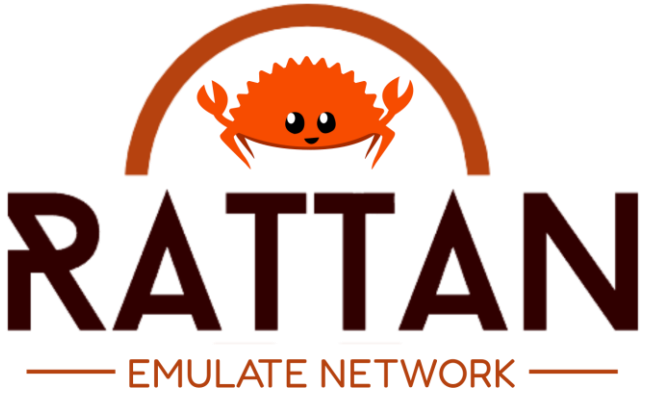


Rattan: An Extensible and Scalable Modular Internet Path Emulator

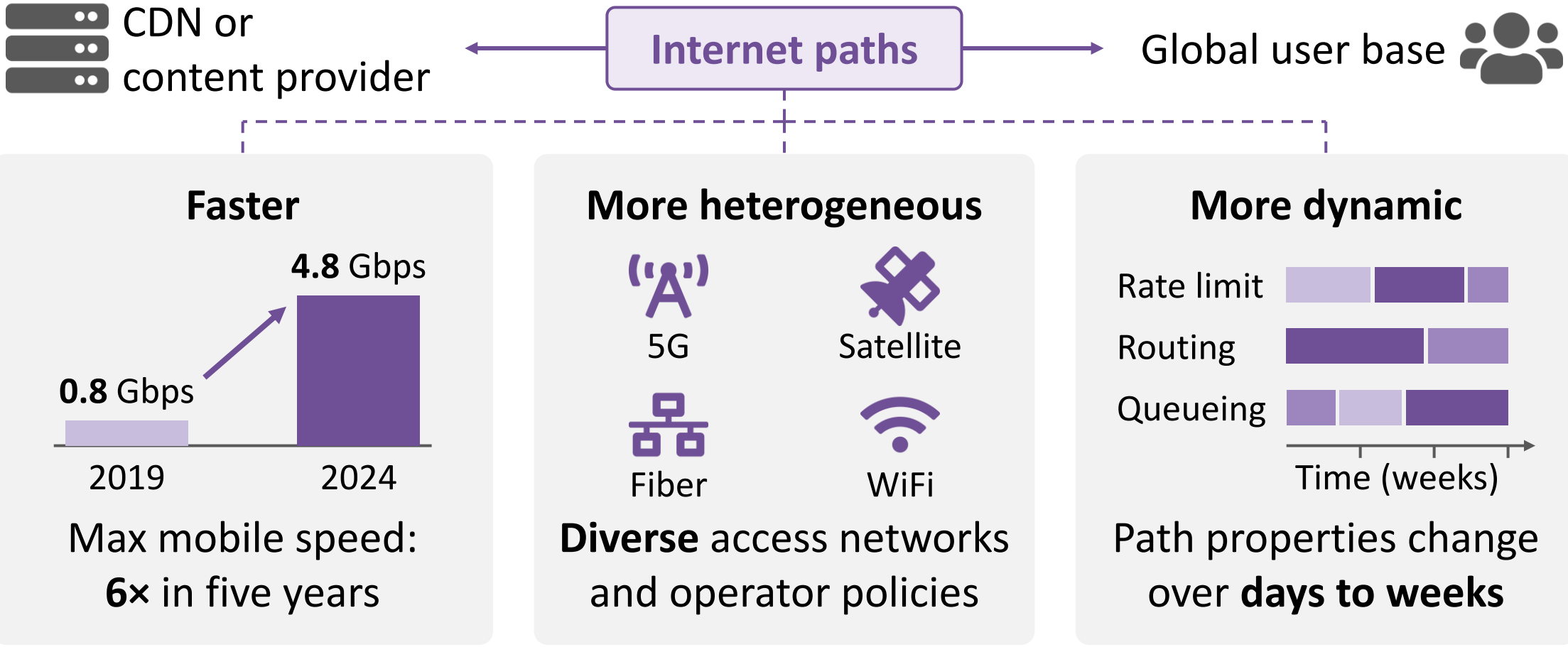
Yixin Shen*, Minhu Wang*, Bo Wang, **Haixuan Tong**, Chenghao Li, Yutong Xie, Yixuan Gao, Mingwei Xu, Jianping Wu

* Equal contribution

Tsinghua University

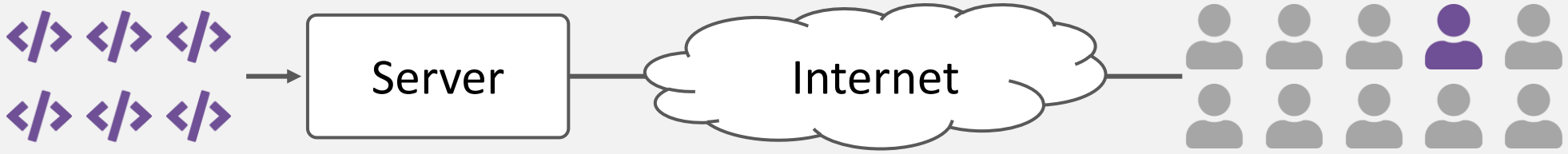
Internet Paths at Hyperscale

- A hyperscale provider serves a globally distributed user base



Validation Becomes the Bottleneck

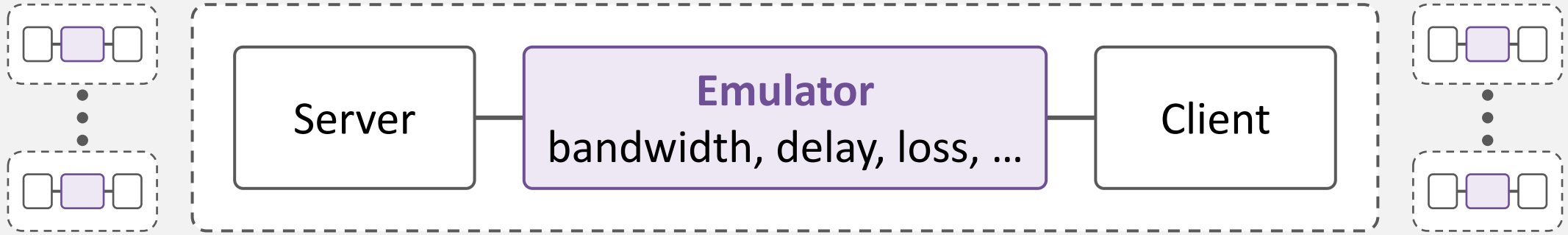
- Transport changes arrive faster and in larger numbers
  - E.g., new protocols, congestion control, multipath scheduling
  - Each must be validated: it may help some users but hurt others
- Online A/B testing is slow, and regressions are hard to diagnose
  - Only a few changes at a time, each on a small fraction of users
  - It shows that performance changed, but not where or why



