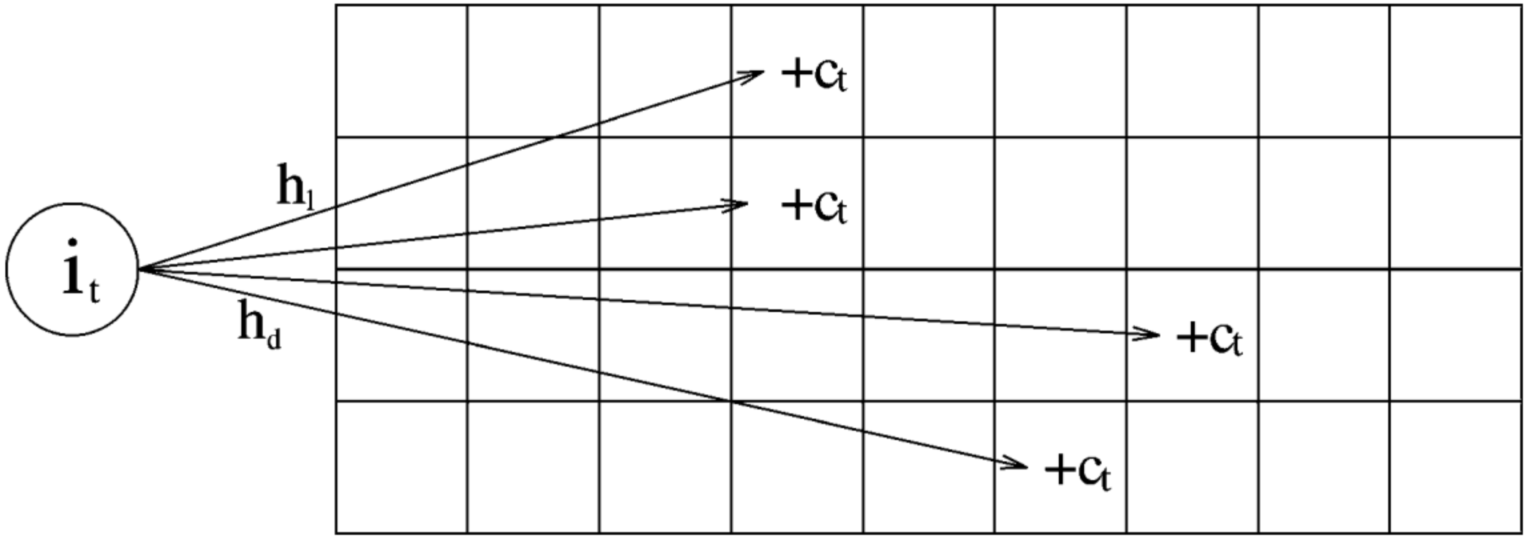
Toward Nearly-Zero-Error Sketching via Compressive Sensing

Qun Huang, Siyuan Sheng, Xiang Cheng,

Yungang Bao, Rui Zhang, Yanwei Xu, Gong Zhang

Sketches

- Data Stream Summarizing Techniques for Utilizing Limited Resources
- Count-based Measurements (or Point Queries)
- Hash Table Data Structure

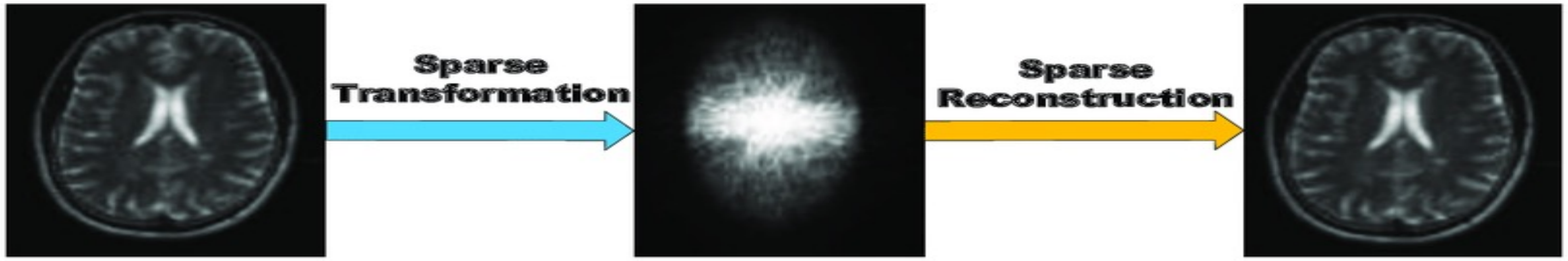


Compressive Sensing

- A Signal Processing Method for Acquiring and Reconstructing Signals
- Suitable for Sparse Signals
- Using An Optimization-based Approach for The Recovery Procedure:

Minimize: $||\vec{x}||_1$,

Subject to: $\vec{y} = \phi\vec{x}$



Matrix Orthonormality

- **Orthonormal Vectors:**

$$\vec{x}.\vec{y} = 0, \qquad \|\vec{x}\|_2^2 = 1, \qquad \|\vec{y}\|_2^2 = 1$$

- **Orthonormal Matrix:** A Matrix With Pairwise Orthonormal Columns
- Any Orthonormal Matrix Preserves Differences for Sparse Vectors

  - For Distinct Sparse Vectors $\vec{x}_1$ and $\vec{x}_2$ and Orthonormal Matrix ϕ , $\phi\vec{x}_1$ and $\phi\vec{x}_2$ remain distinct.

