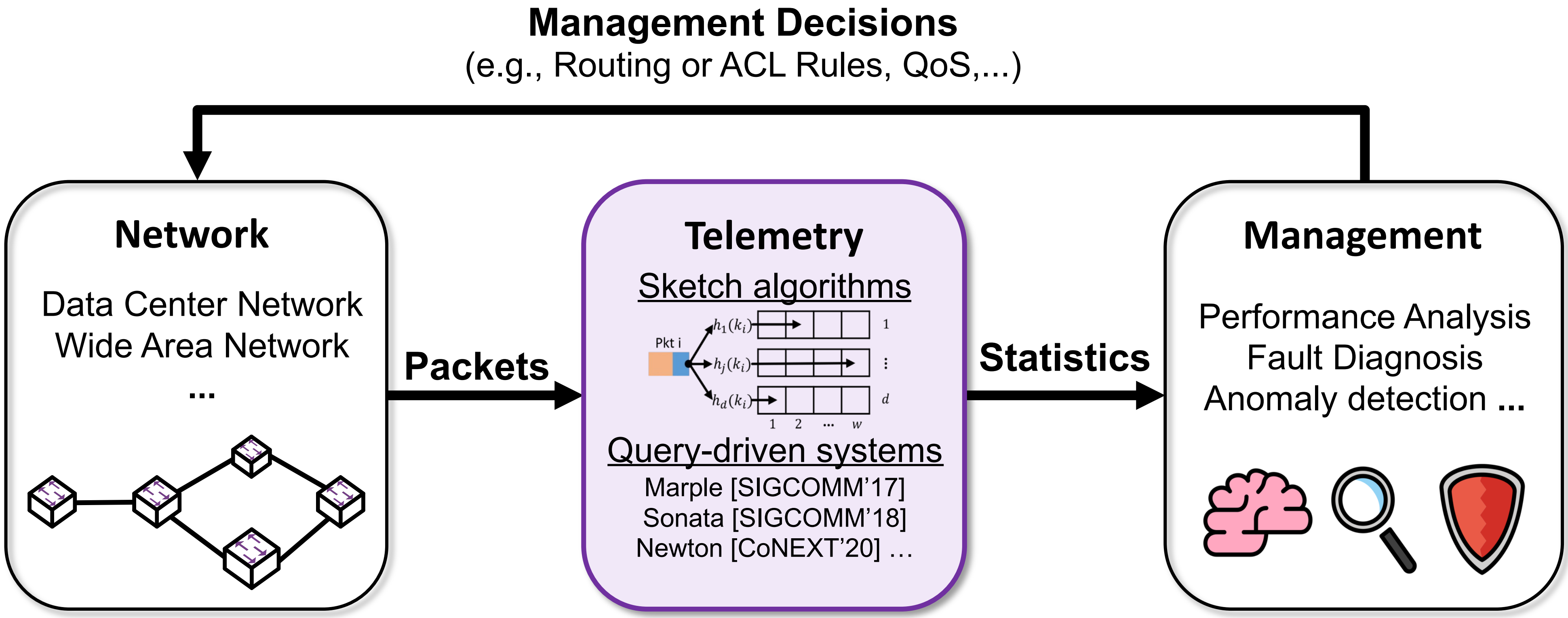
AutoSketch: Automatic Sketch-Oriented Compiler for Query-driven Network Telemetry

Haifeng Sun, Qun Huang, Jinbo Sun, Wei Wang,

Jiaheng Li, Fuliang Li, Yungang Bao, Xin Yao, Gong Zhang

Network Telemetry

- ➢ Network telemetry is significant to network management



Management Decisions
(e.g., Routing or ACL Rules, QoS,...)

Network

Data Center Network
Wide Area Network
...

Packets

Telemetry

Sketch algorithms

Query-driven systems

Marple [SIGCOMM'17]
Sonata [SIGCOMM'18]
Newton [CoNEXT'20] …

Statistics

Management

Performance Analysis
Fault Diagnosis
Anomaly detection ...

Sketch-based Telemetry Algorithms

- Sketches are popular for measuring flow statistics
  - **memory efficiency**
  - **controllable accuracy**

MV-Sketch: A Fast and Compact

for Heavy Flow Detection in Netw

Elastic Sketch: Adaptive and Fast Network-wide Measurements

Tong Yang Jie Jiang Peng Liu

Sliding Sketches: A Framework using Tim

Stream Proces

NitroSketch: Rob

CocoSketch: High-Performance Sketch-based Meas

over Arbitrary Partial Key Query

SpreadSketch: Tov

Detect

[1]Depart

[2]State Key Lab

SketchLearn: Reli

Measurement w

One Sketch to Rule Them All:

Rethinking Network Flow Monitoring with

SketchVisor: Robust N

Software Pa

Qun Huang[1], Xin Jin[2], Patrick P. C. Lee[3], Ru

[1]Huawei Future Network Theory Lab [2]Johns Hop

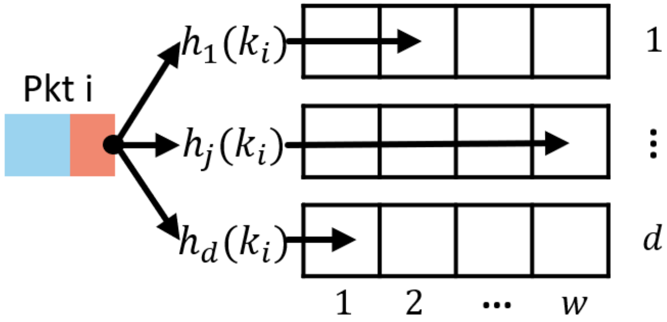
HeteroSketch: Coordinating Network-wide Monitori

in Heterogeneous and Dynamic Networks

Anup Agarwal, *Carnegie Mellon University;* Zaoxing Liu, *Boston University;*
Srinivasan Seshan, *Carnegie Mellon University*