24 weeks on average to roll out one congestion-control change

Emulation: The Necessary Complement

- Run the real, unmodified stack (apps, protocols) over an emulated Internet path (network conditions)

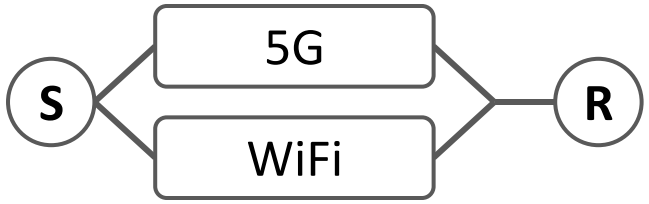


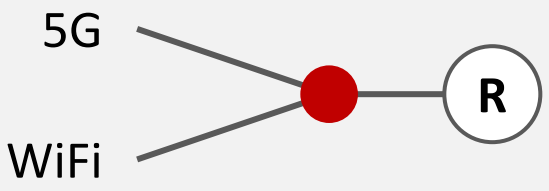
Controlled **Repeatable** **Parallel**

Emulation evaluates changes offline,
only advancing promising ones to A/B testing

A Real Case from Production

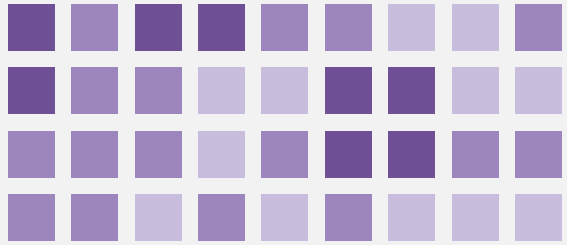
- A video service evaluates a multipath scheduler
  - Its users use both 5G and home WiFi simultaneously





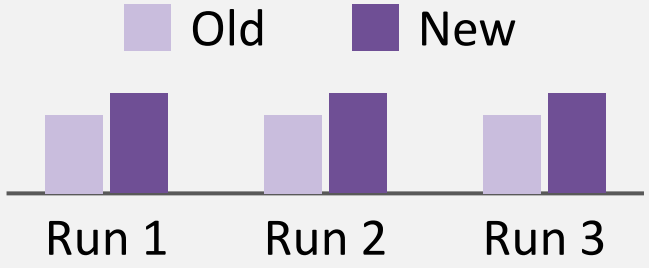
Shared bottleneck?

Capture how the two paths **interact**



Good on all paths?

Sweep **diverse** networks and workloads



Really better?

See **small differences** between schedulers

Fidelity to real paths

Comprehensive coverage of the path space

with **reproducible** and **stable** results

Two Demands, Three Requirements

Fidelity
to real paths

Comprehensive coverage
of the path space

with **reproducible** and
stable results

Expressiveness and extensibility

Diverse path conditions
New effects, easily added

Execution efficiency

High-throughput paths
Many experiments at once

Reproducible scale

Many machines
Stable results across runs

No Existing Emulator Meets All Three

- Each one trades one set of requirements for another

Category	Modeling unit	Limitation
Kernel-based Mininet, NetEm	**Qdisc** composed in the kernel	✗ Per-packet overhead ✗ No native trace replay
User-space Mahimahi	**Process** nested, one in another	✗ Context switches and copies ✗ Linear paths only
Hardware Keysight, HoloWAN	**Hardware block** fixed in an ASIC or FPGA	✗ Rigid to extend ✗ Difficult to scale

The Root Cause Is Coupling

Prior emulators
One unit: **coupled**

Qdisc Process Hardware

serves both simultaneously
Expression 🔗 **Execution**

Structural trade-off
Expressiveness vs. **efficiency**

Why this trade-off?

One unit does two jobs simultaneously

Expression
what users can express

Execution
how it is executed

Flexible **expression** → costly **execution**
Efficient **execution** → restricted **expression**

The trade-off comes from the coupling, not from emulation itself

Basic Idea

Prior emulators

One unit: **coupled**

Qdisc Process Hardware

serves both simultaneously

Expression 🔗 **Execution**

Structural trade-off

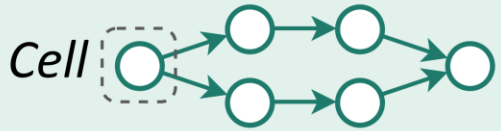
Expressiveness vs. **efficiency**

Rattan

Cell graph + runtime: **decoupled**

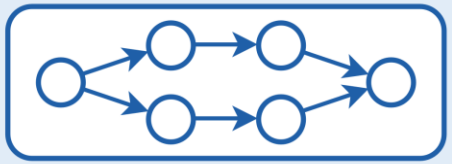
Expression

Cell graph (DAG)