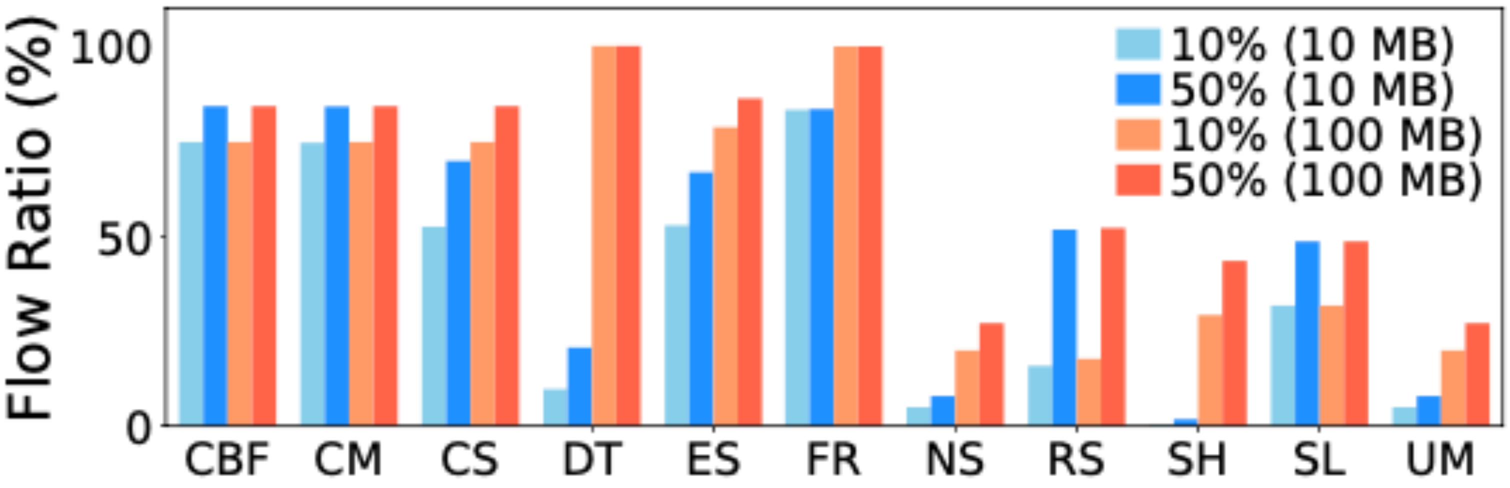
- Restricted Isometry Property (RIP) Characterizes The Extent to Which A Matrix Preserves The Norm of Sparse Signals:

$$\delta_S = \sup\{\frac{\|\phi\vec{x}\|_2 - \|\vec{x}\|_2}{\|\vec{x}\|_2} \ for\ any\ S - Sparse\ \vec{x}\}$$

Limitations of Prior Work

- Their Assumptions:

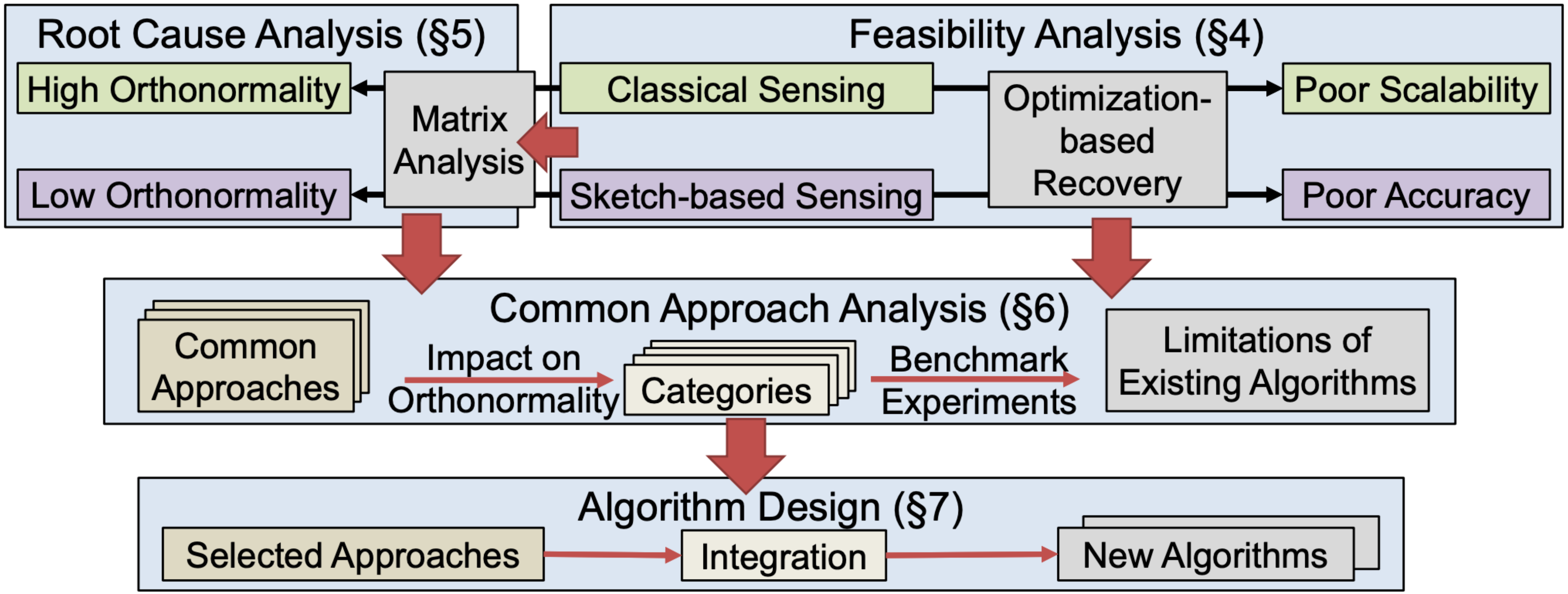
1. It is Sufficient to Address Large Flows
2. Approximate Monitoring is Acceptable



Key Questions

- Is NZE Monitoring Theoretically Feasible?
- What Are The Key Factors To Achieve NZE Monitoring?
- How Do The Key Factors Can be Efficiently Realized in Practice?

Workflow



Root Cause Analysis (§5)
High Orthonormality
Low Orthonormality
Matrix Analysis
Feasibility Analysis (§4)
Classical Sensing
Sketch-based Sensing
Optimization-based Recovery
Poor Scalability
Poor Accuracy
Common Approach Analysis (§6)
Common Approaches
Impact on Orthonormality
Categories
Benchmark Experiments
Limitations of Existing Algorithms
Algorithm Design (§7)
Selected Approaches
Integration
New Algorithms

Classical Sensing

- They Use Four Types of Commonly Used Sensing Matrices:

Gaussian Matrix, Bernoulli Matrix, Incoherence Matrix, and Fourier Matrix

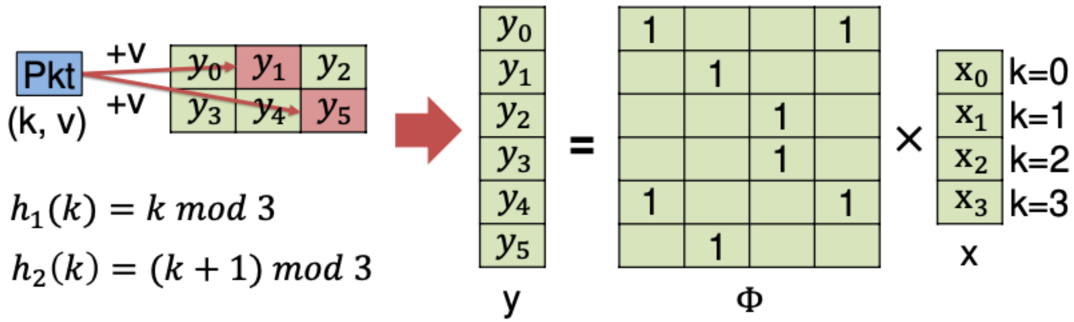
- They Use Two Algorithms for Reconstructing The Flows:

Simplex Method and Orthogonal Matching Pursuit

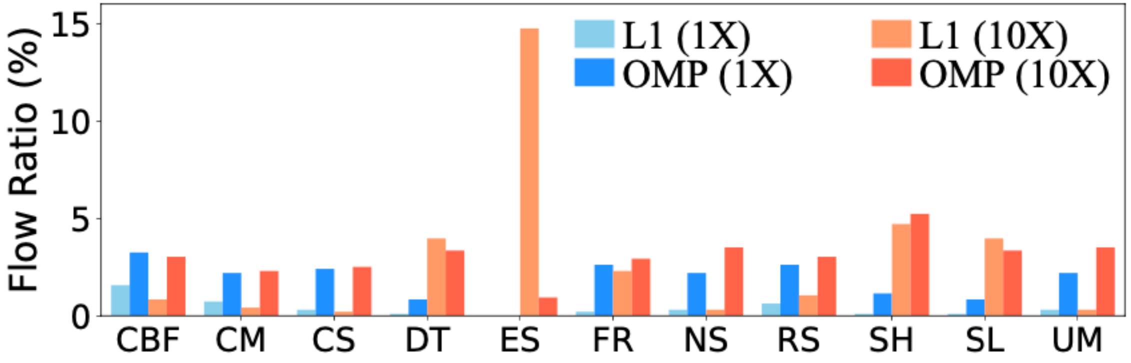
- By Using 400KB Memory, Perfect Recovery Can Be Achieved
- The Sencing Matrices Are Dense → Above Counter Updates Per Packet
- Slow for Software Switches and Not Feasible for Hardware Switches

Sketch-Based Sensing

- **Linear Structures:**

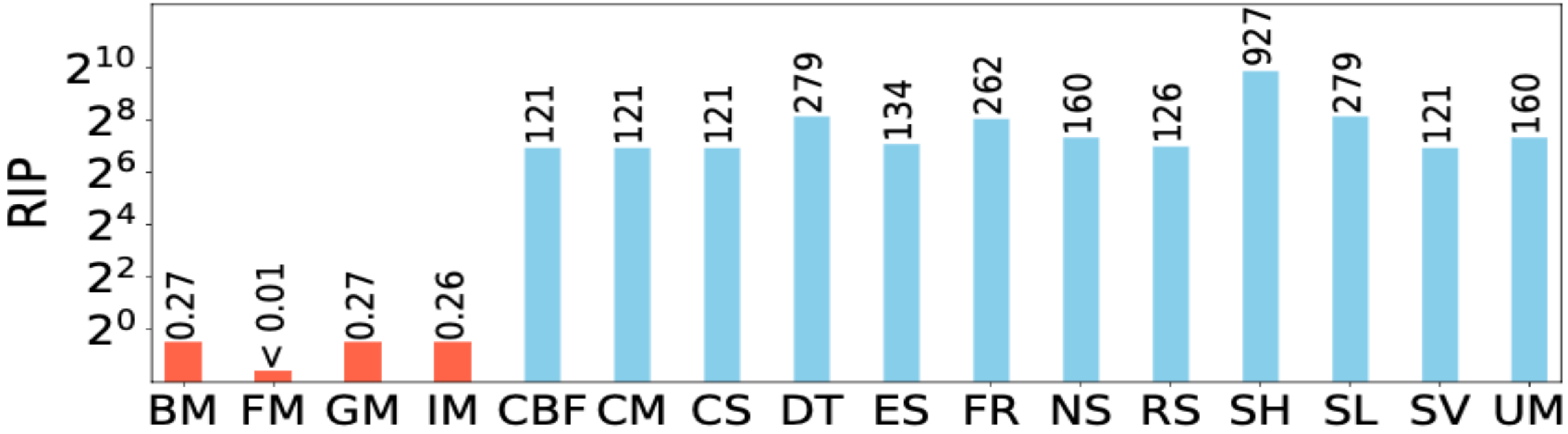


- **NonLinear Structures:** Not Considering The Nonlinear Components and Verifying The Correctness of The Reconstructed Vector By Comparing With The Original One
- **Results:**



Root Causes

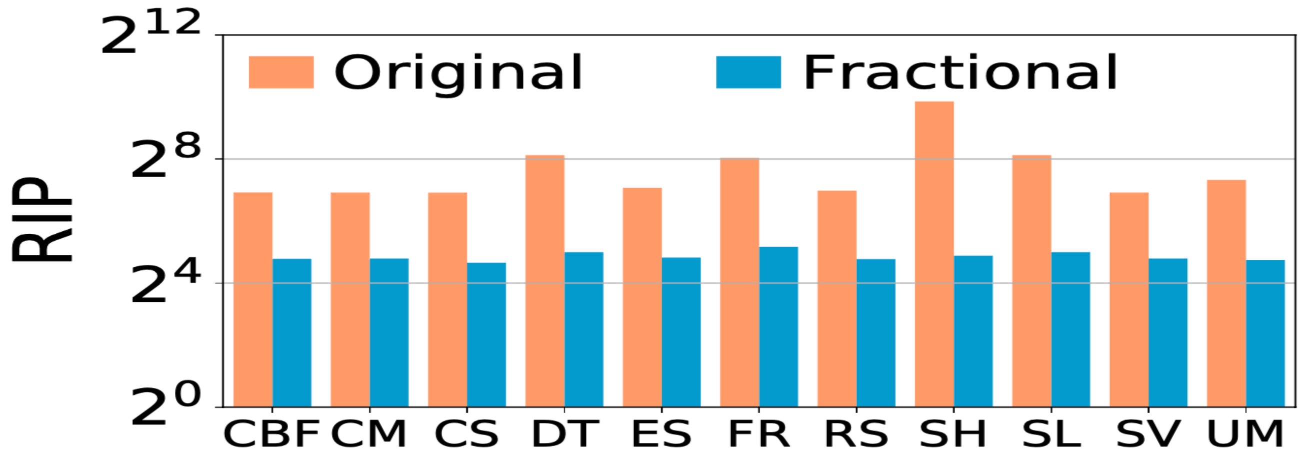
RIP of Classical and Sketch-based Sensing