User Burdens of Deploying Sketches

- It is **non-trivial** to deploy sketches in practice

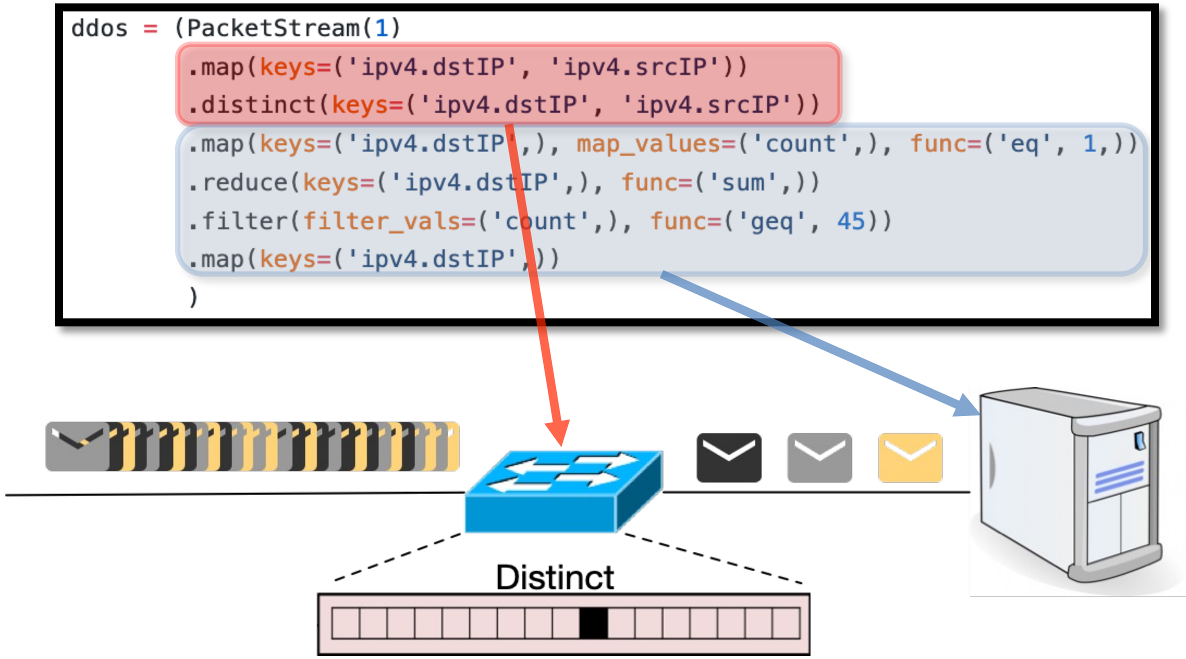


Programmable Switches

- **Burden 1: Sketch selection**
  - Diverse measurement tasks
  - Diverse sketch algorithms
- **Burden 2: Sketch configuration**
  - Theories usually show worst-case results
  - Configuration for worst case not practically efficient
- **Burden 3: Sketch implementation**
  - Unfamiliar programming language (e.g., P4)
  - Various hardware constraints

Query-driven Network Telemetry

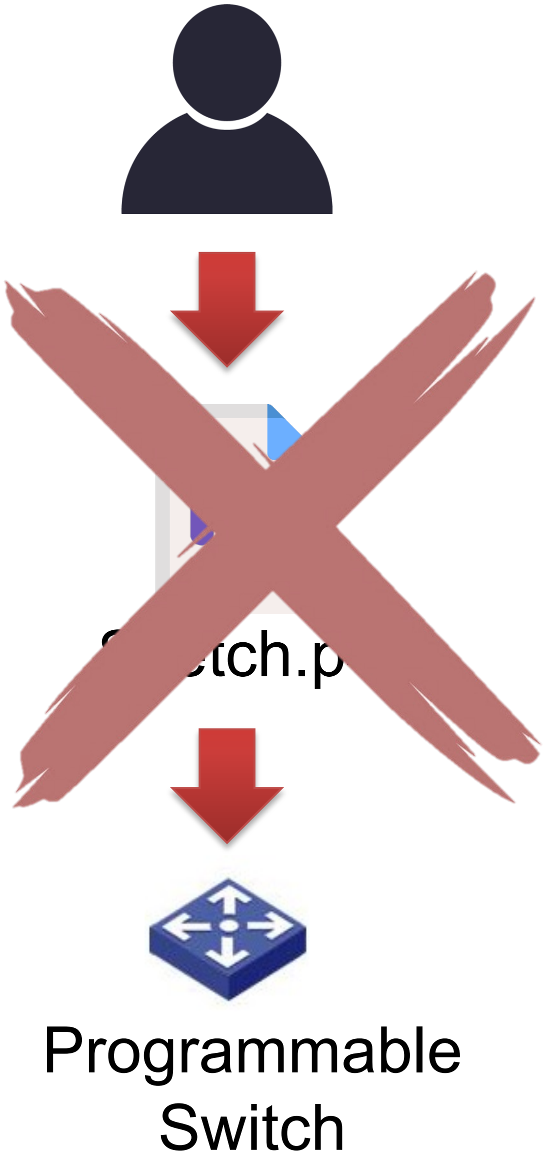
- ➢ Reduce user burdens
- ➢ Expressive telemetry language
  - Focus on query logic
  - Hide the underlying details



```
ddos = (PacketStream(1)
        .map(keys=('ipv4.dstIP', 'ipv4.srcIP'))
        .distinct(keys=('ipv4.dstIP', 'ipv4.srcIP'))
        .map(keys=('ipv4.dstIP',), map_values=('count',), func=('eq', 1,))
        .reduce(keys=('ipv4.dstIP',), func=('sum',))
        .filter(filter_vals=('count',), func=('geq', 45))
        .map(keys=('ipv4.dstIP',))
        )
```

Sonata [SIGCOMM’18]