Execution

Graph-aware runtime

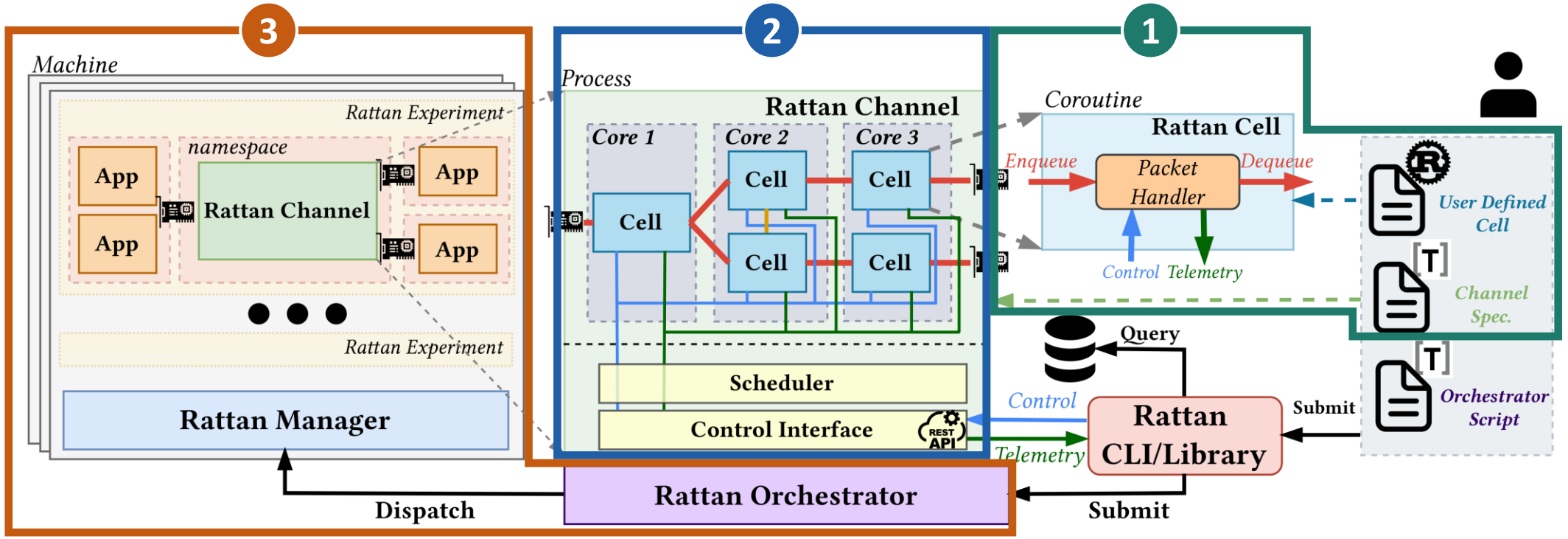


All three requirements

Expressiveness + **efficiency** + **reproducible scale**

Decouple expression from execution

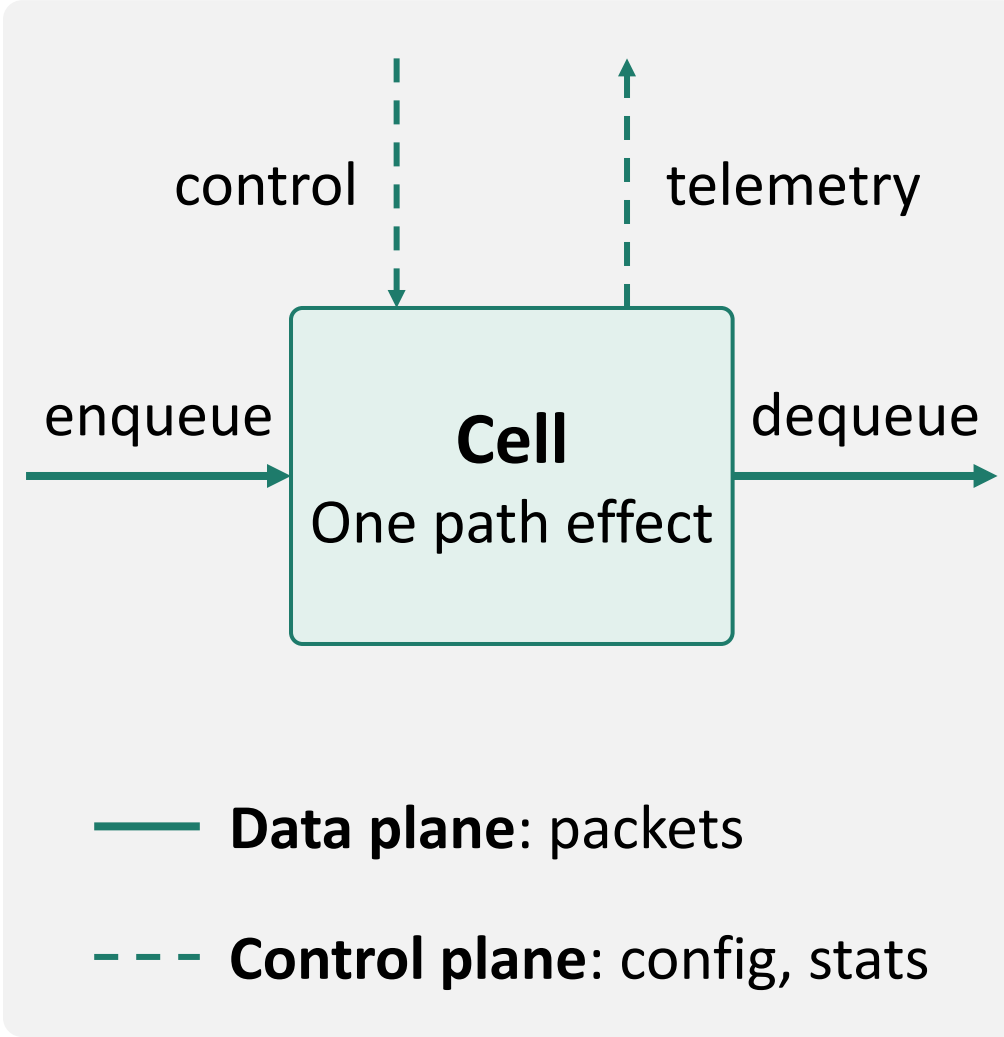
Rattan Overview



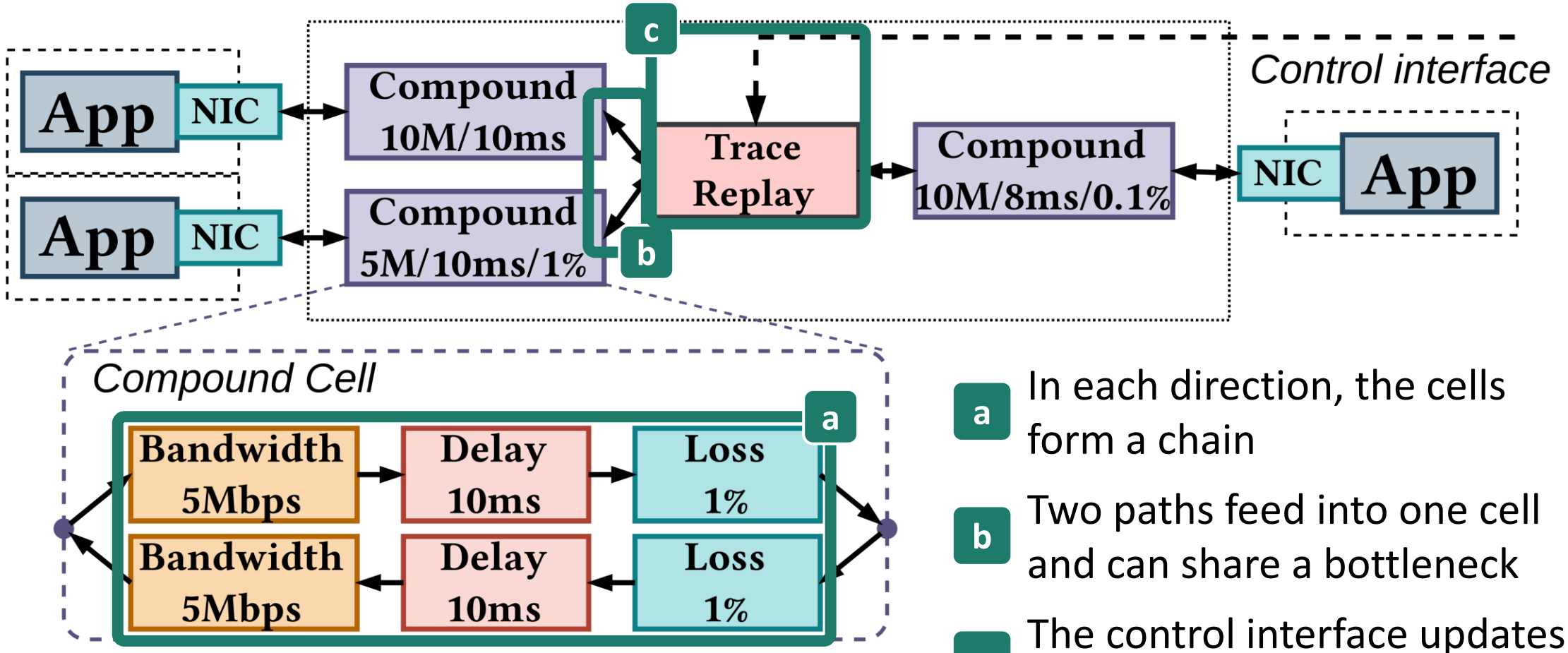
Cells express a path, the runtime executes the whole graph, and the orchestrator scales it with reproducible results