Common Approach Analysis

Class 1: Fractional Elements

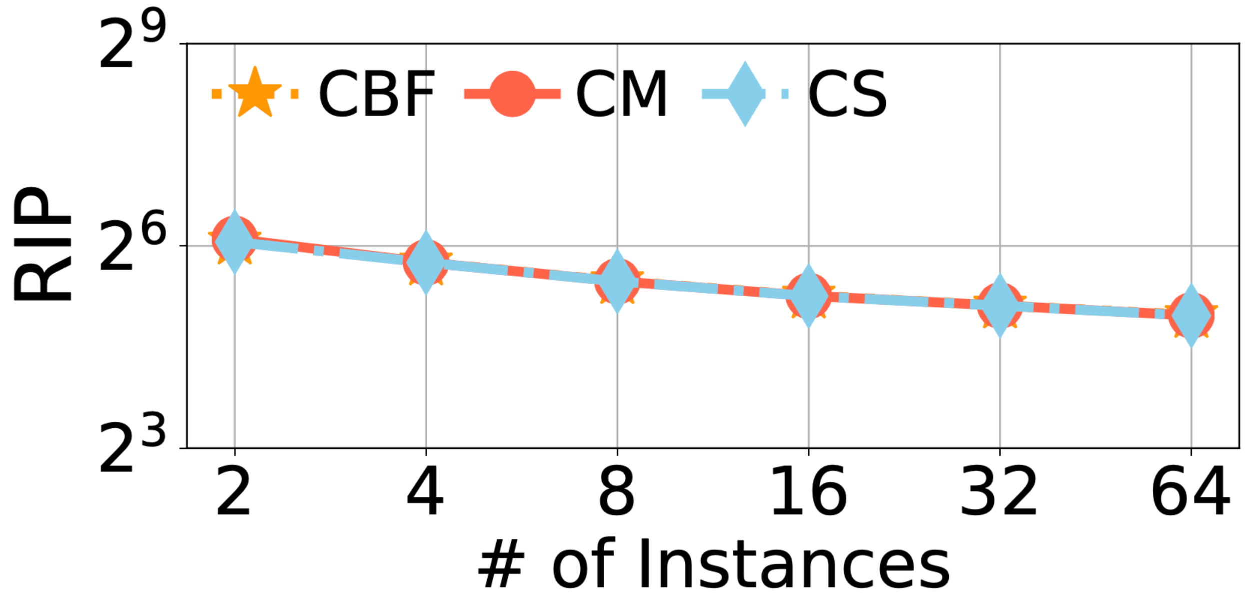
- **Analysis:** By Using Fractional Elements in The Sketch-based Sensing Matrix The Norm of Its Columns Become Closer to 1.
- **Evaluation:** Replacing The Sketch Elements With A Randomized Value $\frac{1}{\sqrt{t}} + \sigma$, Where σ is Sampled From A Gaussian Distribution With Mean 0 and t is The Number of Counters Accessed By A Packet.



Common Approach Analysis

Class 2: Adding Rows

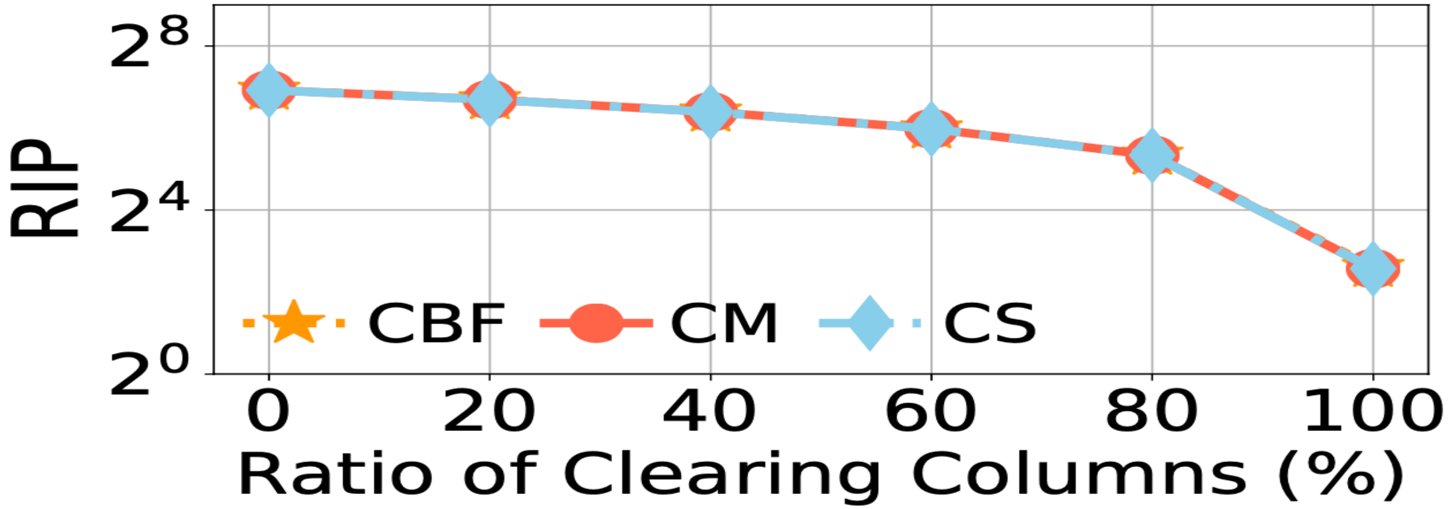
- **Analysis:** Adding Rows (Ideally Adding Sketch Instances) Will Result In Fewer Flow Conflicts
- **Evaluation:** Adding More Sketch Instances to Basic Sketches



