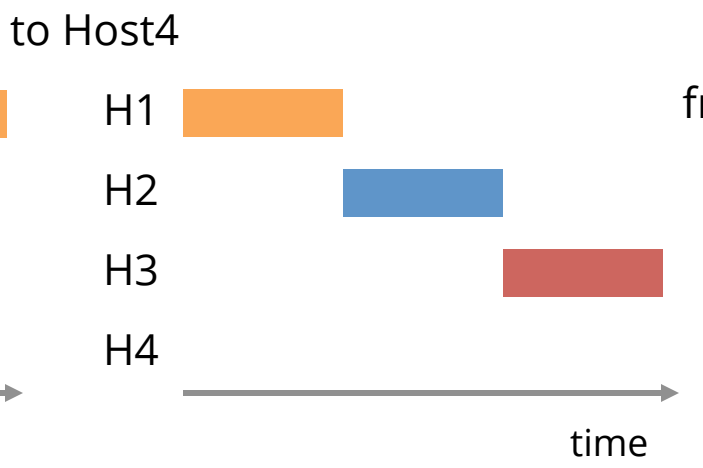


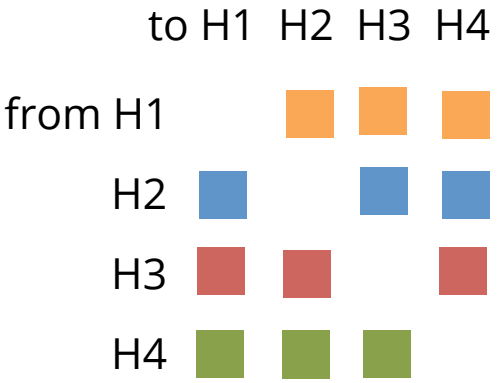
Round-robin Schedule: Visualized









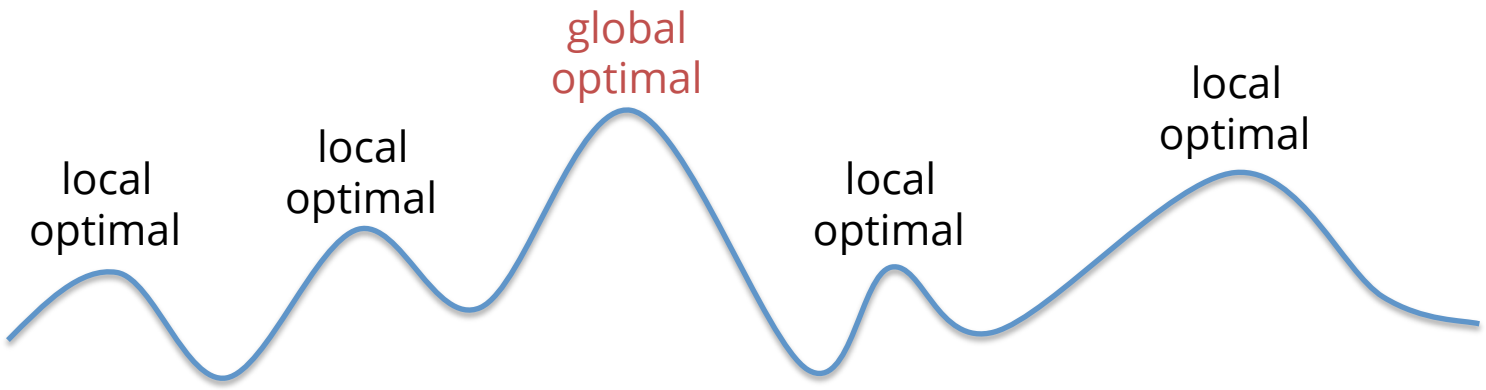


What is a Crossbar Schedule?

- A plan of crossbar configurations and their time durations
  - Example: round-robin for symmetric all-to-all demand
- Goal: deliver bi-sectional bandwidth on demand
- More complex when demand is irregular (i.e. skewed)
- Greedy does not lead to best solution

Crossbar Scheduling for Packet Switches

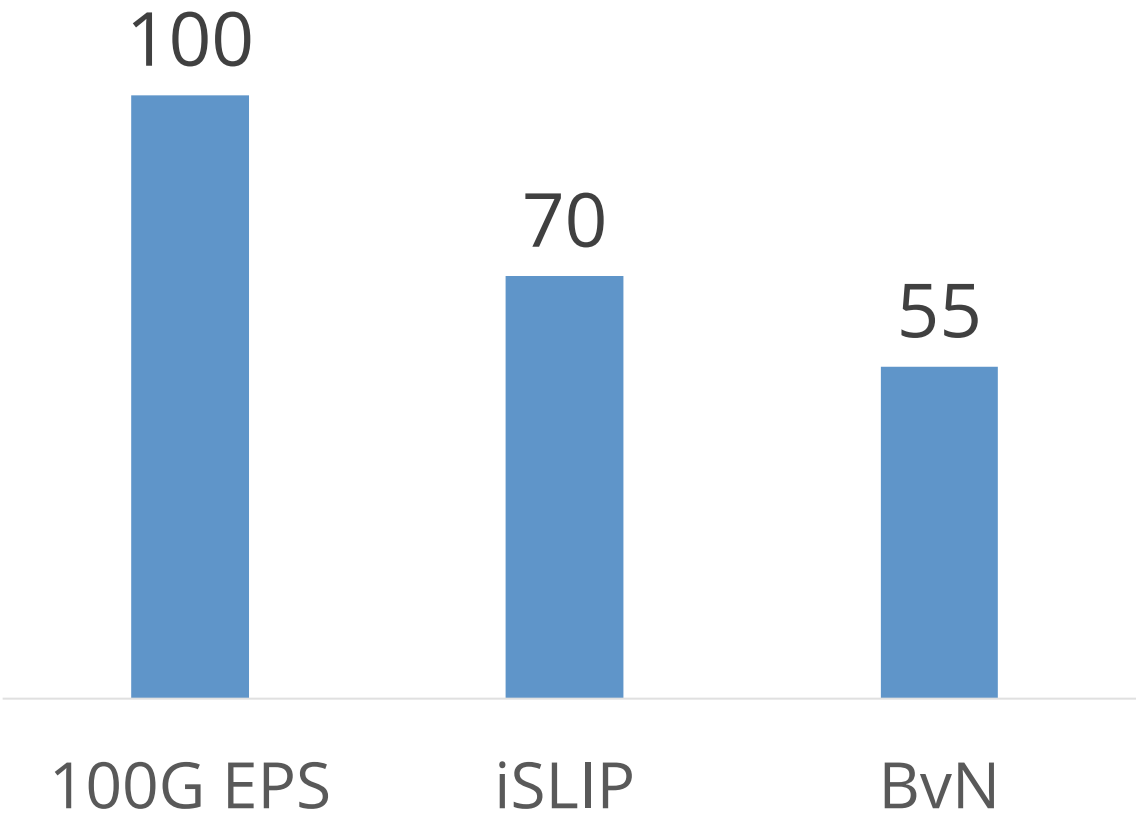
- Solved problem!
  - Packet switches also have crossbars
  - Algorithms are well-studied for decades
- Approach 1: local optimal scheduling, e.g. iSLIP
  - Easy to implement on hardware; fast
- Approach 2: global optimal scheduling, e.g. Birkhoff-von Neumann Decomposition (BvN)
  - 100% bi-sectional bandwidth guarantee; polynomial time