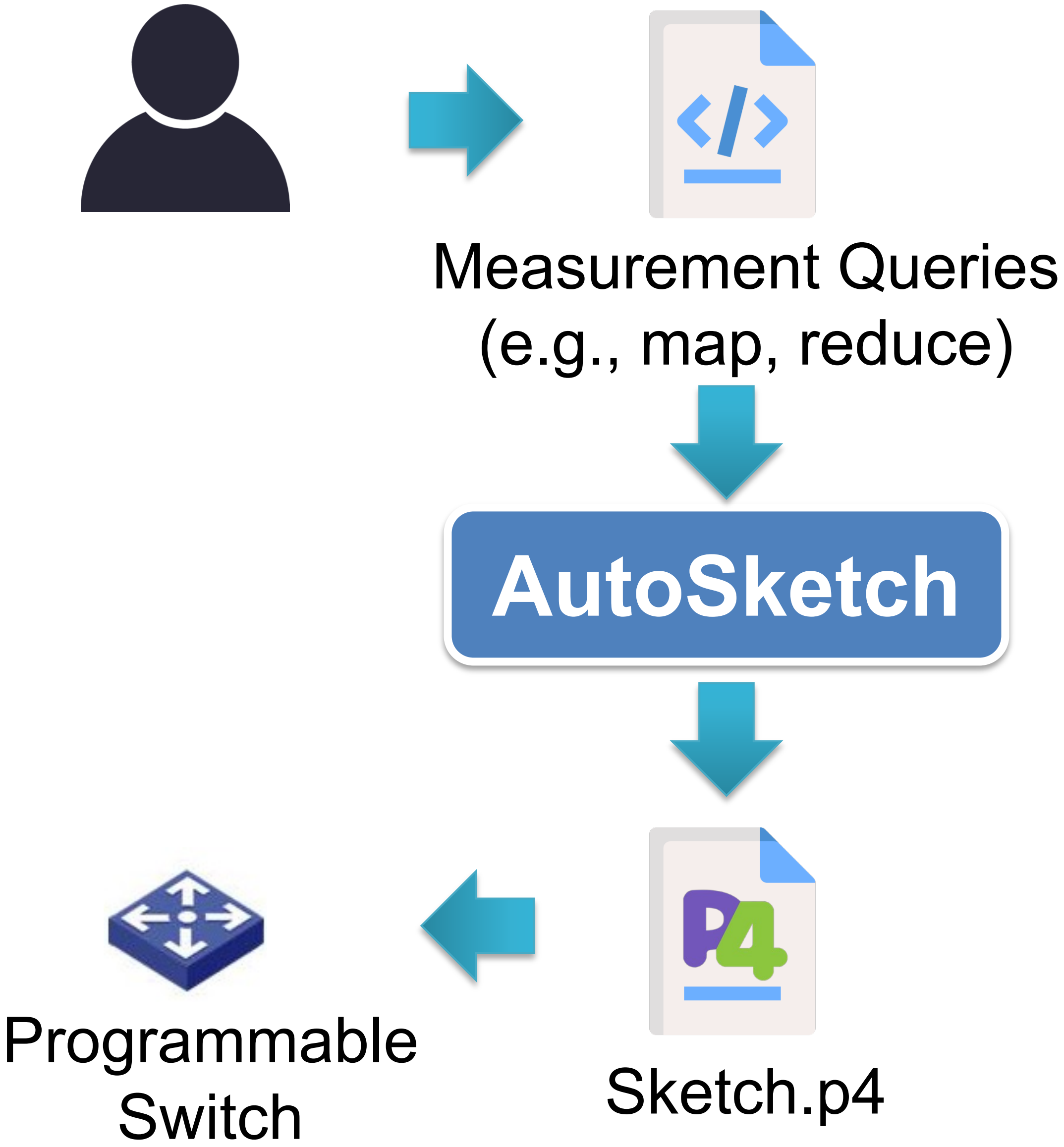
Our Work: AutoSketch



User Burdens

- Sketch selection
- Sketch configuration
- Sketch implementation

Our Work: AutoSketch



Measurement Queries (e.g., map, reduce)

AutoSketch

Sketch.p4

Programmable Switch

Combine the strengths of both query-driven telemetry and sketches

- Strong Expressiveness
- Resource Efficiency
- Controllable Accuracy

AutoSketch in a nutshell

- **Combine the strengths of both sketches and query-driven telemetry**

Challenges

1. How to perceive and control the errors incurred by sketches?

2. How to map diverse telemetry queries into appropriate sketches?

3. How to configure the mapped sketch algorithms?

AutoSketch in a nutshell

- ➢ **Combine the strengths of both sketches and query-driven telemetry**

Challenges

1. How to perceive and control the errors incurred by sketches?

2. How to map diverse telemetry queries into appropriate sketches?

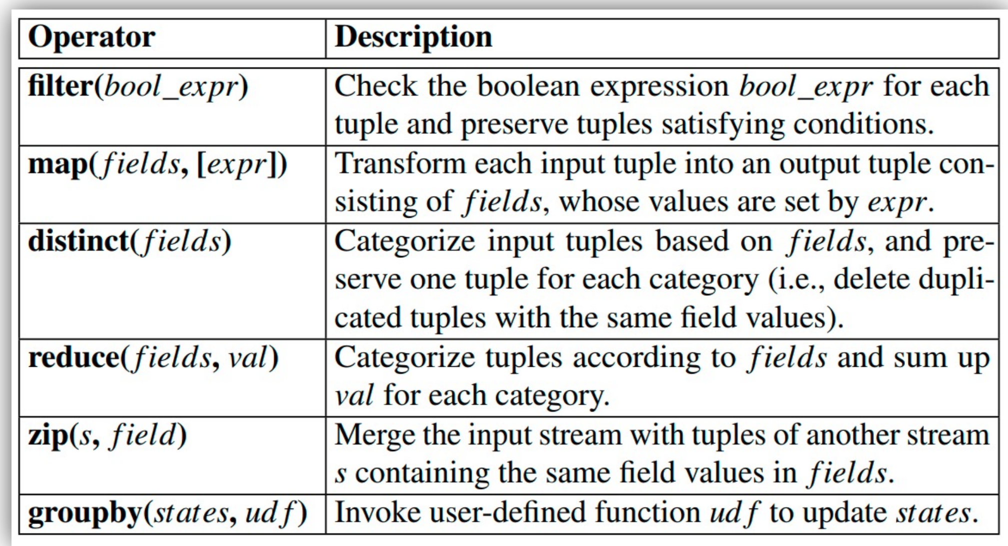
3. How to configure the mapped sketch algorithms?