Common Approach Analysis

Class 3: Clearing Columns

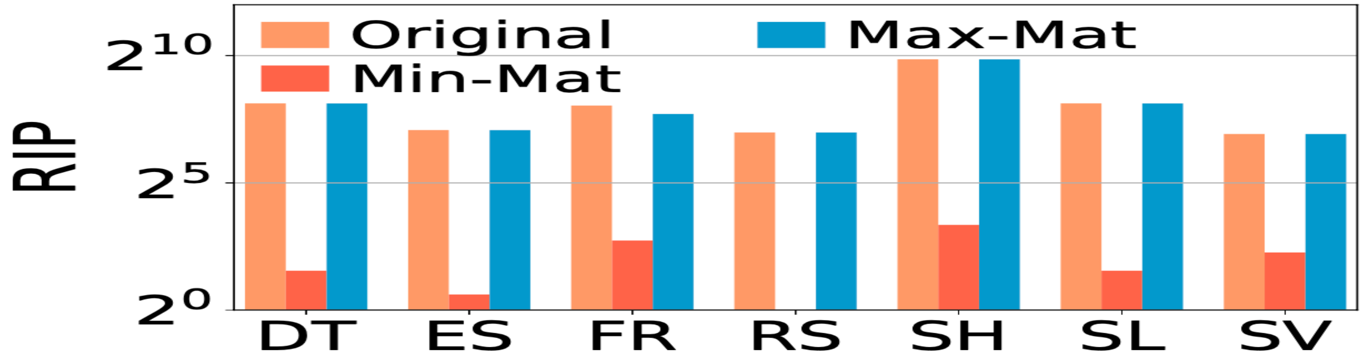
- **Analysis:** Clearing Columns (Corresponding With The Useless Flows) Simplifies The Optimization Problem and Improves Accuracy
- **Evaluation:** Clearing Useless Columns



Common Approach Analysis

Class 4: Matrix Decomposition

- **Analysis:** Decomposing Matrices Will Alleviate Non-Zero Elements By Distributing Them into Different Components
- This Can Be Done By Traffic Splitting or Flow Extraction
- **Evaluation:** Decomposing The Matrix into Minimum RIP (Min-Mat) Component and Maximum RIP (Max-Mat) Component



Common Approach Analysis and New Algorihtms

Algorithm	C1	C2	C3	C4
CU Sketch [25]	Conservative update			
Deltoid [19]		Multiple CM instances		Flow extraction
ElasticSketch [80]				Traffic splitting
FlowRadar [49]		Multiple Bloom Filters	Bloom Filter	Flow extraction
NitroSketch [53]	Sampling	Multiple CS instances	Heap	
RevSketch [68]				Flow extraction
SeqHash [8]		Multiple CM instances		Flow extraction
SketchLearn [37]		Multiple CM instances		Flow extraction
SketchVisor [35]				Traffic splitting
UnivMon [54]		Multiple CS instances	Heap	
SeqSketch	Fractional update		Bloom Filter + Controller	Splitting + Controller
EmbedSketch	Fractional update		Bloom Filter + controller	Extraction + Controller

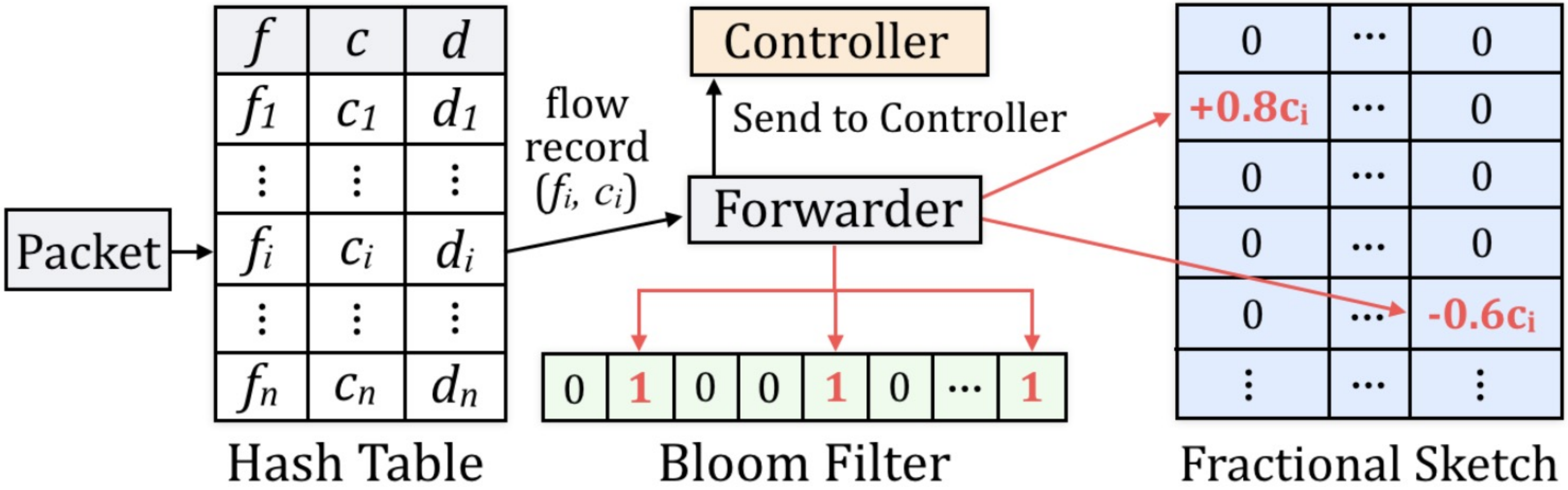
SeqSketch

Algorithm 1 SeqSketch Data Plane

Input: Packet (k, v)

procedure UPDATE(k, v)
    $j = \text{hash}(k)$
    if $H[j]$ is $\emptyset$ **then**
        $H[j].f = k$, $H[j].c = v$, and $H[j].d = 0$
    else if $H[j].f == k$ **then**
        $H[j].c = H[j].c + v$
    else
        $H[j].d = H[j].d + v$
        if $H[j].d > H[j].c$ **then**
            Send $(H[j].f, H[j].c)$ to controller
            $H[j].f = k$, $H[j].c = v$, and $H[j].d = 0$
        else
            for all row i in FS **do**
                Compute $j = h_i(k)$
                Increment counter (i, j) by $g_i(k) \cdot v$
            if $k \notin BF$ **then**
                Send k to controller
                Insert k to BF

SeqSketch


Packet
f
c
d
f_1
c_1
d_1
f_i
c_i
d_i
f_n
c_n
d_n
Hash Table
flow record (f_i, c_i)
Controller
Send to Controller
Forwarder
0 1 0 0 1 0 ··· 1
Bloom Filter
0 ··· 0
+0.8c_i ··· 0
0 ··· 0
0 ··· 0
0 ··· -0.6c_i
Fractional Sketch