Applying to Optical Circuit Switching

Demand: 15 random flows per host (64 hosts simulated)

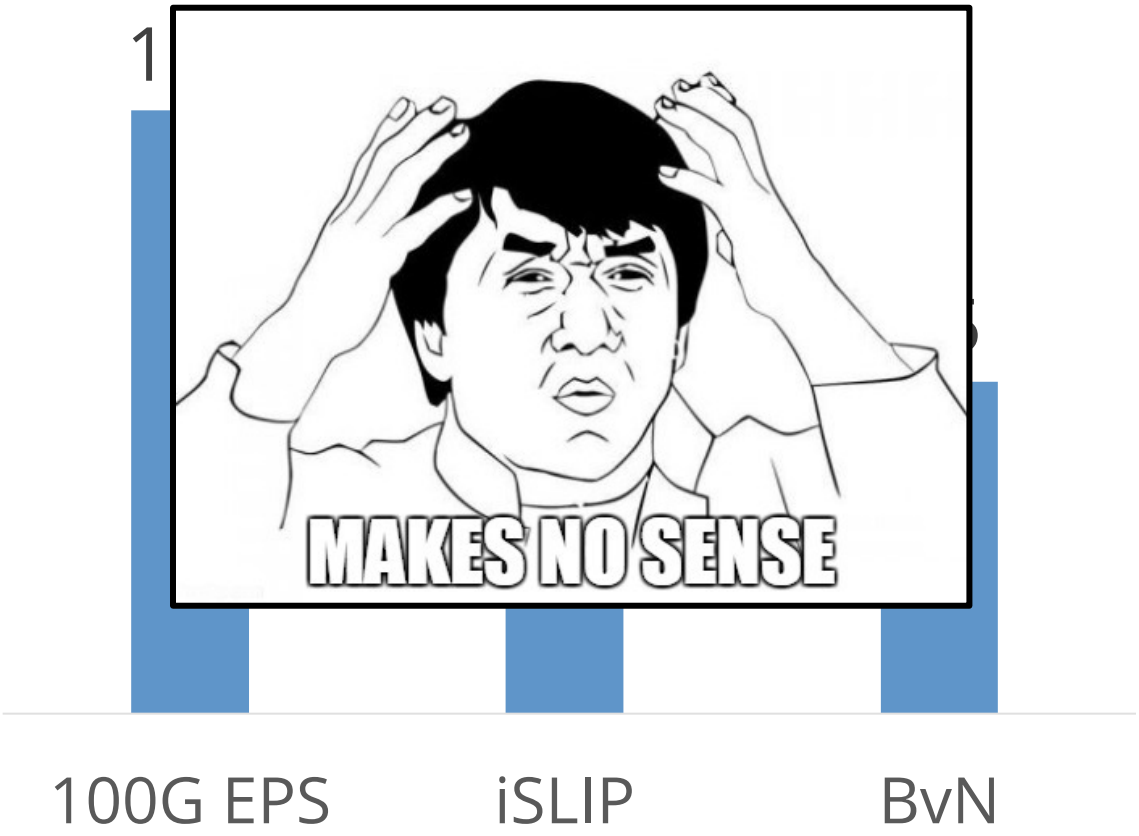
Percentage Served:



Applying to Optical Circuit Switching

Demand: 15 random flows per host (64 hosts simulated)

Percentage Served:



Reason: The Condition Changed

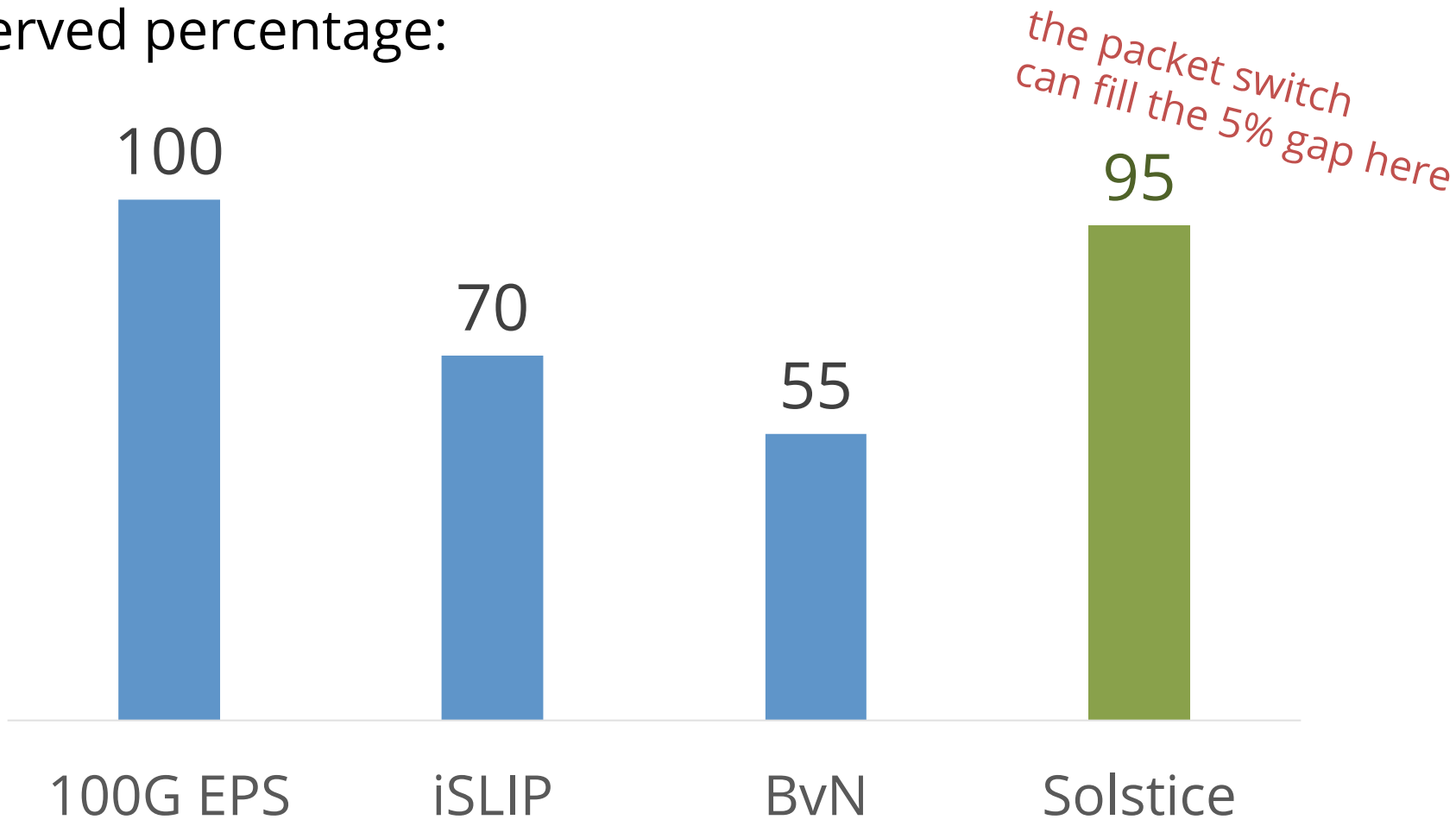
Electrical Packet Switching	**Optical Circuit Switching**
Buffers all the way	No buffers
Instant reconfiguration	~10μs reconfiguration