1 A Cell Encapsulates One Path Effect

- A **cell** encapsulates **one path effect**
  - Bandwidth, delay, loss, token bucket, trace replay, ...
- Each cell exposes **four interfaces**
  - The same for built-in and custom cells
- A new effect enters as **a new cell**
  - The runtime stays unchanged



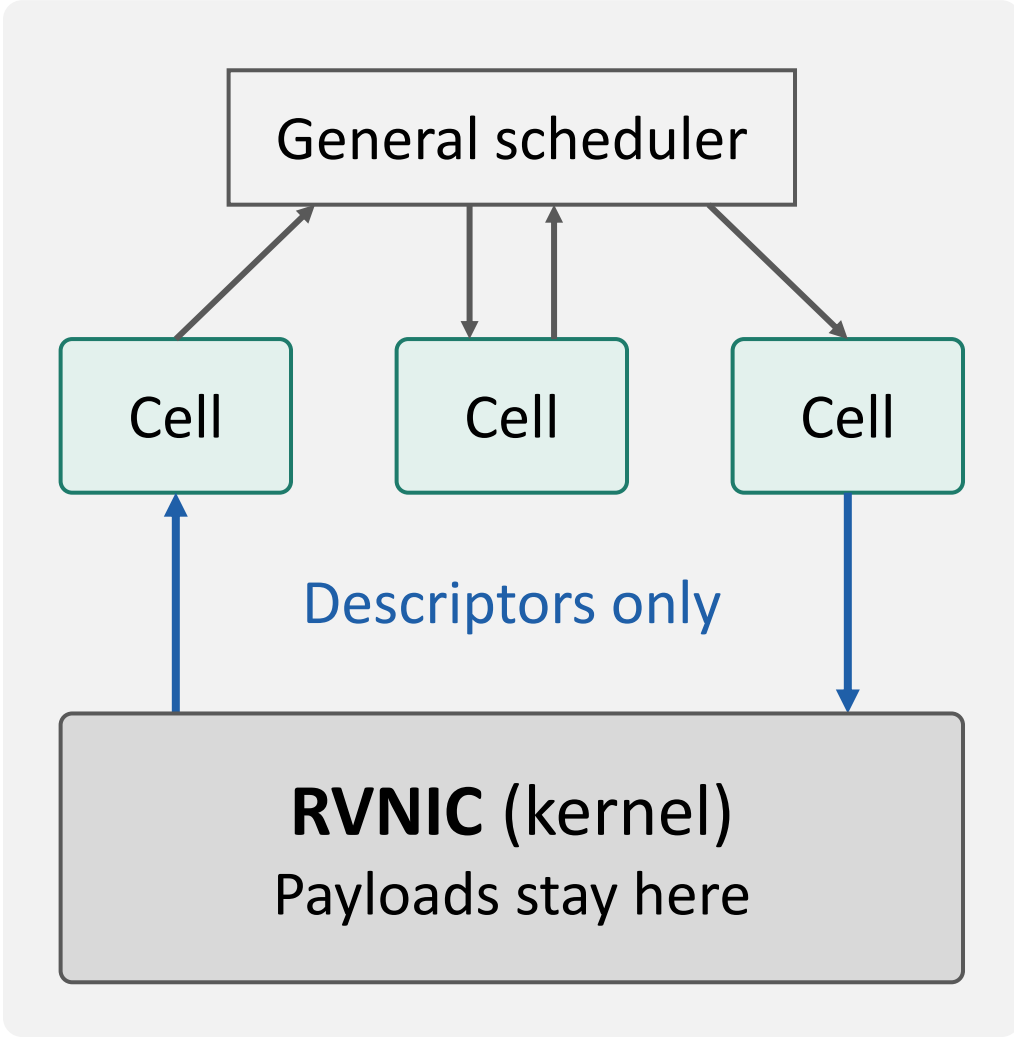
1 A Channel Is a Graph of Cells



- **a** In each direction, the cells form a chain
- **b** Two paths feed into one cell and can share a bottleneck
- **c** The control interface updates this cell at runtime