AutoSketch Outline

1. Data stream abstraction with Accuracy Intent

AutoSketch Interface

- **Data stream abstraction** (e.g., map, reduce, filter)
  - strong expressiveness to cover numerous queries
  - widely adopted by query-driven telemetry
    - Marple [SIGCOMM’17], Sonata [SIGCOMM’18], BeauCoup [SIGCOMM’20], …

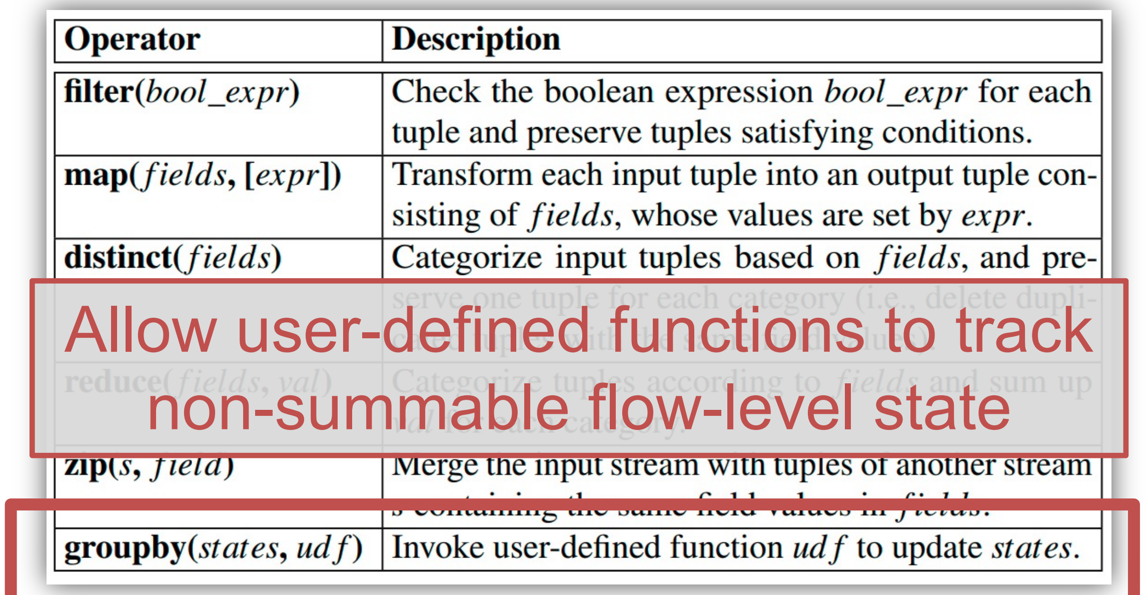


Operator	Description
filter(*bool_expr*)	Check the boolean expression *bool_expr* for each tuple and preserve tuples satisfying conditions.
map(*fields*, [*expr*])	Transform each input tuple into an output tuple consisting of *fields*, whose values are set by *expr*.
distinct(*fields*)	Categorize input tuples based on *fields*, and preserve one tuple for each category (i.e., delete duplicated tuples with the same field values).
reduce(*fields*, *val*)	Categorize tuples according to *fields* and sum up *val* for each category.
zip(*s*, *field*)	Merge the input stream with tuples of another stream *s* containing the same field values in *fields*.
groupby(*states*, *udf*)	Invoke user-defined function *udf* to update *states*.

```
ddos_attack = PacketStream(qid=1)
  .map(pkt -> (ipv4.dstIP, ipv4.srcIP))
  .distinct()
  .map((ipv4.dstIP, ipv4.srcIP) -> (ipv4.dstIP, 1))
  .reduce(keys=(ipv4.dstIP,), func=sum)
  .filter((ipv4.dstIP, count) -> count >= Threshold)
  .map((ipv4.dstIP, count) -> (ipv4.dstIP))
  .distinct()
```

AutoSketch Interface

- **Data stream abstraction** (e.g., map, reduce, filter)
  - strong expressiveness to cover numerous queries
  - widely adopted by query-driven telemetry
    - Marple [SIGCOMM’17], Sonata [SIGCOMM’18], BeauCoup [SIGCOMM’20], …



Operator	Description
filter(*bool_expr*)	Check the boolean expression *bool_expr* for each tuple and preserve tuples satisfying conditions.
map(*fields*, [*expr*])	Transform each input tuple into an output tuple consisting of *fields*, whose values are set by *expr*.
distinct(*fields*)	Categorize input tuples based on *fields*, and preserve one tuple for each category (i.e., delete duplicate tuples with the same field values).
reduce(*fields*, *val*)	Categorize tuples according to *fields* and sum up *val* for each category.
zip(*s*, *field*)	Merge the input stream with tuples of another stream *s* containing the same field values in *fields*.
groupby(*states*, *udf*)	Invoke user-defined function *udf* to update *states*.