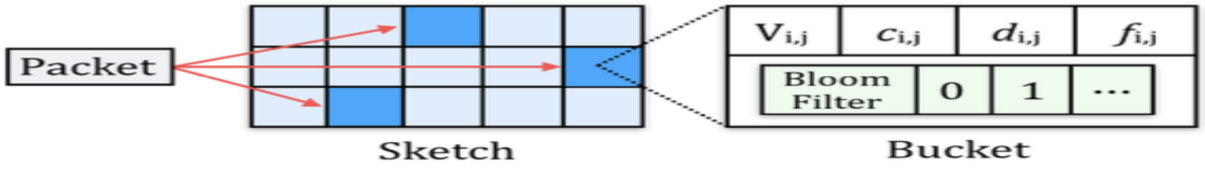
EmbedSketch

Algorithm 2 EmbedSketch Data Plane

Input: Packet (k, v)

function UPDATEBUCKET(k, v, i, j)
    $V_{i,j} = V_{i,j} + g_i(k)$
    if $f_{i,j}$ is empty **then**
        $f_{i,j} = k$, $c_{i,j} = v$, $d_{i,j} = 0$
    else if $f_{i,j}$ is k **then**
        $c_{i,j} = c_{i,j} + v$
    else
        $d_{i,j} = d_{i,j} + v$
        if $d_{i,j} > c_{i,j}$ **then**
            Send $(f_{i,j}, c_{i,j})$ to controller
            $f_{i,j} = k$, $c_{i,j} = v$, $d_{i,j} = 0$
        else
            if $k \notin BF_{i,j}$ **then**
                Send k to controller
                Insert k to $BF_{i,j}$

procedure UPDATE(k, v)
    for row $i = 1, 2, \ldots, r$ **do**
        $j = h_i(k)$
        UPDATEBUCKET(k, v, i, j)



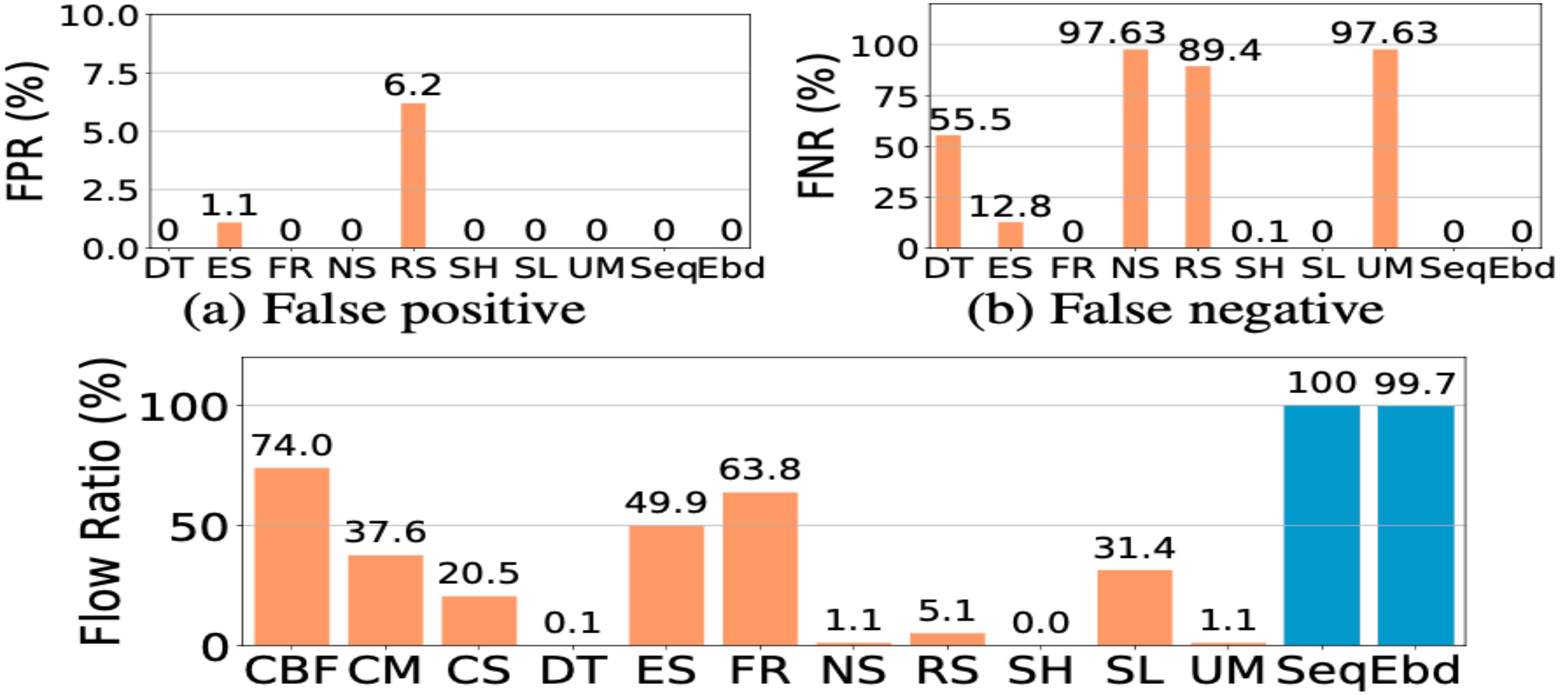
Parameters

- Fractional Sketch: Minimum Amount of Counters Required is $C.Slog_2(\frac{n}{S})$, Where is The Number of Possible Flows and is The Expected Number of Actual Flows
- They Set $S = 100k$ 2-tuple Flows ($n = 2^{64}$) and $C = 0.1$ Which Results in 472 counters and 1888 Total Memory of Fractional Sketch
- Bloom Filter: The False Positive Rate of Bloom Filter is $(0.6185)^{\frac{m}{S}}$ and The Optimal Number of Hash Functions is $\frac{m}{S} ln2$, where m is the length of Bloom Filter
- They Set $m = 9.6S$, Which Results in 120KB of Memory Usage for Bloom Filter

Evaluation Setup

- Implementation Platform: Both Software and Hardware Implementation
- Traces: 2018 CAIDA Traces and Two Data Center Traces
- Flowkey (Flow ID) : 2-tuple (Packet Count)
- Monitoring Intervals: 2 Second Intervals (Around 100k Flows)