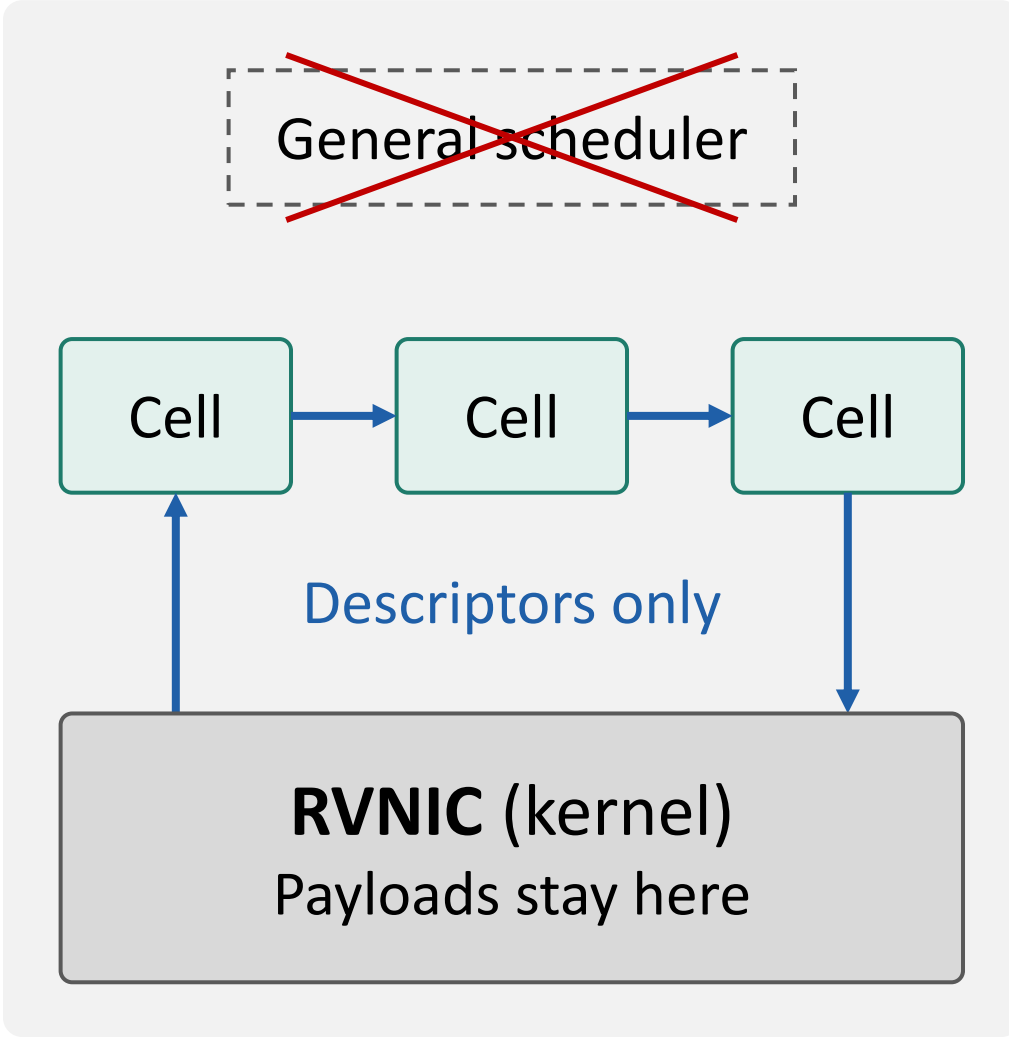
2 Efficient Execution of the Cell Graph

- **RVNIC**, our virtual NIC in the kernel
  - Cells pass descriptors, not payloads
- **Graph-aware scheduling**



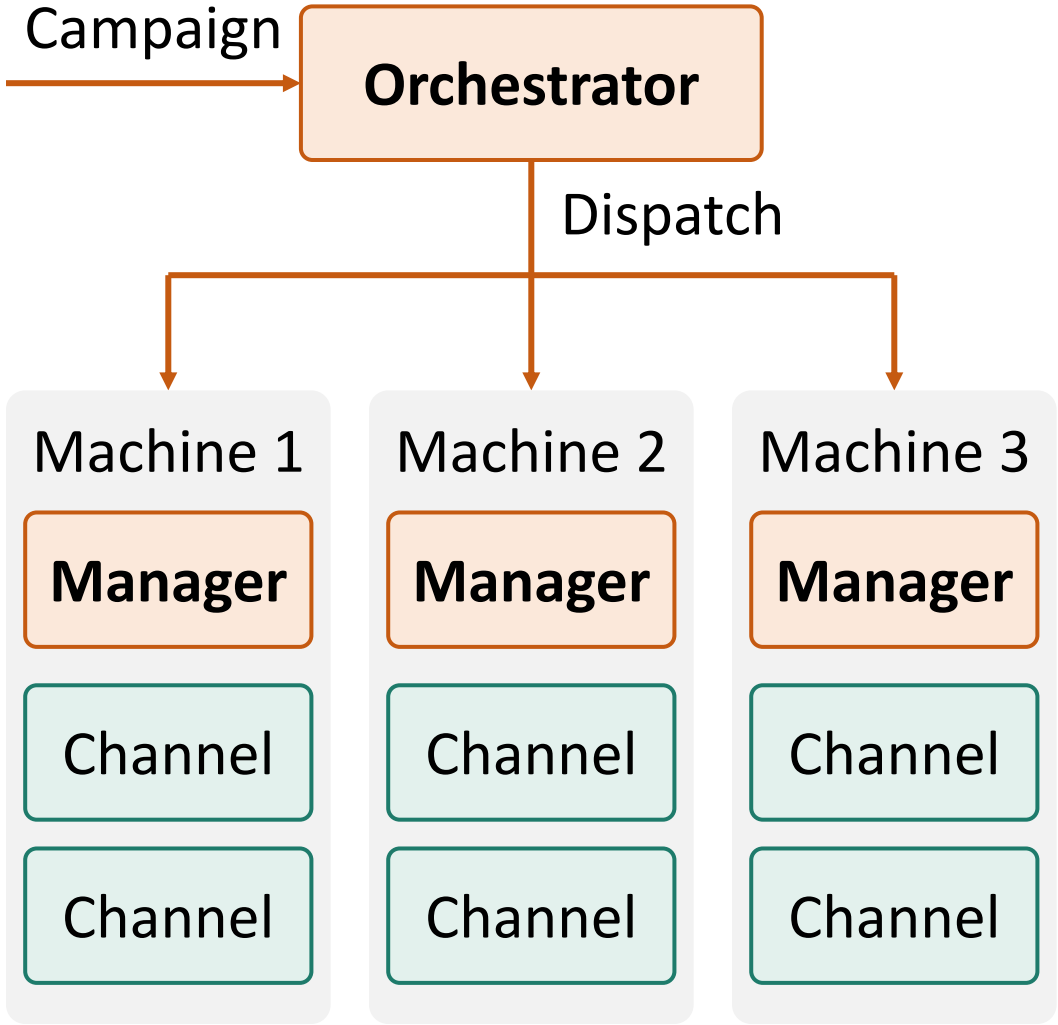
2 Efficient Execution of the Cell Graph

- **RVNIC**, our virtual NIC in the kernel
  - Cells pass descriptors, not payloads
- **Graph-aware scheduling**
  - The cell graph is fixed for a run
  - Descriptors follow its edges directly
- **Offline core placement**
  - A campaign reruns the same config
  - Cells are placed by measured cost



