SketchVisor: Robust Network Measurement for Software Packet Processing

<u>Qun Huang</u>, Xin Jin, Patrick P. C. Lee,

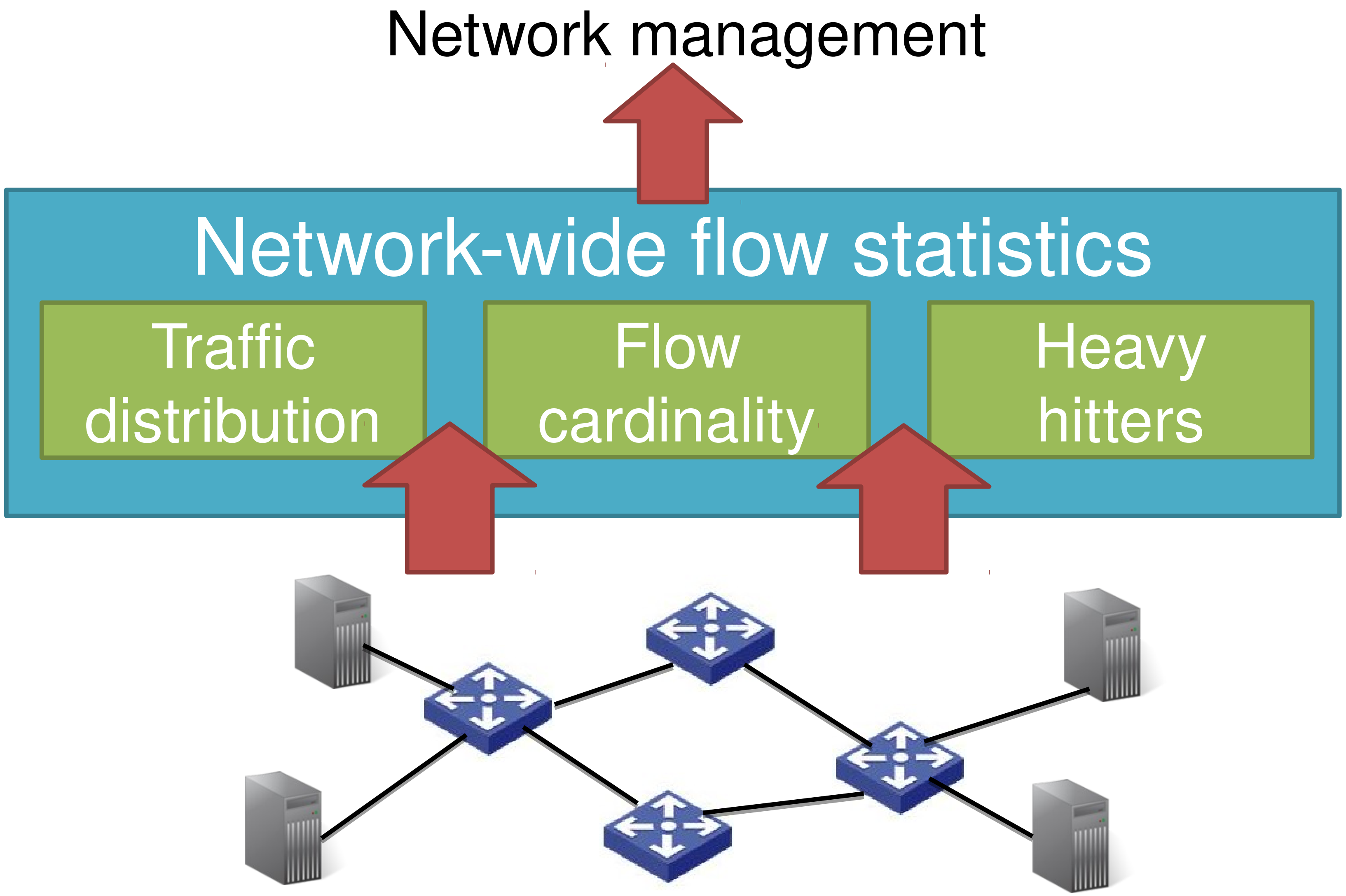
Runhui Li, Lu Tang, Yi-Chao Chen, Gong Zhang

HUAWEI

JOHNS HOPKINS UNIVERSITY

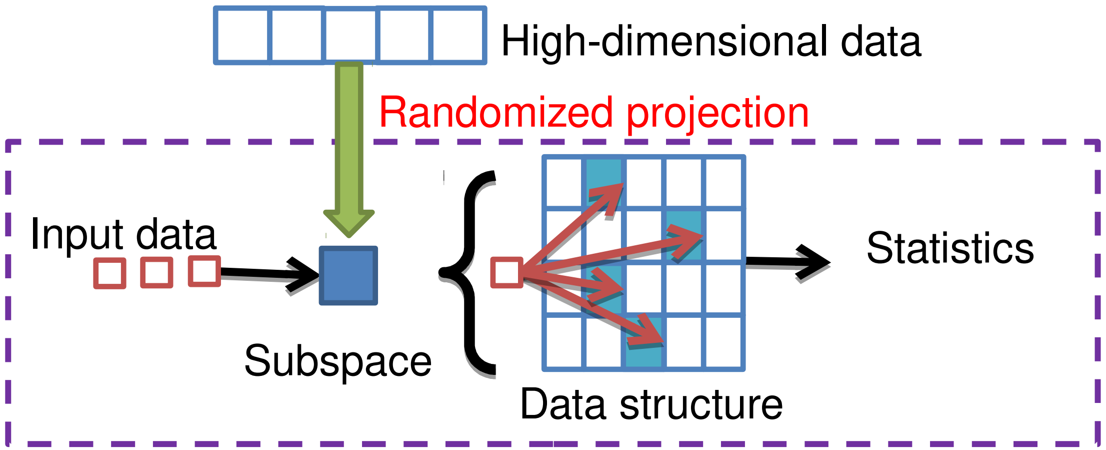
香港中文大學
The Chinese University of Hong Kong