For this, we designed Solstice,
a new scheduling algorithm.

Solstice Delivers the Bandwidth

Demand: 15 random flows per host (64 hosts simulated)

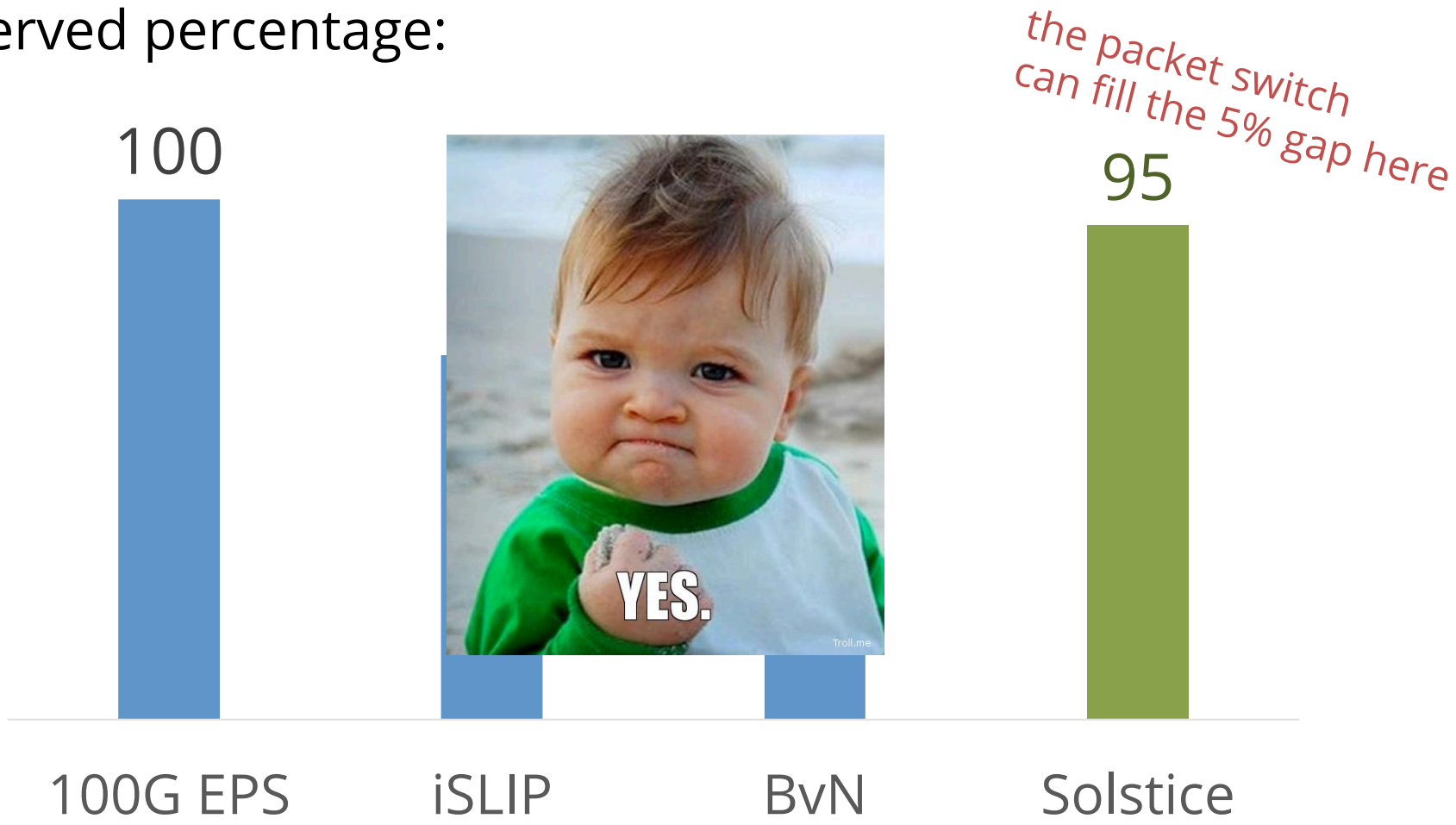
Served percentage:



Solstice Delivers the Bandwidth

Demand: 15 random flows per host (64 hosts simulated)

Served percentage:



Outline

- Define our circuit switch scheduling problem
- Explain why previous algorithms do not fit
- Introduce how Solstice works
- Evaluation: efficiency and performance

The Problem in Math

- Input: an $N \times N$ demand matrix D to send in W
- Output: a linear combination of permutation matrices
  - $\{ T_i P_i \}$
  - P_i: Permutation matrix (i.e. perfect matching)
- Optimization goal: maximize throughput
  - Throughput = Demand served / Total serving time
  - Total serving time = $\Sigma (T_i + \delta) = W$
  - Reconfiguration time: δ
  - Demand served = $\max(D, \Sigma (T_i P_i))$

Example: A Round-robin Schedule

```
    1   2   3   4
1:  .   30  13  7
2:  3   .   .   27
3:  18  12  .   14
4:  13  .   3   .
```

- Demand for next 50 time unit
- Rows, sources; columns, dests
- Sum of each row/col less than 50
- Satisfiable: EPS can serve 100%
- Reconfiguration: δ=1