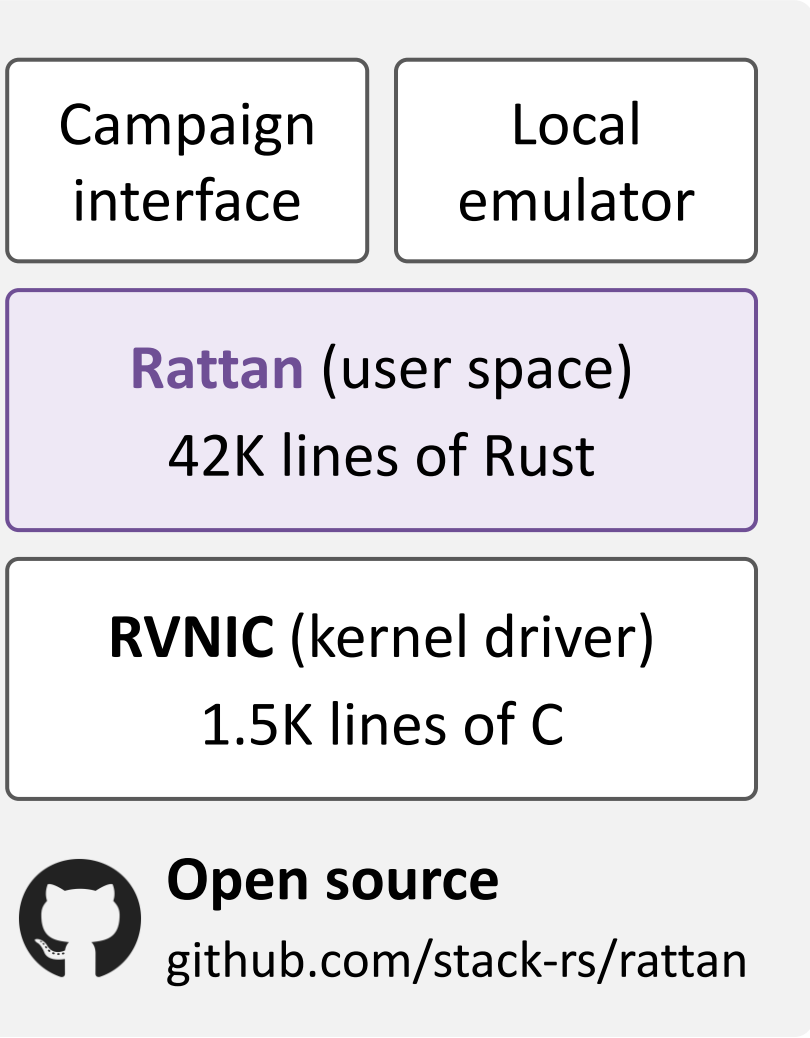
3 Managing and Scaling Channels

- A channel is a self-contained graph
- The **manager** provisions isolated channels on one machine
  - It records each channel as a reproducible artifact
- The **orchestrator** dispatches a campaign across many machines



Implementation

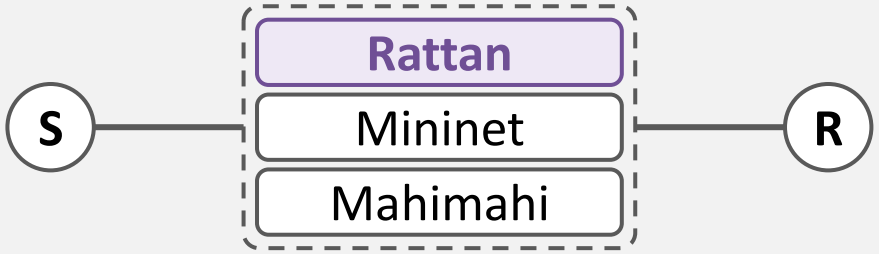
- Supports Linux kernel 5.4 or later
- Two entry points for users
  - Campaign interface to a worker cluster
  - Single binary with a Mahimahi-style shell
- Deployed in production for 21 months
  - At a large CDN provider, for TCP and QUIC



Evaluation

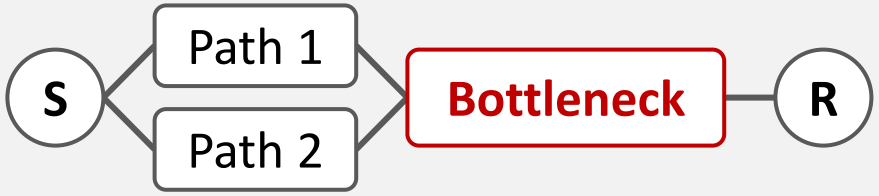
Microbenchmarks

Performance against Mininet and Mahimahi



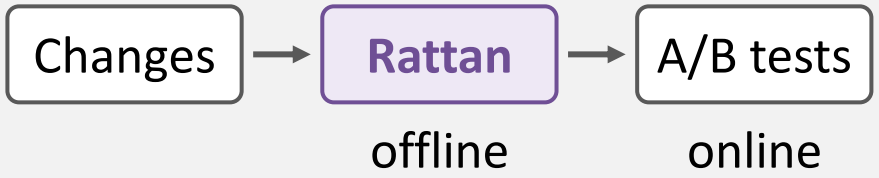
Multipath case study

Expressiveness on production scenarios



Production deployment

Practicality at a large CDN provider



Microbenchmark Setup

- Same path in **Rattan**, Mininet and Mahimahi
- One TCP CUBIC flow per path, all on one machine
  - Intel Core i7-10700F, 64 GiB memory, Linux 6.12

Throughput

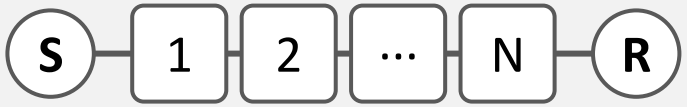
Can one path keep up with a fast link?

0.1 → 16 Gbps

Path complexity

What does each added stage cost?