Monitoring Traffic Statistics



Sketch: A Promising Solution

- Sketch: a family of randomized algorithms
  - Key idea: project high-dimensional data into small subspace

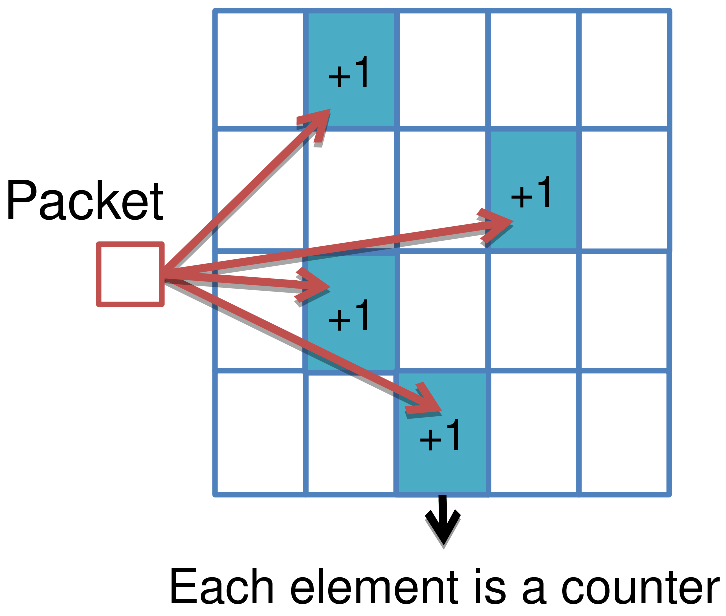


Small subspace:
low computation & communication overheads

- Subspace reflects mathematical properties
  - Strong theoretical error bounds when querying for statistics