A round-robin schedule: 16 + 16 + 15 + 3δ = 50

```
.  16 .  .        .  .  16 .        .  .  .  15
.  .  16 .        .  .  .  16       15 .  .  .
.  .  .  16       16 .  .  .        .  15 .  .
16 .  .  .        .  16 .  .        .  .  15 .
```

Example: A Round-robin Schedule

```
    1  2  3  4
1:  .  30 13 7
2:  3  .  .  27
3:  18 12 .  14
4:  13 .  3  .
```

- Demand for next 50 time unit
- Rows, sources; columns, dests
- Sum of each row/col less than 50
- Satisfiable: EPS can serve 100%
- Reconfiguration: δ=1

A round-robin schedule: 16 + 16 + 15 + 3δ = 50

```
.  16 .  .      .  .  16 .      .  .  .  15
.  .  16 .      .  .  .  16     15 .  .  .
.  .  .  16     16 .  .  .      .  15 .  .
16 .  .  .      .  16 .  .      .  .  15 .
```

Total demand: 140 Served: 113

Why iSLIP only delivers 70%?

- Local optimal algorithm, imperfect by design
  - Small $\Sigma (T_i P_i)$
  - 50% minimum guaranteed, 70% is normal result
- Used with speed-up: transmit 10G, switch 20G
  - Requires output buffers
- Optical circuit switches
  - All optics, no buffers (the cost savings feature)
  - Run at 70%? 30% capacity wasted.

Why BvN only delivers 50%?

- Designed for no reconfiguration cost
  - Guaranteed 100% bi-sectional bandwidth
- Output up to $O(N^2)$ configurations
  - Even when the matrix is skewed and sparse
- Low efficiency if each reconfiguration costs 10μs
  - $\Sigma (T_i + \delta)$ has $N^2 \times 10\mu s$
  - Spending too much time just rotating the mirrors

The Key Ideas of Solstice

- Idea 1 - Target global optimal
  - since there is no speed up
- Idea 2 - Look for longer-lasting configurations first
  - since each reconfiguration costs 10μs
- Inherits the basic steps of BvN.

How BvN works?

- Step 1: Stuffing // necessary for optimal
  - Stuff the demand matrix into a doubly-stochastic matrix

- Step 2: Slicing // lead to possibly $O(N^2)$ slices
  - Pick permutation matrix masks of non-zero elements
  - Slice out the permutation until one element is 0
  - Repeat

How BvN works

Step 1: Stuff

So that each row and column has the same sum.

	1	2	3	4
1:	.	30	13	7
2:	3	.	.	27
3:	18	12	.	14
4:	13	.	3	.

	1	2	3	4
1:	.	30	13	7
2:	4	7	12	27
3:	18	12	6	14
4:	28	1	19	2

How BvN works

Step 2: Slice
Pick a permutation of non-zero elements
Get a slice with the smallest value

.	30	13	7
4	7	12	27
18	12	6	14
28	1	19	2

.	.	1	.
.	.	.	1
1	.	.	.
.	1	.	.

How BvN works

Step 2: Slice
Pick a permutation of non-zero elements
Get a slice with the smallest value

.	30	12	7
4	7	12	26
17	12	6	14
28	.	19	2

.	.	1	.
.	.	.	1
1	.	.	.
.	1	.	.

How BvN works

Slice 2

.	30	12	7
4	7	12	26
17	12	6	14
28	.	19	2

.	2	.	.
2	.	.	.
.	.	2	.
.	.	.	2

.	.	1	.
.	.	.	1
1	.	.	.
.	1	.	.

How BvN works

Slice 3

.	28	12	7
2	7	12	26
17	12	4	14
28	.	19	.

.	.	.	2
2	.	.	.
.	2	.	.
.	.	2	.

.	.	1	.
.	.	.	1
1	.	.	.
.	1	.	.

.	.	2	.
2	.	.	.
.	.	2	.
.	.	.	2

How BvN works

Slice 4

.	28	12	5
.	7	12	26
17	10	4	14
28	.	17	.

.	.	.	4
.	4	.	.
.	.	4	.
4	.	.	.

.	.	1	.
.	.	.	1
1	.	.	.
.	1	.	.

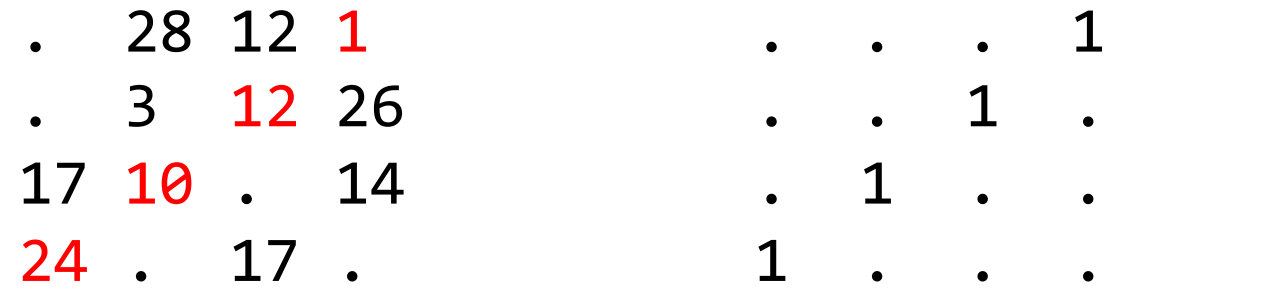
.	.	2	.
2	.	.	.
.	.	.	2
.	.	.	2

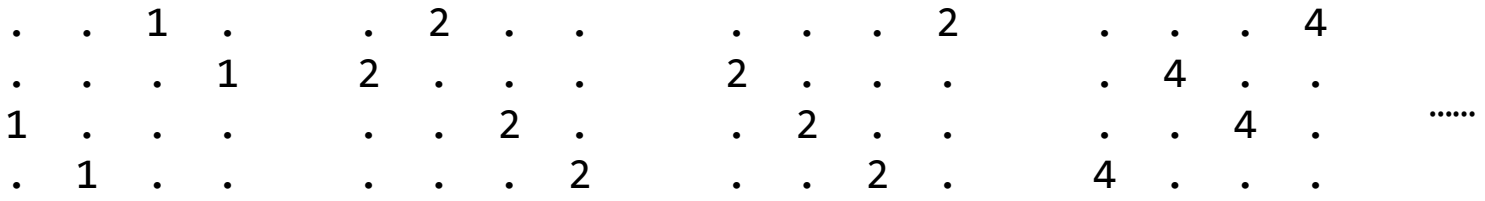
.	.	.	2
2	.	.	.
.	2	.	.
.	.	2	.