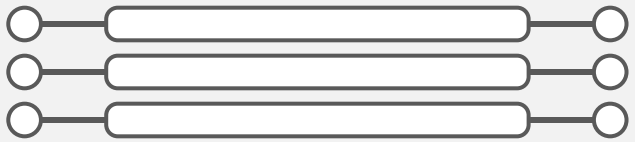
1 → 24 stages



Density

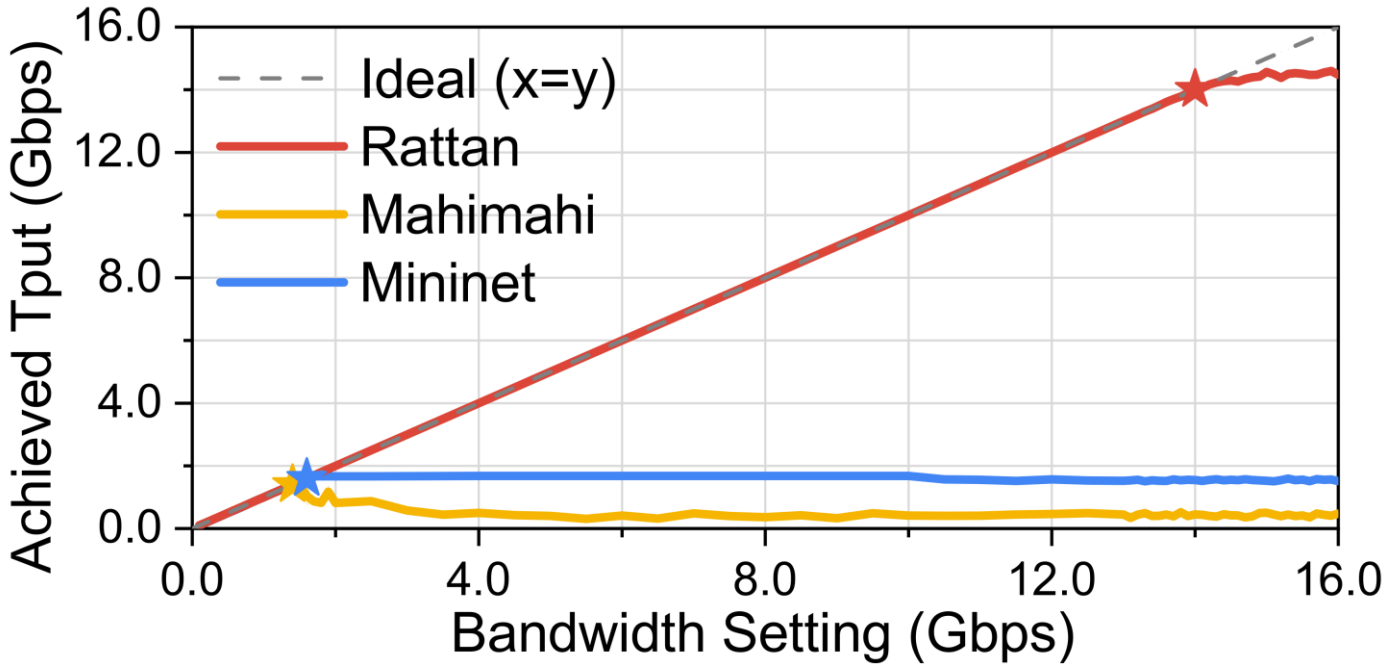
How many paths fit on one machine?

8 → 512 paths



Single-Path Throughput

Rattan reaches 14 Gbps, 8.7–10× that of Mininet and Mahimahi



Rattan
14 Gbps

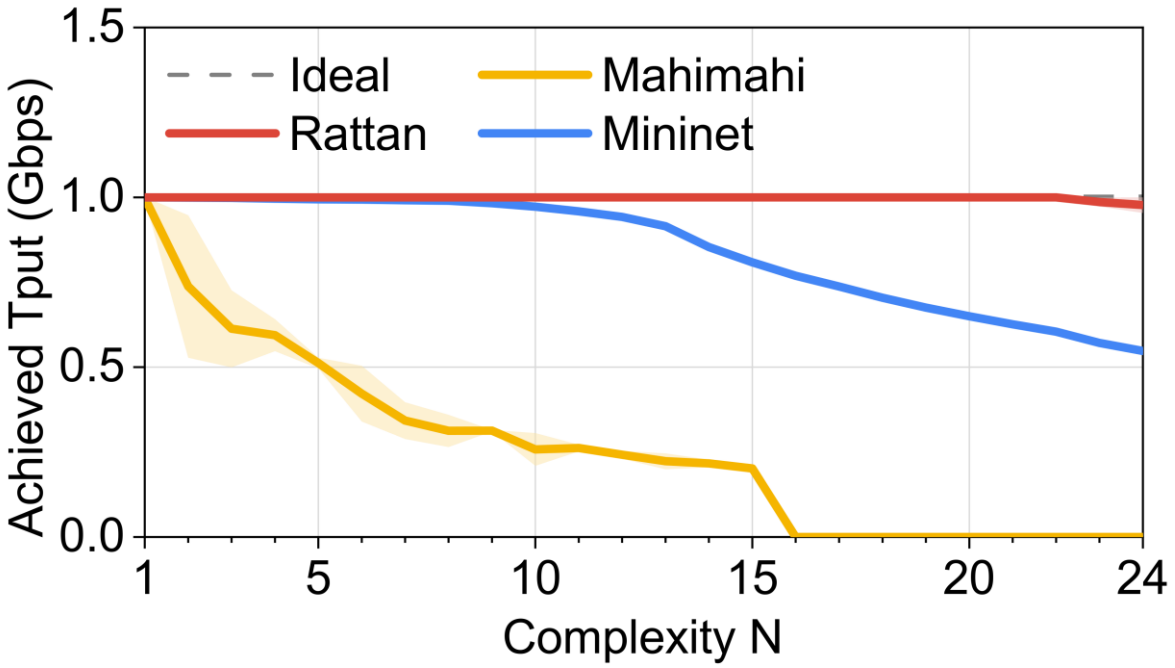
Mininet
1.6 Gbps

Mahimahi
1.4 Gbps

One TCP CUBIC flow, 50 ms RTT, 6 cores; mean ± std of 5 runs

Path Complexity

Rattan holds 1 Gbps on paths up to 22 stages deep



Rattan
1 Gbps up to N = 22

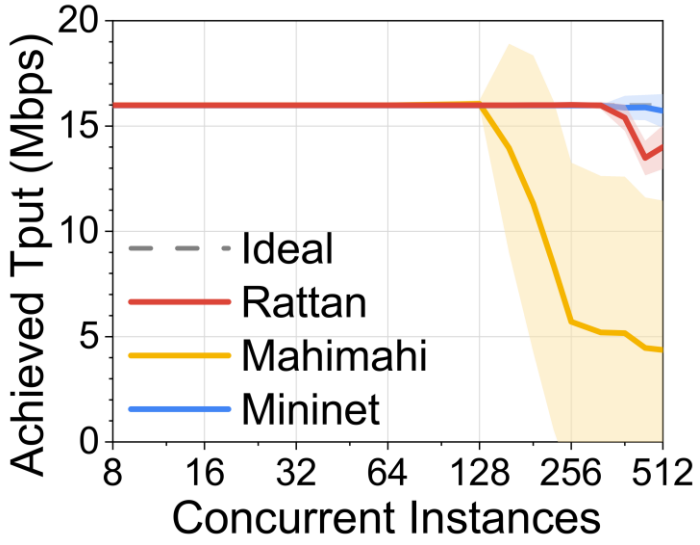
Mininet
drops beyond N = 8

Mahimahi
drops from N = 2

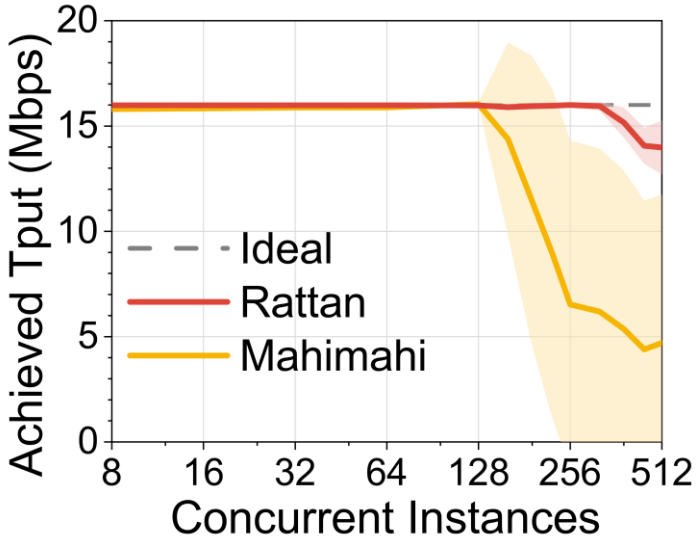
Stage: 0% loss, 1 Gbps, 1 ms delay; one TCP CUBIC flow, 4 cores