Example: Count-Min Sketch

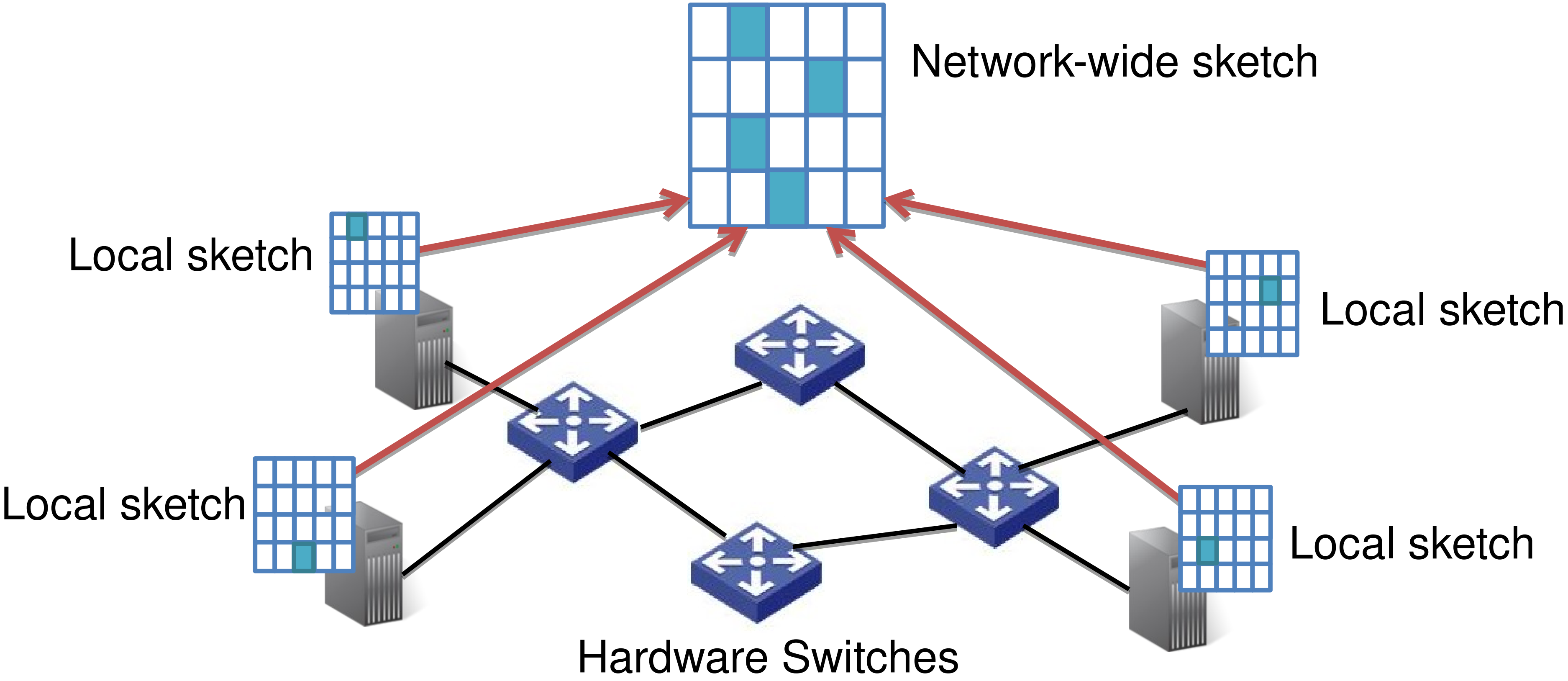
- Count flow packets



- Update with a packet
  - Hash flow id to one counter per row
  - Increment each selected counter
- Query a flow
  - Hash the flow to multiple counters
  - Take the minimum counter as estimated packet count
- Theoretical guarantees
  - Allocate rows and counters each row
  - The error for a flow is at most with probability at least