Accuracy

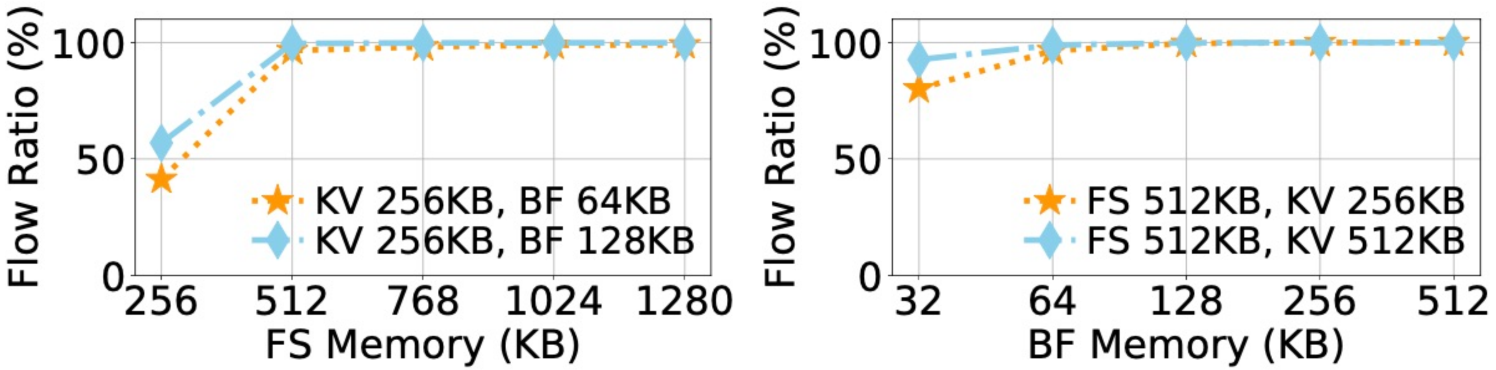


(a) False positive

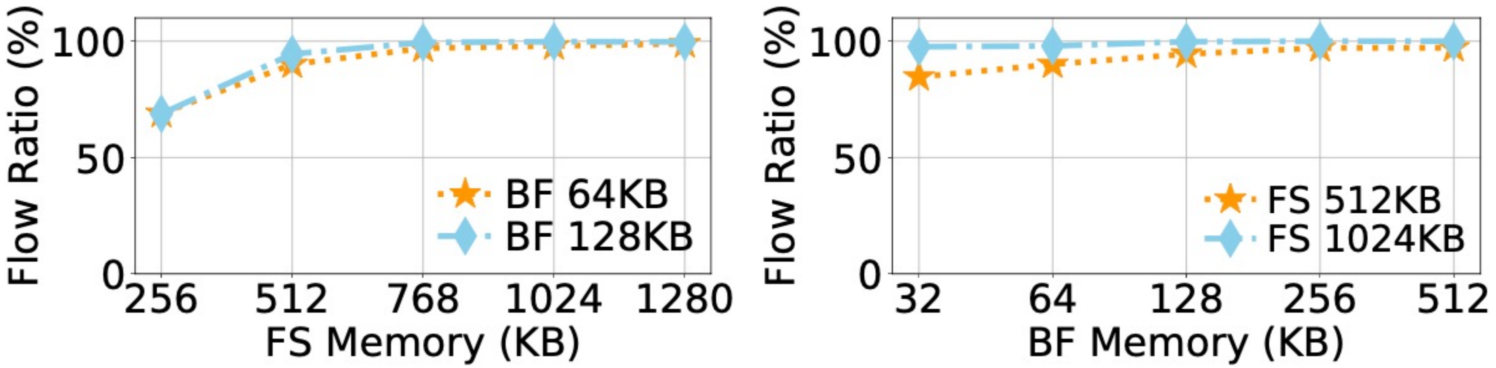
(b) False negative

(c) Fractions of flows with relative errors less than 0.1%

Robustness



(a) SeqSketch w.r.t. FS size (b) SeqSketch w.r.t. BF size

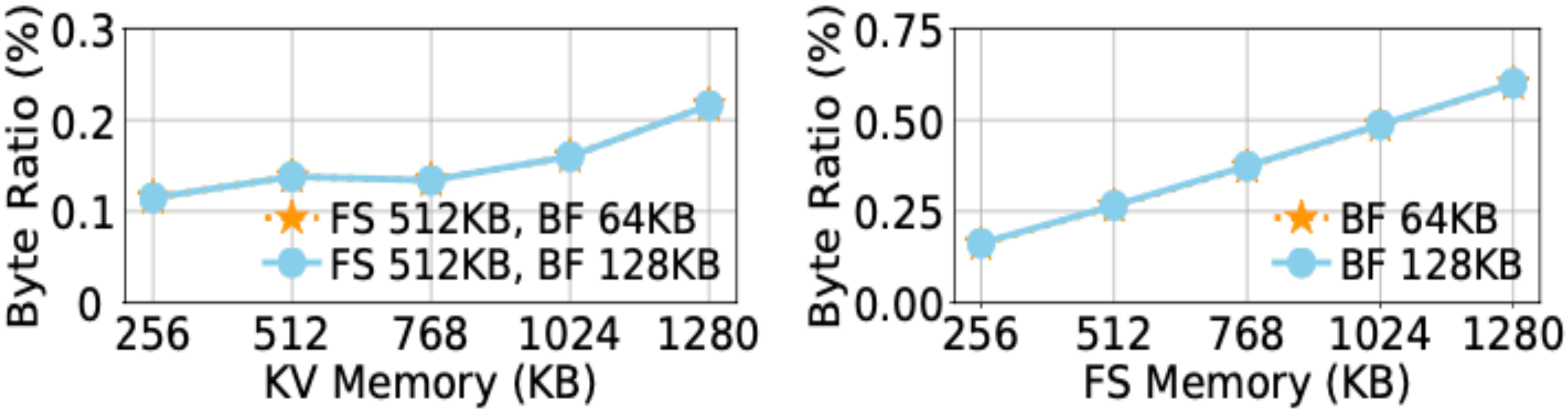


(c) EmbedSketch w.r.t. FS size (d) EmbedSketch w.r.t. BF size

Resource Usage in Tofino

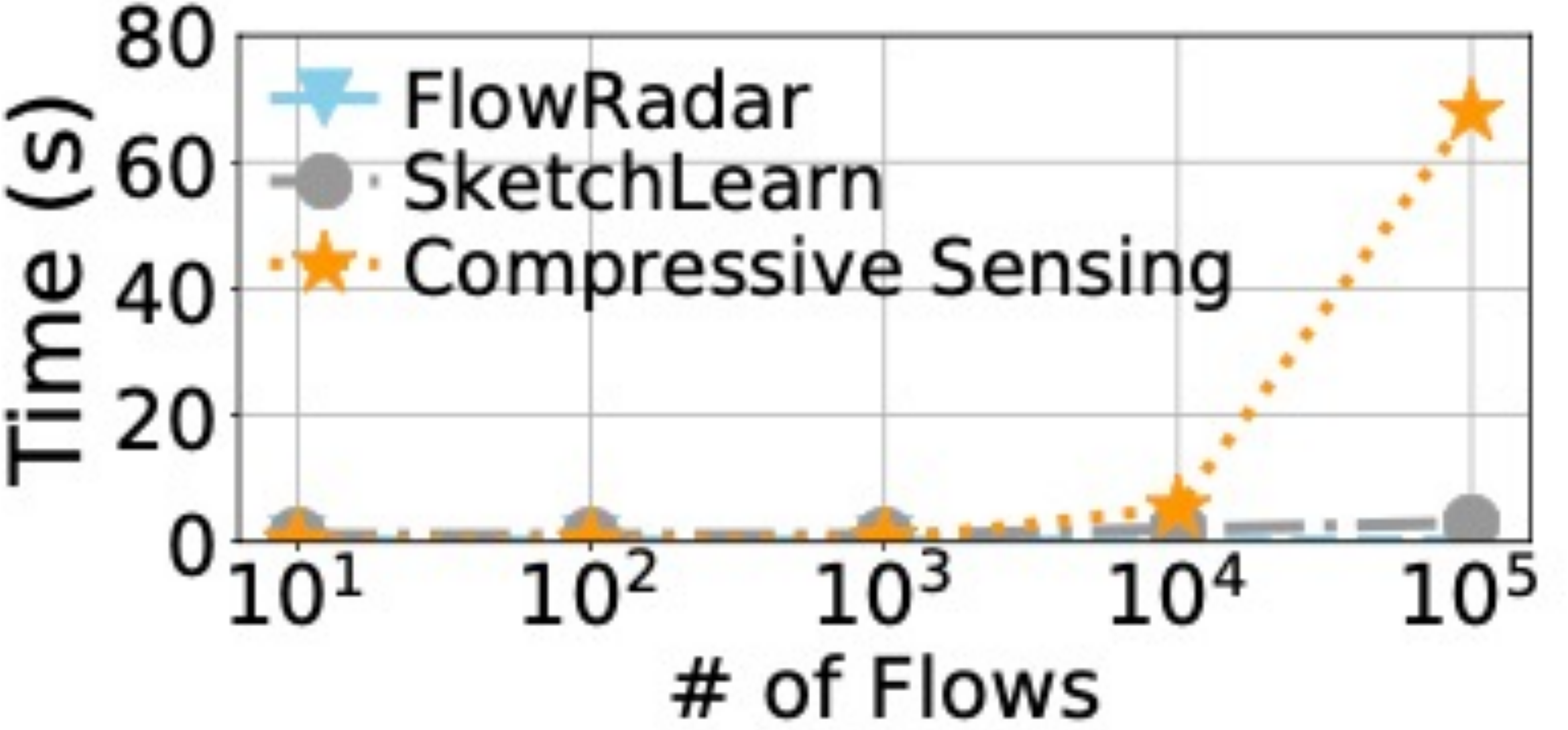
Name	PHV (Bytes)	VLIW	ALU	Stage