Density and Reproducibility

Rattan keeps 320 channels stable on 8 cores, even in trace replay



(a) Static bandwidth



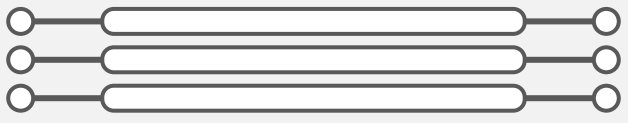
(b) **Trace-driven** bandwidth

At 320 channels

Rattan
15.98 ± 0.02 Mbps

Mininet
no trace replay

Mahimahi
5.15 ± 7.43 Mbps

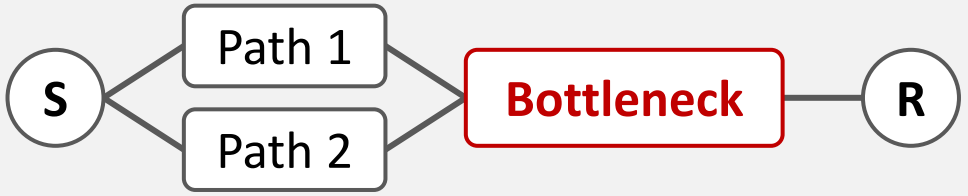


Each path: 16 Mbps, 40 ms RTT, one TCP CUBIC flow; 512 trials

Multipath Case Study

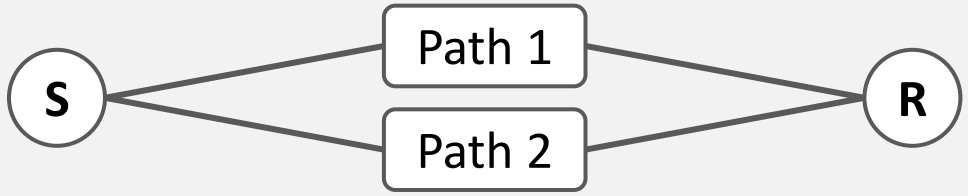
- MPTCP with 3 schedulers × 6 CCAs over 4 path scenarios
- The best combination depends on the path

A: shared bottleneck



Flow completion time (50 MiB, ECF scheduler)

B: independent paths



Flow completion time (50 MiB, ECF scheduler)

What Existing Emulators Miss

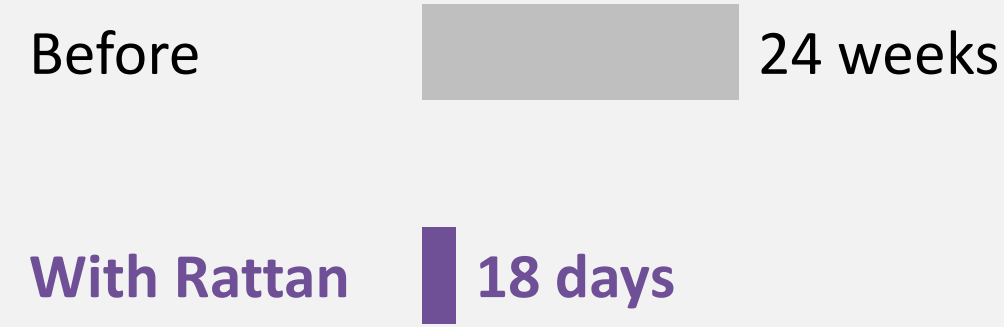
- These scenarios need merging paths and trace replay together

	Shared bottleneck Merging paths	**Trace replay** Time-varying conditions
Mininet	✓ any topology	✗ no trace replay
Mahimahi	✗ linear paths only	✓ trace-driven bandwidth
Rattan	✓ **any graph of cells**	✓ **bandwidth, delay, loss traces**

Deployment at a Large CDN Provider

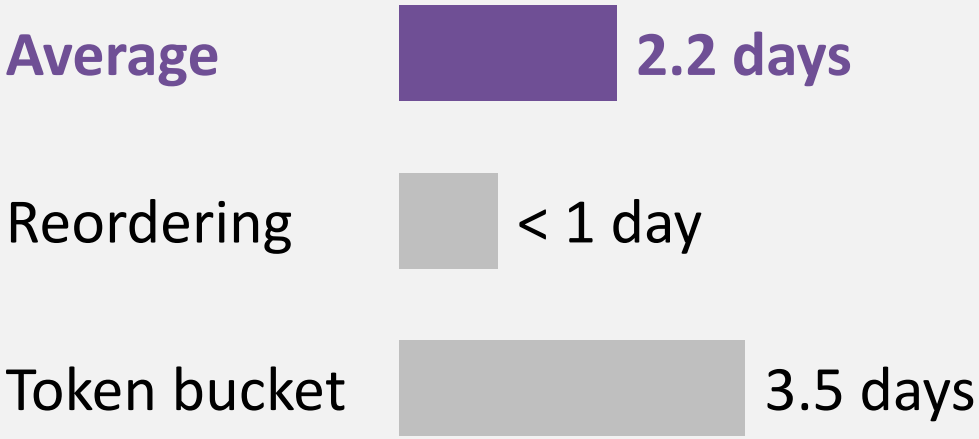
- Used in production for 21 months, running 20 million experiments
- Offline conclusions agree with online A/B tests

Rollout cycle of a CCA change



Fewer candidates need online A/B tests

Time to develop a custom cell



12 custom cells added over 21 months

Conclusion

1. **Offline emulation complements online A/B testing**

   Only promising changes reach production, easing the validation bottleneck

2. **Coupling expression and execution forces an avoidable trade-off**

   With one unit for both, emulators trade expressiveness against efficiency

3. **Rattan decouples expression from execution**

   It provides expressiveness, efficiency and reproducible scale at once

Thank You!

Questions?

Paper: Rattan: An Extensible and Scalable Modular Internet Path Emulator

Code: github.com/stack-rs/rattan

Artifact: doi.org/10.5281/zenodo.23267891

Contact: syx@ieee.org, minhuw@acm.org, thx24@mails.tsinghua.edu.cn