Our Focus

- Sketch-based measurement atop software switches



Limitation of Sketches

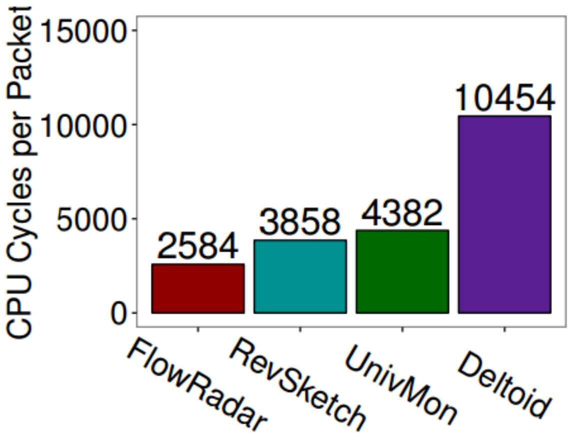
Basic sketches

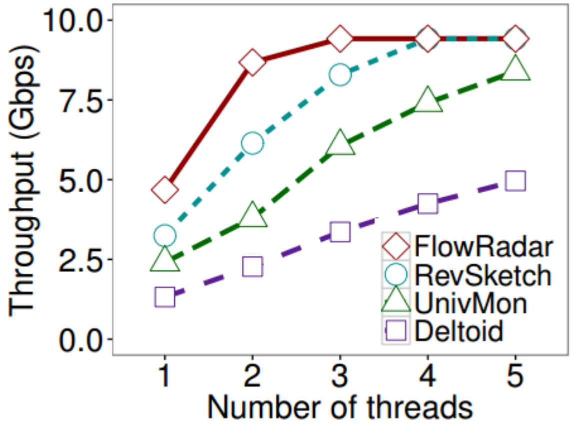
Lack of generality

Limited query

More structures

Complicated sketches





Our Contributions

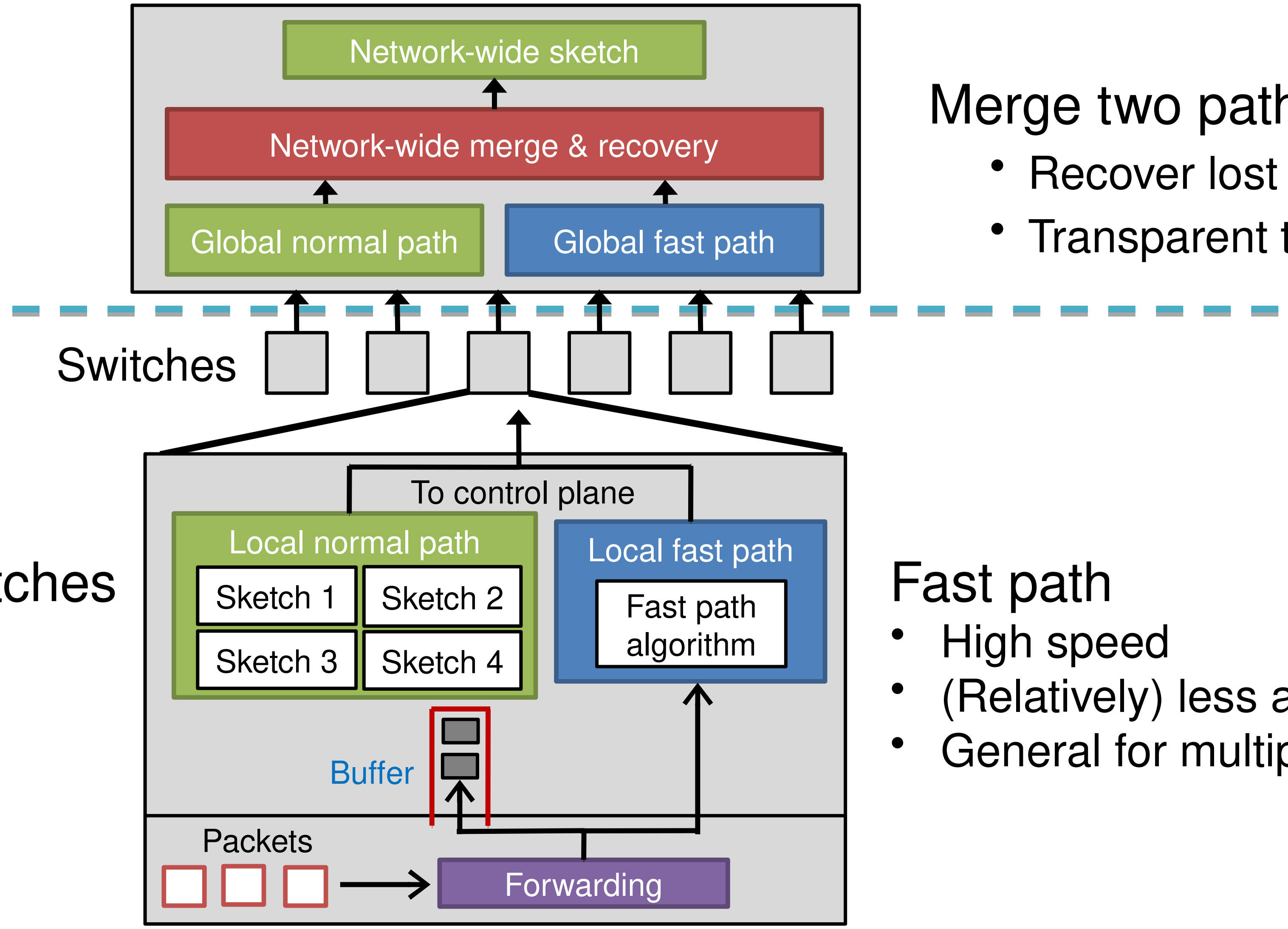
SketchVisor: Sketch-based Measurement System for Software Packet Processing

- Performance
  - Catch up with underlying packet forwarding speed
- Resource efficiency
  - Consume only limited resources
- Accuracy
  - Preserve high accuracy of sketches
- Generality
  - Support multiple sketch-based algorithms
- Simplicity
  - Automatically mitigate performance burdens of sketches without manual tuning

Architecture: Double-Path Design

Control plane

Data plane



Merge two paths

- Recover lost information
- Transparent to users

User-defined sketches

- High accuracy
- (Relatively) slower

Fast path

- High speed
- (Relatively) less accurate
- General for multiple sketches

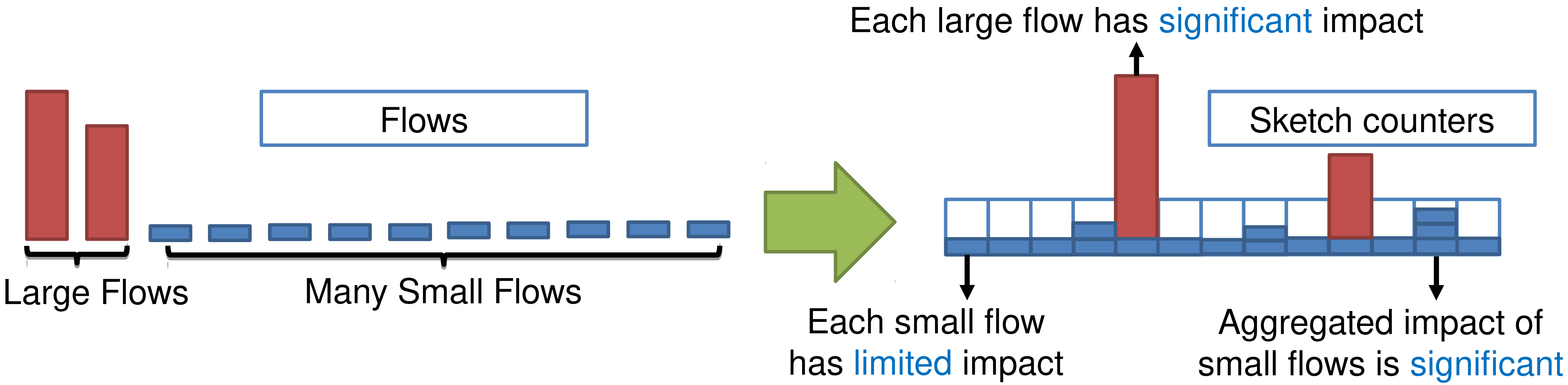
Key Questions

- Data plane: how to design the fast path algorithm?

- Control plane: how to merge the normal path and fast path?