Allow user-defined functions to track non-summable flow-level state

```
def nonmt(tcp.seq):
  if maxseq < tcp.seq or maxseq == 0:
    maxseq = tcp.seq
  else:
    nm_count += 1

tcp_nm = PacketStream(qid=2)
  .filter(ipv4.protocol == TCP)
  .groupby({5tuple: (maxseq, nm_count)}, nonmt)
```

AutoSketch Interface

- **Data stream abstraction** (e.g., map, reduce, filter)
  - strong expressiveness to cover numerous queries
  - widely adopted by query-driven telemetry
    - Marple [SIGCOMM’17], Sonata [SIGCOMM’18], BeauCoup [SIGCOMM’20], …

- **Accuracy Intent** allows users to specify an acceptable error bound
  - recall
  - precision
  - average relative error
  - confidence

```
ddos_attack = PacketStream(qid=1, recall_min=0.95,
        precision_min=0.95, confidence=0.99)
  .map(pkt -> (ipv4.dstIP, ipv4.srcIP))
  .distinct()
  .map((ipv4.dstIP, ipv4.srcIP) -> (ipv4.dstIP, 1))
  .reduce(keys=(ipv4.dstIP,), func=sum)
  .filter((ipv4.dstIP, count) -> count >= Threshold)
  .map((ipv4.dstIP, count) -> (ipv4.dstIP))
  .distinct()
```

AutoSketch Interface

- **Data stream abstraction** (e.g., map, reduce, filter)
  - strong expressiveness to cover numerous queries
  - widely adopted by query-driven telemetry
    - Marple [SIGCOMM’17], Sonata [SIGCOMM’18], BeauCoup [SIGCOMM’20], …

- **Accuracy Intent** allows users to specify an acceptable error bound
  - recall
  - precision
  - average relative error
  - confidence

```
ddos_attack = PacketStream(qid=1, recall_min=0.95,
          precision_min=0.95, confidence=0.99)
  .map(pkt -> (ipv4.dstIP, ipv4.srcIP))
  .distinct()
  .map((ipv4.dstIP, ipv4.srcIP) -> (ipv4.dstIP, 1))
  .reduce(keys=(ipv4.dstIP,), func=sum)
  .filter((ipv4.dstIP, count) -> count >= Threshold)
  .map((ipv4.dstIP, count) -> (ipv4.dstIP))
  .distinct()
```

The accuracy intent guides the sketch selection and configuration

AutoSketch in a nutshell

- **Combine the strengths of both sketches and query-driven telemetry**

Challenges

1. How to perceive and control the errors incurred by sketches?

2. How to map diverse telemetry queries into appropriate sketches?

3. How to configure the mapped sketch algorithms?

AutoSketch Outline

1. Data stream abstraction with Accuracy Intent

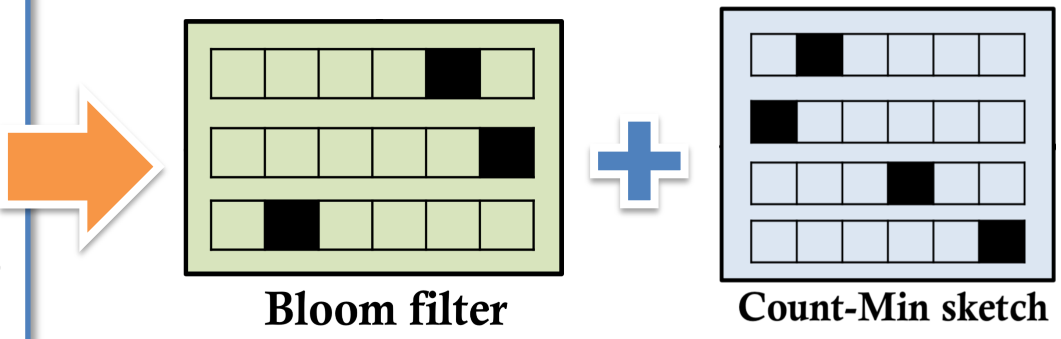
2. Operator-level sketch mapping

Operator-level Sketch Mapping

➢ Operator-level sketch mapping

- Map each **stateful operator** into one sketch instance
  - built-in operators (i.e., **`distinct`**, **`reduce`**)
  - user-defined operators (i.e., **`groupby`**)

```
ddos_attack = PacketStream(qid=1)
   .map(pkt -> (ipv4.dstIP, ipv4.srcIP))
   .distinct()
   .map((ipv4.dstIP, ipv4.srcIP) -> (ipv4.dstIP, 1))
   .reduce(keys=(ipv4.dstIP,), func=sum)
   .filter((ipv4.dstIP, count) -> count >= Threshold)
   .map((ipv4.dstIP, count) -> (ipv4.dstIP))
   .distinct()
```



Bloom filter

Count-Min sketch

Operator-level Sketch Mapping

Operator-level sketch mapping

- Map each **stateful operator** into one sketch instance
  - built-in operators (i.e., **`distinct`**, **`reduce`**)
  - user-defined operators (i.e., **`groupby`**)

Better generality and flexibility

Query-level sketch mapping

- Map the entire query into one **universal** sketch algorithm
  - Limited query tasks
  - Cannot cover user-defined operators
  - Hard for fine-grained sketch tuning

Operator-level Sketch Mapping

- **Operator-level sketch mapping**
  - Map each **stateful operator** into a sketch instance
    - built-in operators (i.e., `distinct`, `reduce`)
    - user-defined operators (i.e., `groupby`)
  - **Better generality and flexibility** than **query-level** mapping

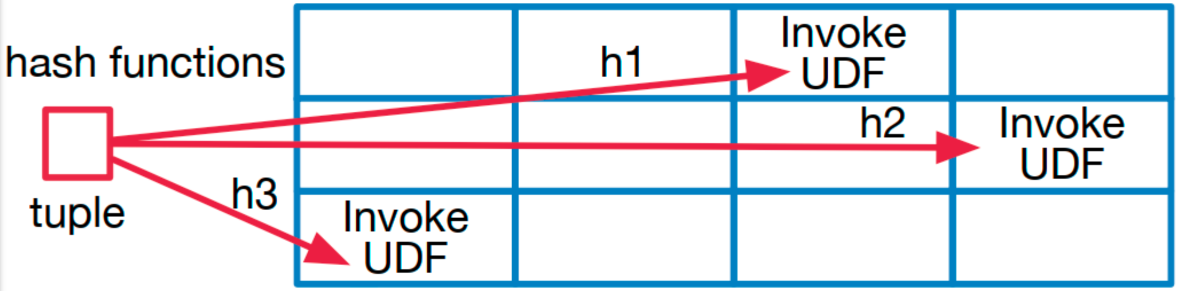
- **Key-value separation storage**
  - Key: Use the flowkey buffer mechanism to sequentially store new flowkeys (OmniWindow [SIGCOMM'23])
  - Value: Use different sketch algorithms to maintain various flow states