How BvN works

Slice 5

(up to $O(N^2)$ slices)





How Solstice works?

- Step 1: Sharp Stuffing
  - Stuff the demand matrix into a doubly-stochastic matrix
  - Prioritize non-zero elements
  - Prioritize idler queues

- Step 2: Threshold Slicing
  - Pick permutation matrix masks of elements larger than *Thresh*
  - Slice out the permutation until one element is 0
  - Repeat; exponentially reduce the *Thresh* if needed

How Solstice works

Step 1: Sharp Stuff

	1	2	3	4
1:	.	30	13	7
2:	3	.	.	27
3:	18	12	.	14
4:	13	.	3	.

	1	2	3	4
1:	.	30	13	7
2:	19	2	.	29
3:	18	18	.	14
4:	13	.	37	.

How Solstice Work

Step 2: Slice

Threshold = 16

```
.  30 13 7          .  18 .  .
19 2  .  29         .  .  .  18
18 18 .  14         18 .  .  .
13 .  37 .          .  .  18 .
```

How Solstice Work

Step 2: Slice

Threshold = 8

.	12	13	7
19	2	.	11
.	18	.	14
13	.	19	.

.	.	11	.
.	.	.	11
.	11	.	.
11	.	.	.

.	18	.	.
.	.	.	18
18	.	.	.
.	.	18	.

How Solstice Work

Step 2: Slice

Threshold = 8

```
.  12  2  7
19  2  .  .
.  7  .  14
2  .  19 .
```

```
.  12 .  .
12 .  .  .
.  .  .  12
.  .  12 .
```

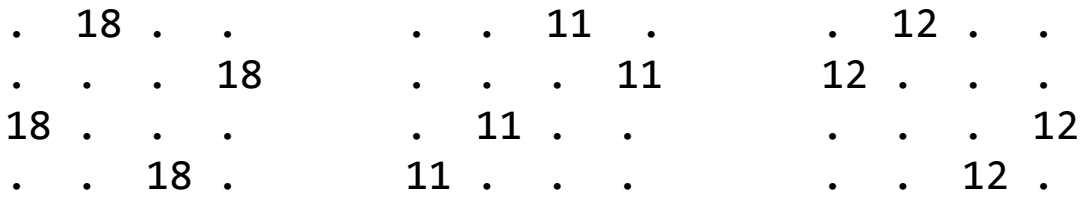
```
.  18 .  .
.  .  .  18
18 .  .  .
.  .  18 .
```

```
.  .  11 .
.  .  .  11
.  11 .  .
11 .  .  .
```

How Solstice Work

Step 2: Slice

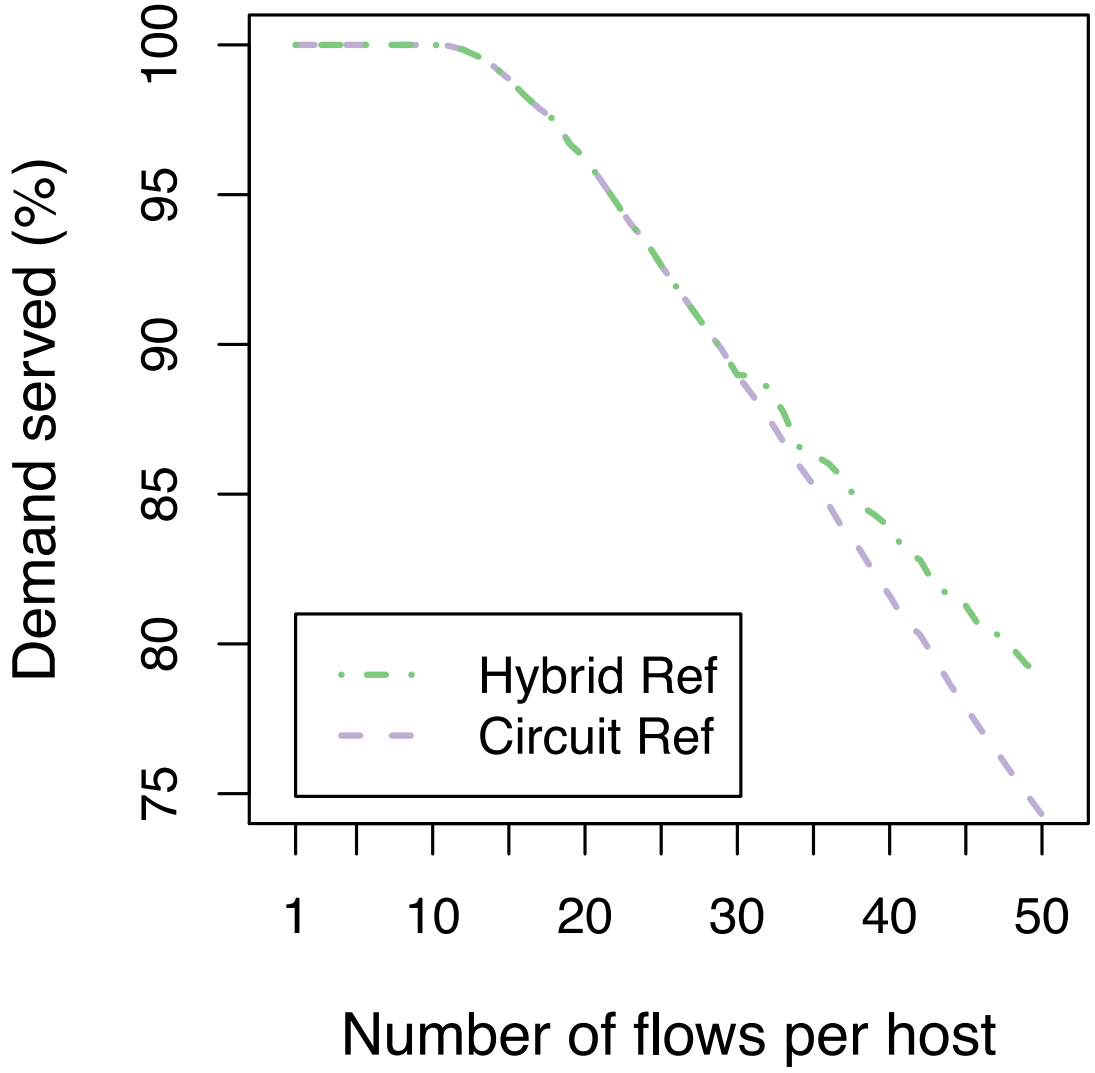
Threshold = 4
(much larger and fewer slices)