Intuitions

- Consider sketches which map flow byte counts into counters
  - Other sketches (e.g., Bloom Filter) can be converted



Fast Path Algorithm

Ideal algorithm

Infeasible with limited resources

- Per-flow byte count of large flows
- Aggregated byte count of small flows

Our practical algorithm

- How: (Approximate) per-flow byte count of large flows
- Easy: (Approximate) aggregated byte count of small flows

Byte of small flows = total byte – byte of large flows

Approximate Tracking of Large Flows

- A small hash table
  - "Guess" and kick out potentially small flows when table is full
  - Each flow has three counters

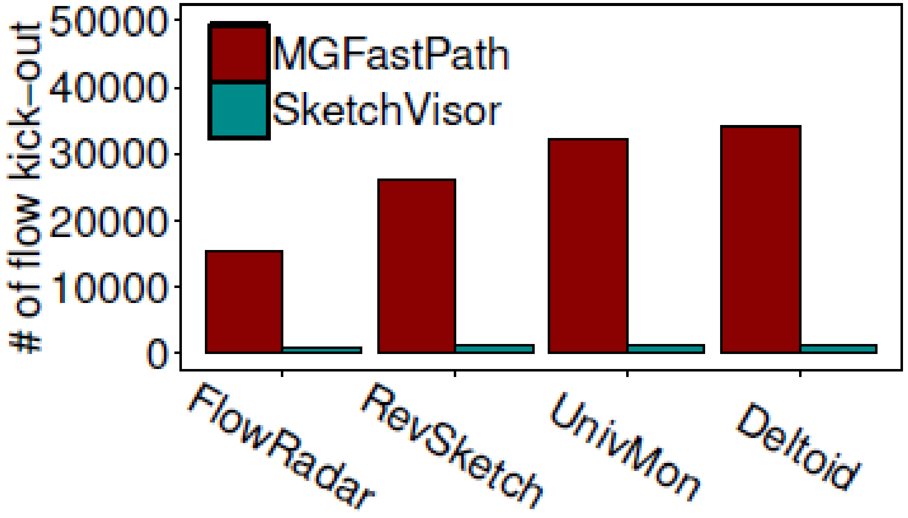
Byte count (Counter 1)

Estimated errors due to flow kick-outs (Counter 2, Counter 3)

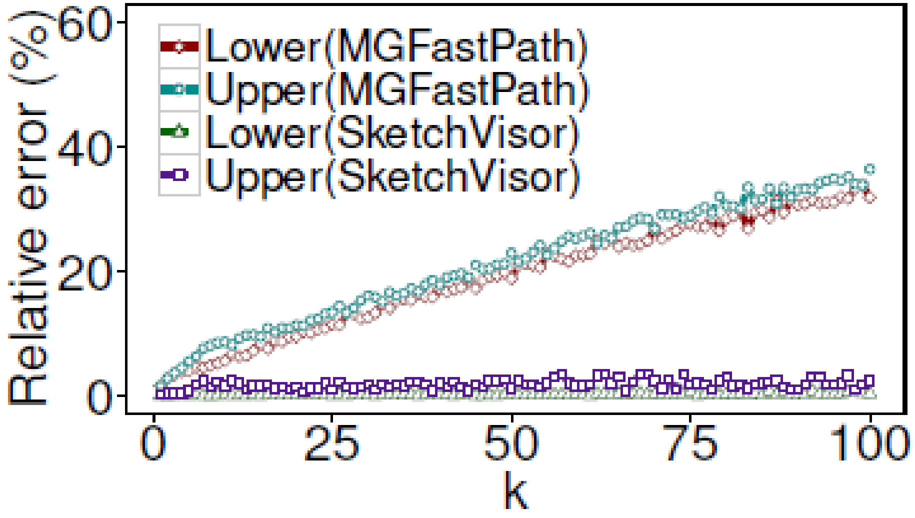
Flow ID	Counter 1	Counter 2	Counter 3
Flow 1	4	0	1
Flow 2	1	0	1
Flow 3	2	0	1

Performance and Accuracy

- Theoretical analysis shows:
  - All large flows are tracked
  - Amortized O(1) processing time per packet
  - Bounded errors
- Compared to Misra Gries top-k algorithm