ElasticSketch	163 (21.22%)	13 (3.39%)	9 (18.75%)	10 (83.33%)
FlowRadar	134 (21.22%)	11 (2.86%)	15 (31.25%)	10 (83.33%)
SketchLearn	156 (20.31%)	11 (2.86%)	33 (68.75%)	8 (83.33%)
UnivMon	132 (17.19%)	13 (3.39%)	33 (68.75%)	12 (100%)
SeqSketch	151 (19.66%)	12 (3.12%)	13 (27.08%)	8 (66.67%)
EmbedSketch	137 (17.84%)	10 (2.60%)	6 (12.50%)	8 (66.67%)

Bandwidth Usage

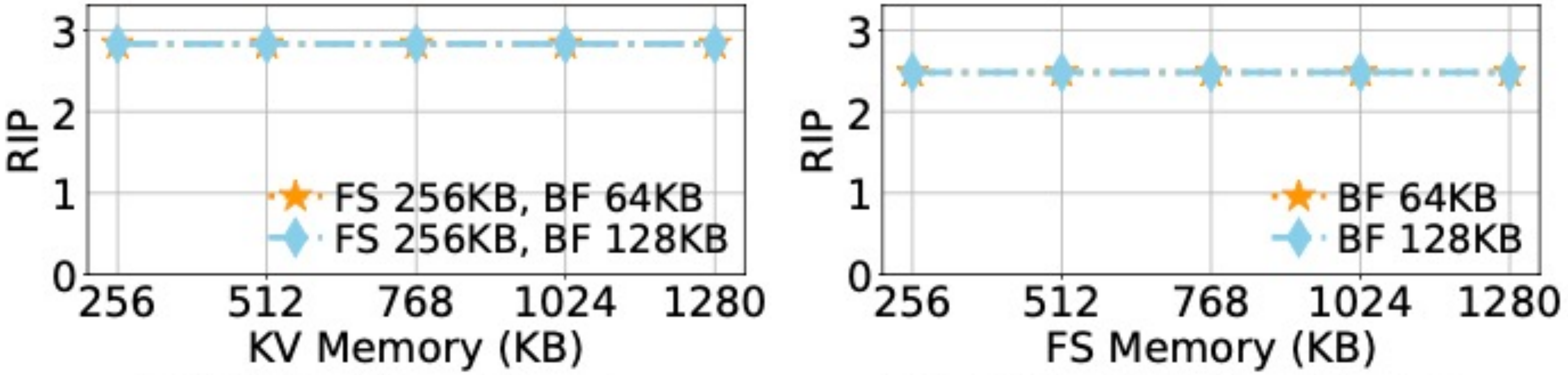


(a) SeqSketch (b) EmbedSketch

Recovery Time



RIP



(a) RIP of SeqSketch

(b) RIP of EmbedSketch