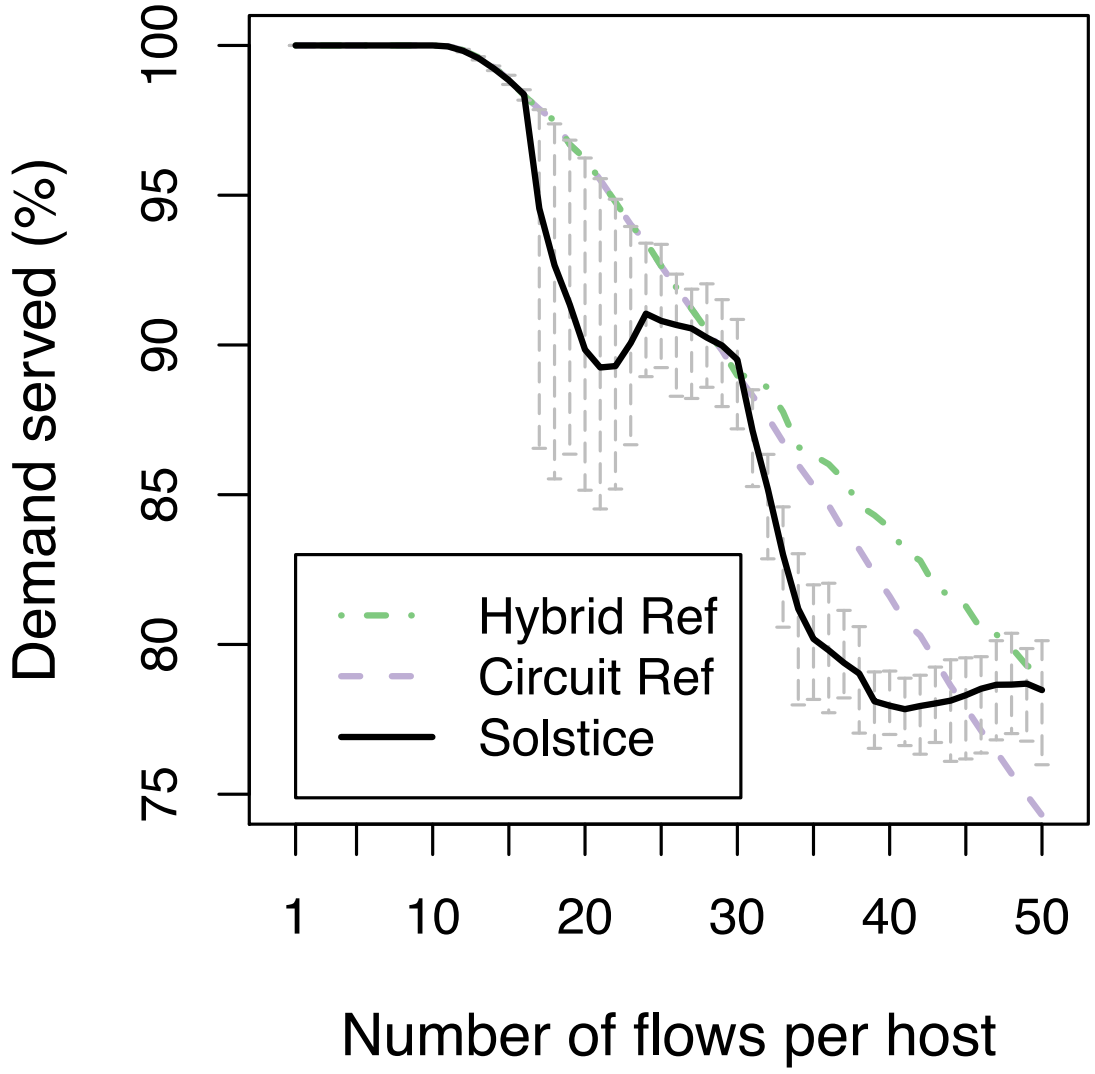
Evaluation

- Simulates hybrid REACToR switch of 64 hosts
  - 100G circuit switch + 10G packet switch
- Demand 1: a stack of random permutations
- Demand 2: random skewed demand
- Demand 3: random flows in fair-sharing state