(a) Number of flow kick-outs



(b) Errors of top-k flows

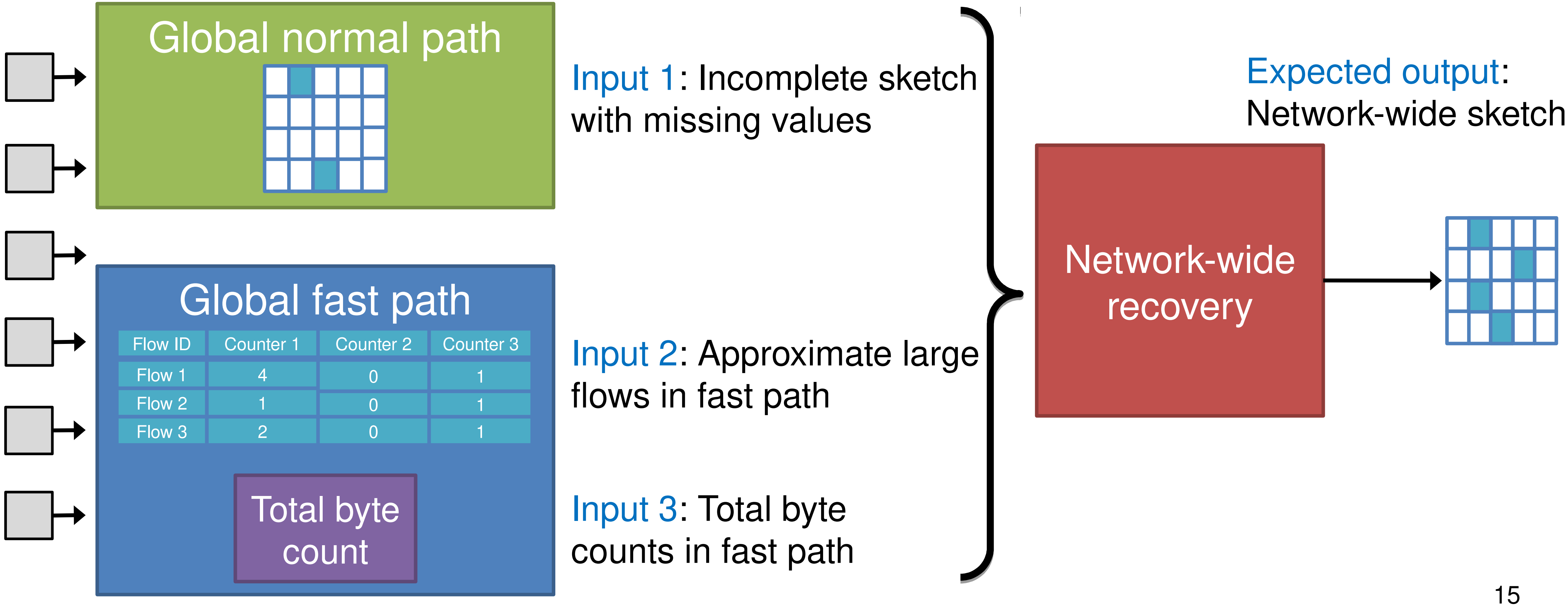
Key Questions

- Data plane: how to design a fast path algorithm?

- Control plane: how to merge the normal path and fast path?

Control Plane: Challenge

- Input insufficient to form network-wide sketches



Flow ID	Counter 1	Counter 2	Counter 3
Flow 1	4	0	1
Flow 2	1	0	1
Flow 3	2	0	1

Matrix Interpolation Problem

- The recovery process can be expressed as

Expected output sketch (unknown)

Large flows in fast path (unknown)

$$T = N + sk(\mathbf{x} + \mathbf{y})$$

Sketch in global normal path (known)

Small flows in fast path (unknown)

Matrix Interpolation Problem

➢ Based on theoretical analysis and microbenchmarks

Expected output sketch (unknown)

Large flows in fast path (unknown)

$$T = N + sk(\mathbf{x} + \mathbf{y})$$

Sketch in global normal path (known)

Small flows in fast path (unknown)

Matrix Interpolation Problem

- Based on theoretical analysis and microbenchmarks

(low-rank structure)

Expected output sketch (~~unknown~~)

Large flows in fast path (unknown)

$$T = N + sk(\mathbf{x} + \mathbf{y})$$

Sketch in global normal path (known)

Small flows in fast path (unknown)

Matrix Interpolation Problem

- Based on theoretical analysis and microbenchmarks

(low-rank structure)

Expected output sketch (~~unknown~~)

(1. sparse vector)
(2. each flow is bounded)

Large flows in fast path (~~unknown~~)

$$T = N + sk(\mathbf{x} + \mathbf{y})$$

Sketch in global normal path (known)

Small flows in fast path (unknown)

Matrix Interpolation Problem

- Based on theoretical analysis and microbenchmarks

(low-rank structure)

Expected output sketch (~~unknown~~)

(1. sparse vector)
(2. each flow is bounded)

Large flows in fast path (~~unknown~~)

$$T = N + sk(\mathbf{x} + \mathbf{y})$$

Sketch in global normal path (known)

Small flows in fast path (~~unknown~~)

(small and close values)

Matrix Interpolation Problem

- Based on theoretical analysis and microbenchmarks

(low-rank structure)

Expected output sketch (~~unknown~~)

(1. sparse vector)
(2. each flow is bounded)

Large flows in fast path (~~unknown~~)