Sketch Mapping for User-defined Operators

```
def nonmt(tcp.seq):
  if maxseq < tcp.seq or maxseq == 0:
    maxseq = tcp.seq
  else:
    nm_count += 1

tcp_nm = PacketStream(qid=2)
  .filter(ipv4.protocol == TCP)
  .groupby({5tuple: (maxseq, nm_count)}, nonmt)
```

- ➢ Sketch-like structure

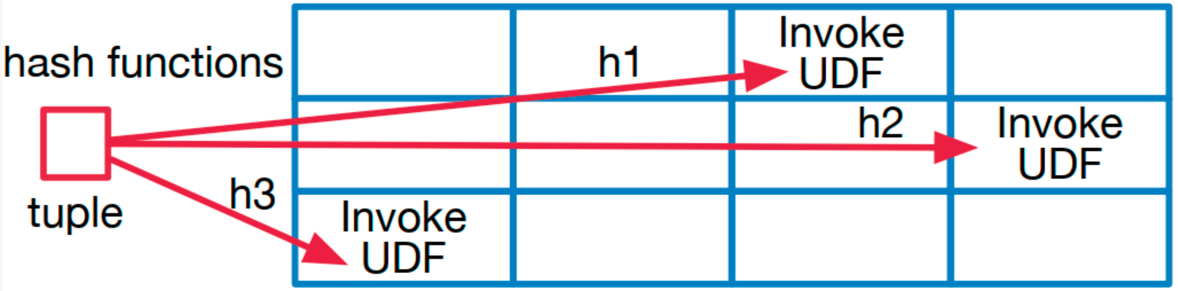


Sketch Mapping for User-defined Operators

```
def nonmt(tcp.seq):
  if maxseq < tcp.seq or maxseq == 0:
    maxseq = tcp.seq
  else:
    nm_count += 1

tcp_nm = PacketStream(qid=2)
  .filter(ipv4.protocol == TCP)
  .groupby({5tuple: (maxseq, nm_count)}, nonmt)
```

- ➢ Sketch-like structure



A groupby operator may contain **multiple** states

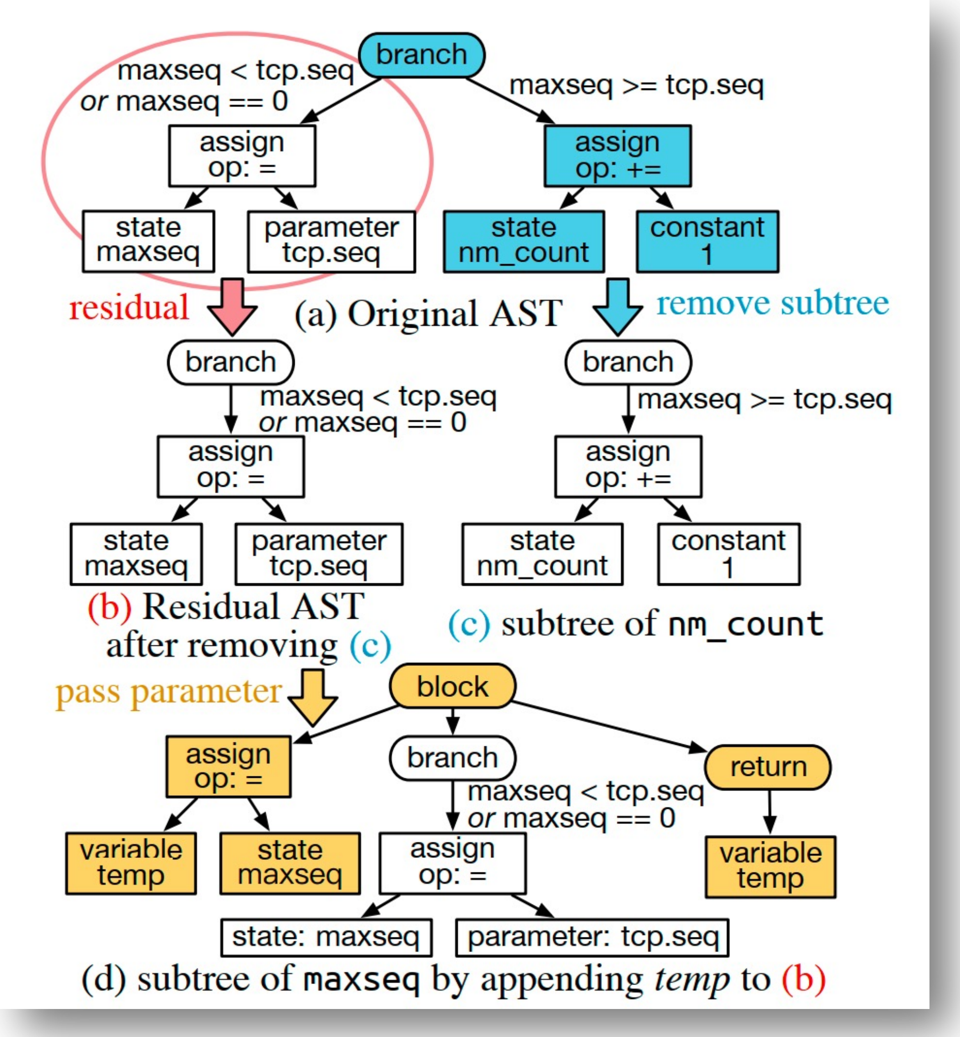
- Multi-state updates may exceed hardware capabilities
- Hard to tune memory resources for different states