Serving a Stack of Permutations

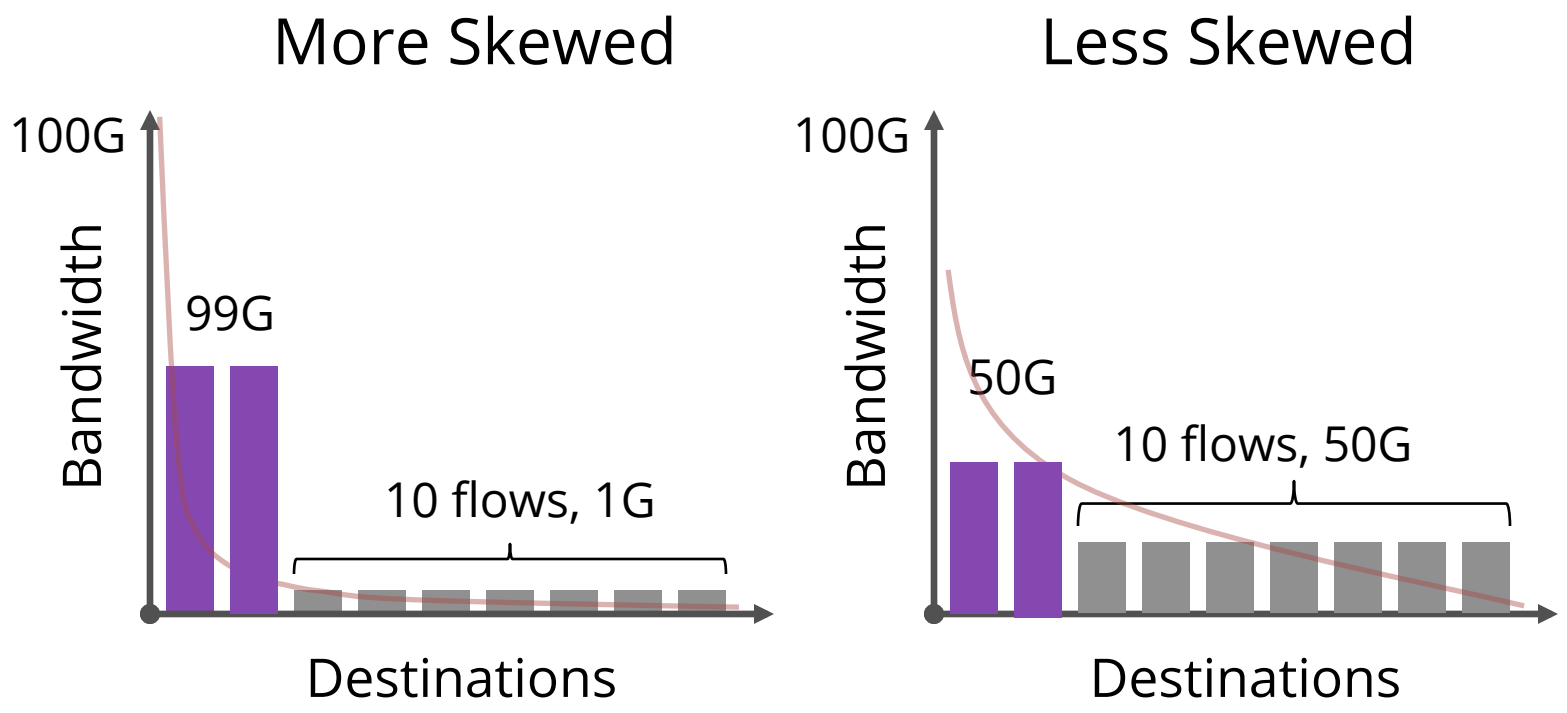


Serving a Stack of Permutations

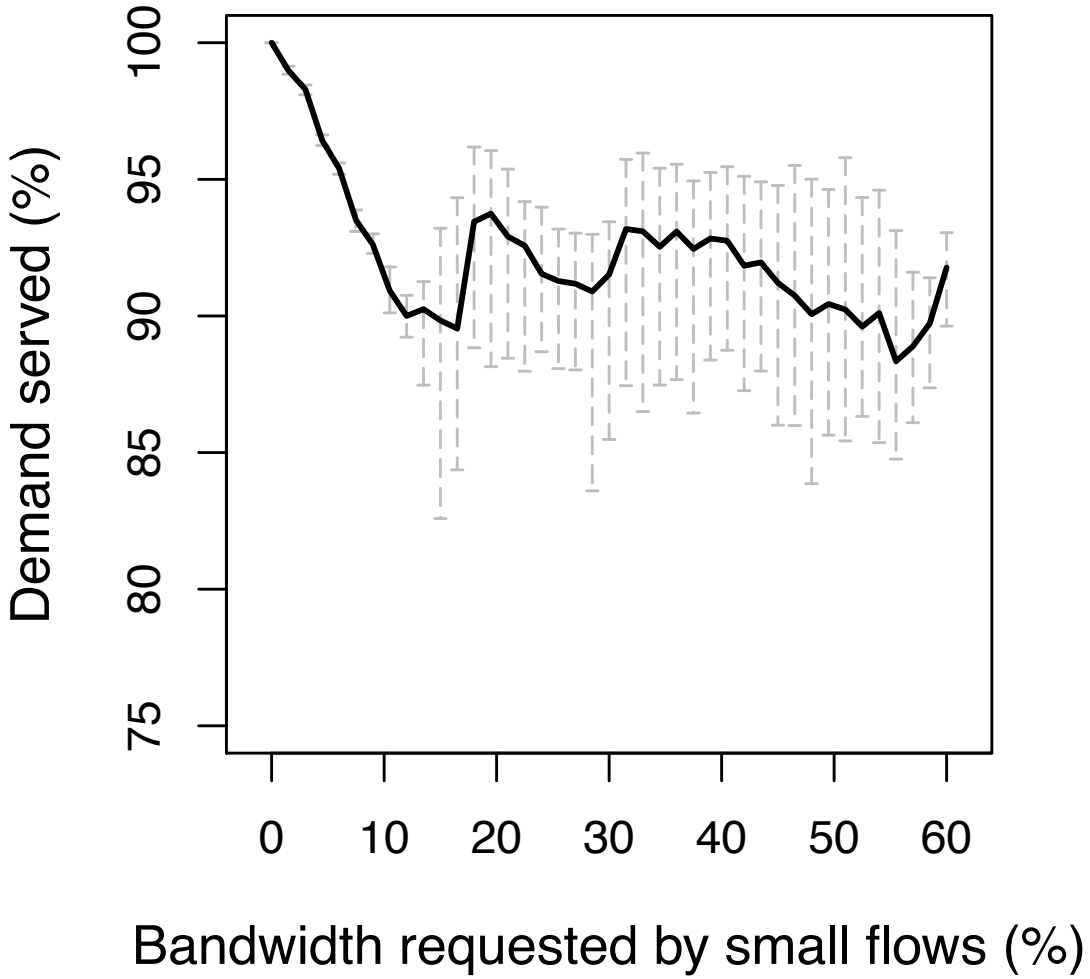


Serving Skewed Random Flows

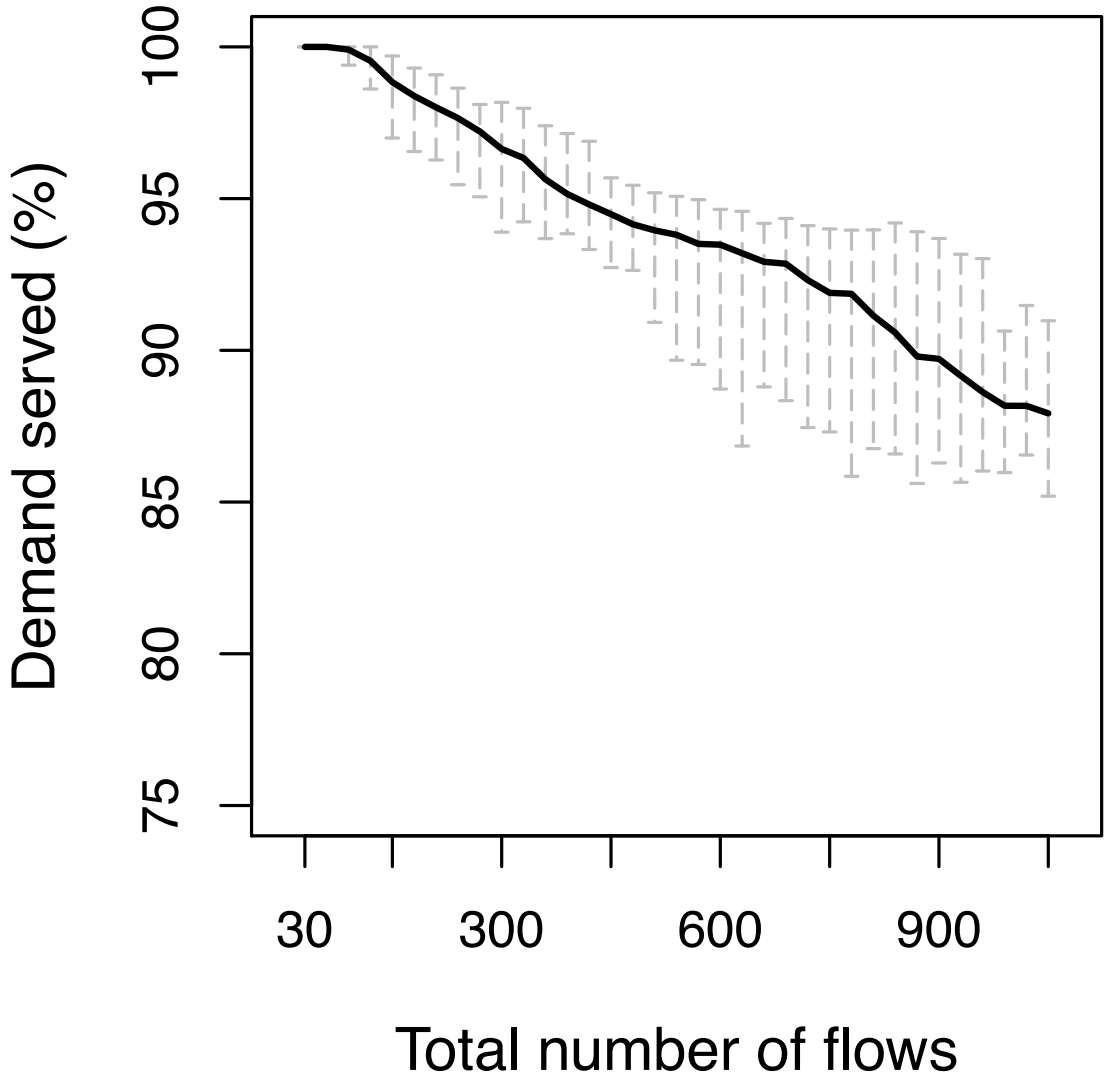
- Random permutations, 2 big flows, 10 small flows
- Different skewness:



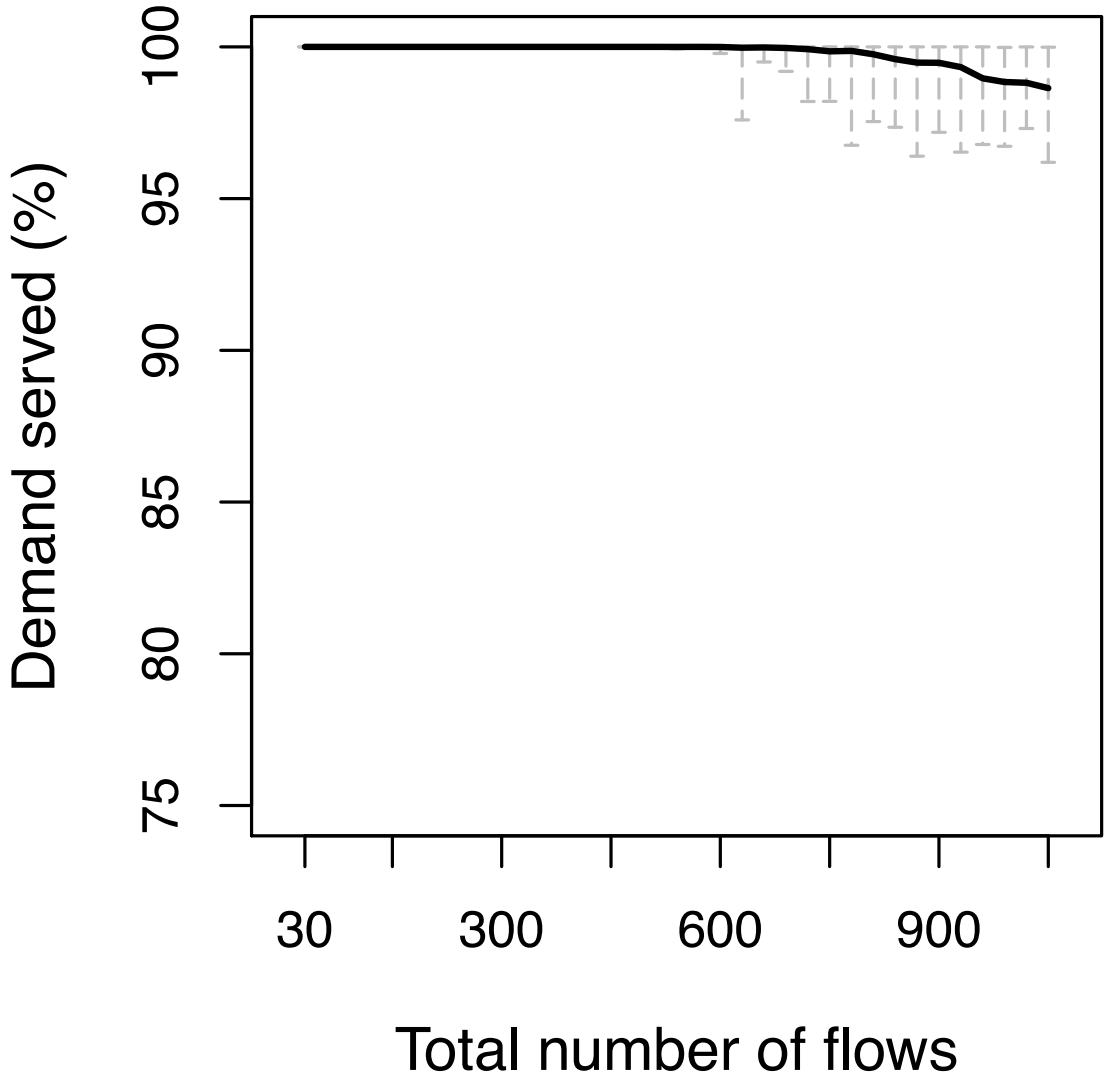
Serving Skewed Random Flows



Serving Random Flows



Serving Random Flows with Hybrid



Computation Speed

- Concern: Solstice is a (special) BvN, and BvN is slow
- Demand input: big flows, sparse matrix
  - 5 big flows per host, 5 non-zero elements per row/column
- O(N logN) computation time complexity observed
  - N = number of hosts
- On 8-host prototype test-bed:
  50μs to compute a schedule for the next 1.5ms
  (using single core)

Conclusion

- Defined the circuit scheduling problem for hybrid datacenters switches.

- Designed the Solstice algorithm.
  - Scheduled throughput akin to 100G electrical packet switch for skewed datacenter workloads

- Ongoing work:
  - Integration with dynamic demand estimation
  - Proof of time complexity
  - Hardware/parallel/distributed implementation
  - Finding a job