Total traffic is known

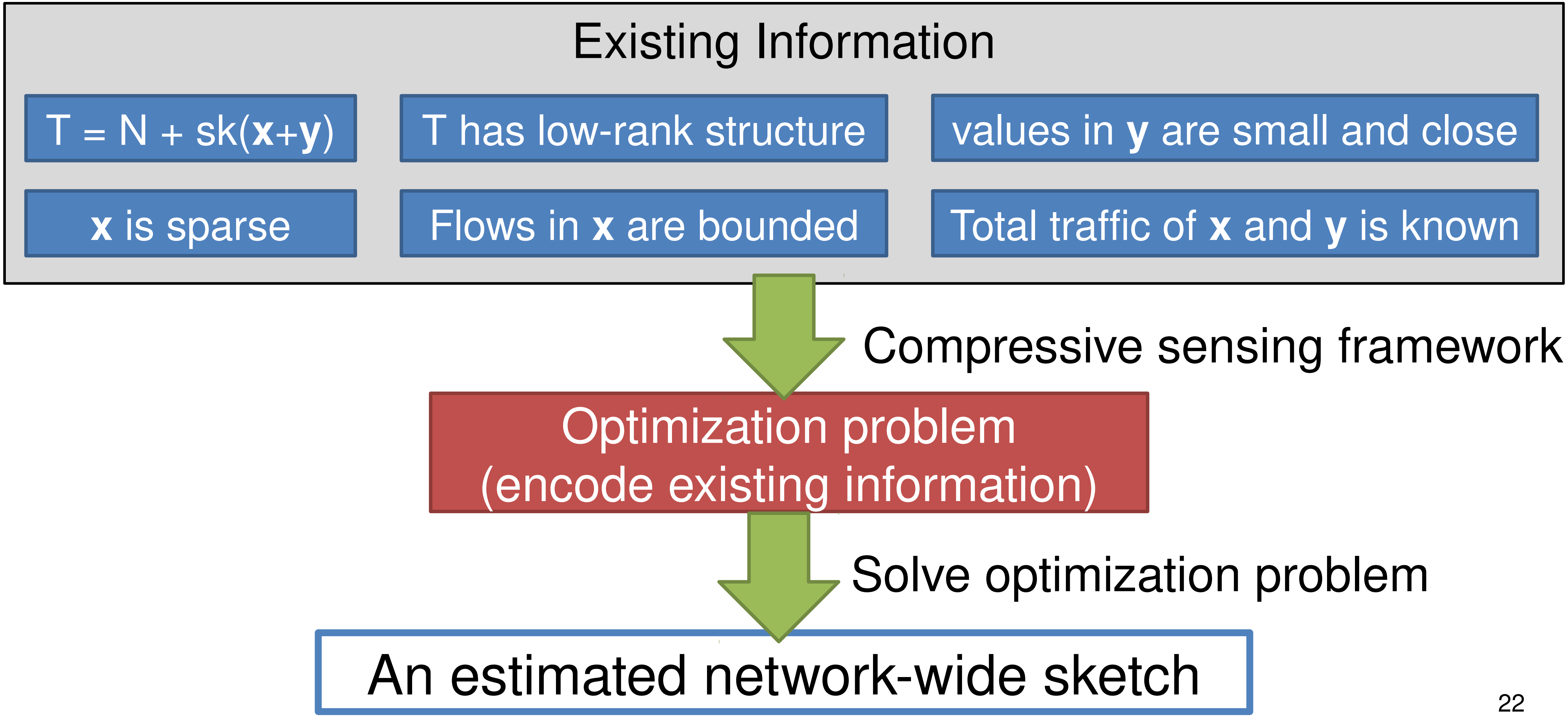
$$T = N + sk(\mathbf{x} + \mathbf{y})$$

Sketch in global normal path (known)

Small flows in fast path (~~unknown~~)

(small and close values)

Recovery Approach



Evaluation

Evaluation Setup

- Prototype based on OpenVSwitch
- Environments
  - Testbed: 8 OVS switches connected by one 10Gbps hardware switch
  - In-memory simulation: 1 – 128 simulation processes
- Workloads: CAIDA

Measurement tasks

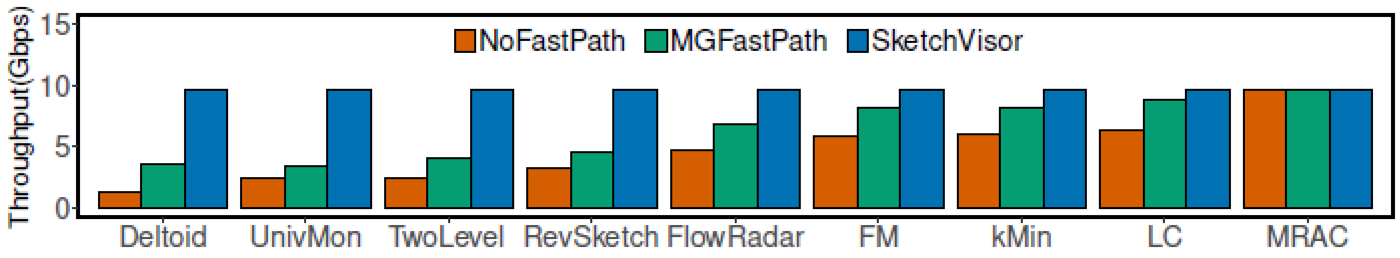
Heavy hitter detection | Heavy changer detection | Superspreader detection