Sketch Mapping for User-defined Operators

- AST-based operator decomposition
  - One state, One `groupby`
  - Sketch-like structure for decomposed states

```
def nonmt(tcp.seq):
  if maxseq < tcp.seq or maxseq == 0:
    maxseq = tcp.seq
  else:
    nm_count += 1

tcp_nm = PacketStream(qid=2)
  .filter(ipv4.protocol == TCP)
  .groupby({5tuple: (maxseq, nm_count)}, nonmt)
```



(a) Original AST

(b) Residual AST after removing (c)

(c) subtree of `nm_count`

(d) subtree of `maxseq` by appending *temp* to (b)

More details in the paper

Sketch Mapping for User-defined Operators

- ➢ AST-based operator decomposition
  - One state, One `groupby`
  - Sketch-like structure for decomposed states

```
def nonmt(tcp.seq):
  if maxseq < tcp.seq or maxseq == 0:
    maxseq = tcp.seq
  else:
    nm_count += 1

tcp_nm = PacketStream(qid=2)
  .filter(ipv4.protocol == TCP)
  .groupby({5tuple: (maxseq, nm_count)}, nonmt)
```

```
def nonmt_cf(tcp.seq):
  temp = maxseq
  if maxseq < tcp.seq or maxseq == 0:
    maxseq = tcp.seq
  return temp
def nonmt_uf(tcp.seq, temp):
  if temp >= tcp.seq:
    nm_count += 1

tcp_nm = PacketStream(qid=2)
  .filter(ipv4.protocol == TCP)
  .groupby({5tuple: maxseq}, nonmt_cf)
  .groupby({5tuple: nm_count}, nonmt_uf)
```

Sketch mapping for Built-in operators

- Fixed function with well-known studied sketch algorithms

```
ddos_attack = PacketStream(qid=1)
  .map(pkt -> (ipv4.dstIP, ipv4.srcIP))
  .distinct() → Bloom Filter? Counting Bloom Filter? …?
  .map((ipv4.dstIP, ipv4.srcIP) -> (ipv4.dstIP, 1))
  .reduce(keys=(ipv4.dstIP,), func=sum) → Count-Min Sketch? Count Sketch? …?
  .filter((ipv4.dstIP, count) -> count >= Threshold)
  .map((ipv4.dstIP, count) -> (ipv4.dstIP))
  .distinct()
```

- **Sampling-based sketch selection**
  - Initial phase of benchmark-based sketch configuration (details below)

AutoSketch in a nutshell

- **Combine the strengths of both sketches and query-driven telemetry**

Challenges

1. How to perceive and control the errors incurred by sketches?

2. How to map diverse telemetry queries into appropriate sketches?

3. How to configure the mapped sketch algorithms?

AutoSketch Outline

1. Data stream abstraction with Accuracy Intent

2. Operator-level sketch mapping

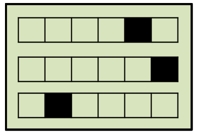
3. Benchmark-based sketch configuration

Benchmark-based Sketch Configuration

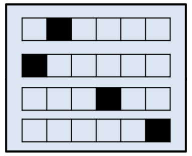
Accuracy Intent

```
ddos = PacketStream(qid=1, recall_min=0.95,
         precision_min=0.95, confidence=0.95)
```

Sketch error bound



Bloom filter



Count-Min sketch

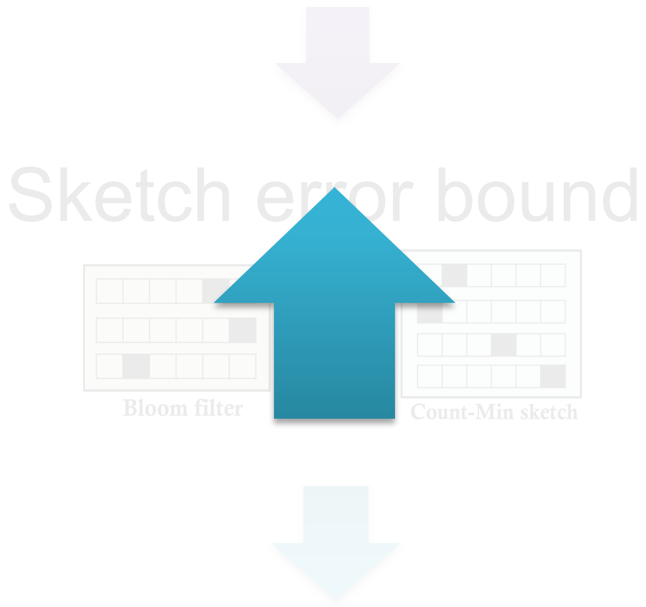
Sketch Type & Config

- Hard to integrate the errors of multiple heterogeneous sketches to form query-level accuracy
- Existing sketch algorithms typically address worst-case scenarios

Benchmark-based Sketch Configuration

Accuracy Intent

```
ddos = PacketStream(qid=1, recall_min=0.95,
        precision_min=0.95, confidence=0.95)
```



Sketch Type & Config

- **Benchmark-based searching**
  - Enumerate all possible configurations
  - Synthetic or real workloads
  - **Efficient search algorithm**
    - LHS-based initialization, which determines the sketch type of built-in operators at the same time
    - Hardware-aware configuration generation and pruning
    - Return the sketch configuration that **satisfies the accuracy intent while incurring minimal resource usage**