DDoS detection | Cardinality estimation | Entropy estimation

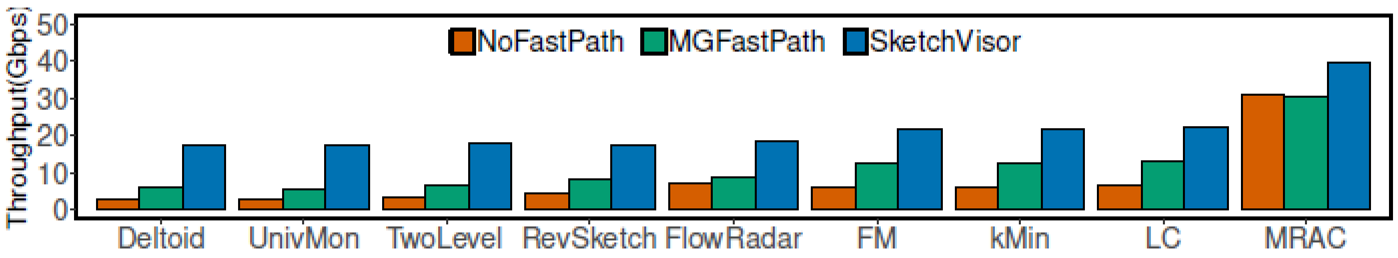
Flow distribution estimation

Throughput

- Compared with two data plane approaches
  - NoFastPath: use only Normal Path to process all traffic
  - MGFastPath: use Misra-Gries Algorithm to track large flows in Fast Path
- Achieve ~10 Gbps in testbed (single CPU core)

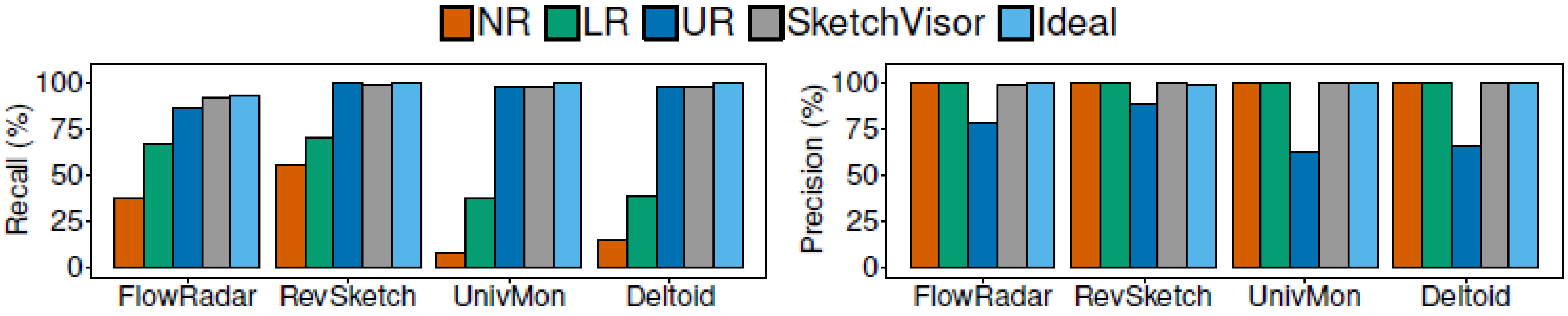


- Achieve ~20 Gbps in simulation (single CPU core)



Accuracy

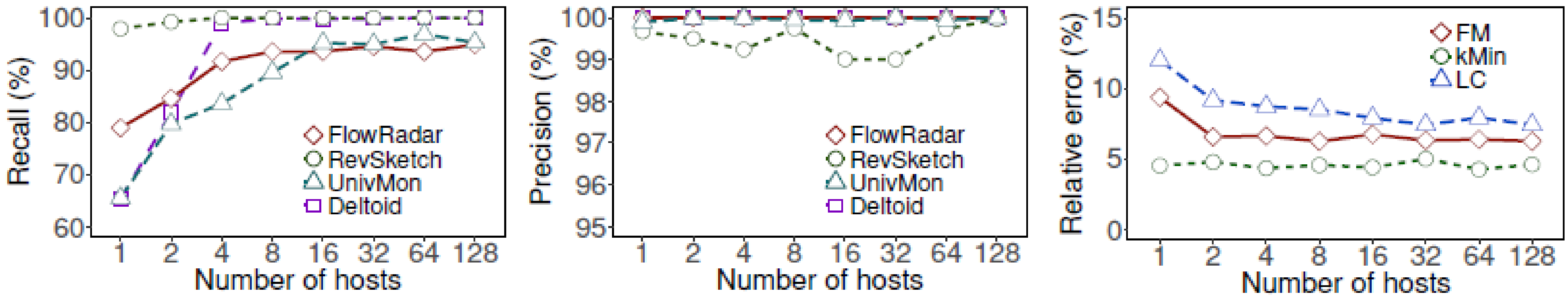
- Compare with four recovery approaches
  - Ideal: an oracle to recover the perfect sketch
  - NR: no recovery at all
  - LR: only use lower estimate of large flows in Fast Path
  - UR: only use upper estimate of large flows in Fast Path
- SketchVisor matches the ideal approach



(a) HH Recall (b) HH Precision

Network-wide Results

- Recover sketch from 1-128 hosts
- Accuracy improved as number of hosts increases



(a) HH recall (b) HH precision (d) Cardinality error

- Work for both byte-based tasks (heavy hitter detection) and connection-based tasks (cardinality estimation)

Conclusion

- SketchVisor: high-performance system for sketch algorithms
- Double-path architecture design
  - Slower and accurate sketch channel (normal path)
  - Fast and less accurate channel (fast path)
- Fast path algorithm in data plane
  - General and high performance
- Recovery in control plane
  - Achieve high accuracy using compressive sensing
- Implementation and evaluation
  - OpenVSwitch based implementation
  - Trace-driven experiments