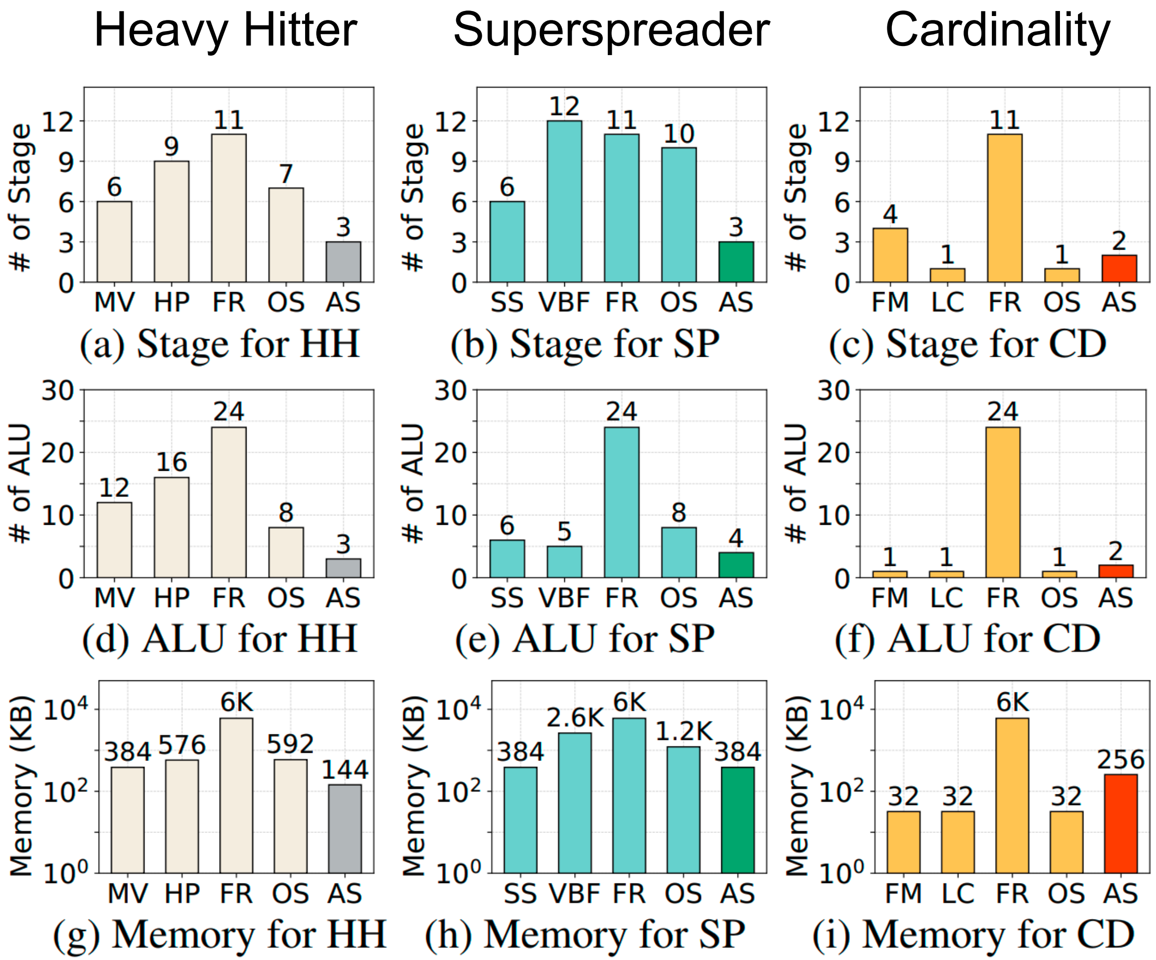
More details in the paper

Evaluation

- AutoSketch backend: Tofino Switch
- **11** telemetry queries
- Compare with 2 state-of-the-art query-driven systems
- Compare with **8** classical sketch algorithms
- Two accuracy intents

Accuracy intent	precision_min	recall_min	ARE_max
AS-1	95%	95%	1%
AS-2	99%	55%	3%

Switch Resource Overhead



(a) Stage for HH (b) Stage for SP (c) Stage for CD

(d) ALU for HH (e) ALU for SP (f) ALU for CD

(g) Memory for HH (h) Memory for SP (i) Memory for CD

- ➢ AutoSketch consumes **much lower switch resource usage** than classical sketch algorithms
- ➢ AutoSketch needs no efforts to tune parameters

Two specified sketches for each query

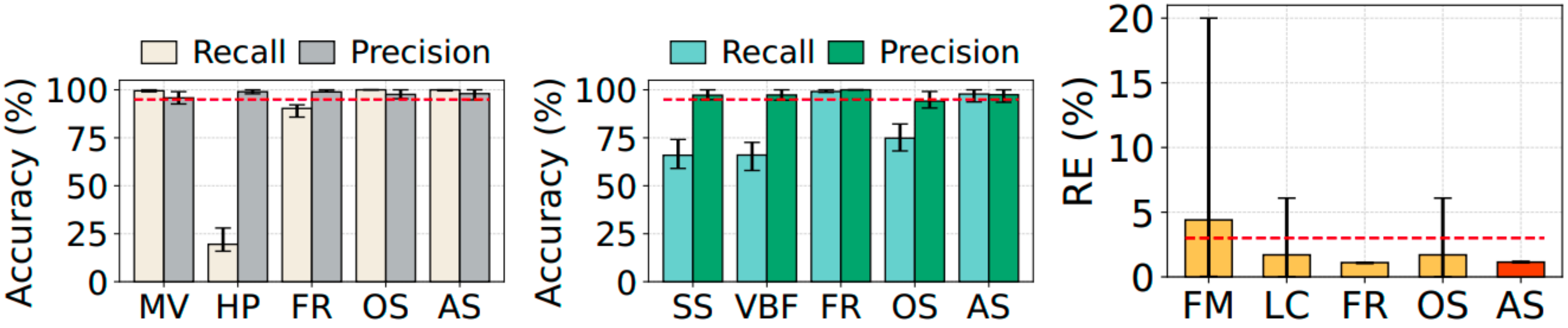
- Heavy Hitter: **MV**-Sketch [INFOCOM’19] **H**ash**P**ipe [SOSR’17]
- Superspreader: **S**pread**S**ketch [INFOCOM’20] **V**ector**BF** [TIFS’16]
- Cardinality: **FM**-Sketch [JCSS’85] **L**inear **C**ounting [TODS’90]

Two universal sketches for all queries

- **FR**: FlowRadar [NSDI’16]
- **OS**: OpenSketch [NSDI’13]

Controllable Accuracy

➢ AutoSketch meets the accuracy intent for all the telemetry queries



(a) Heavy Hitter (b) Superspreader (c) Cardinality

More Results

- Compare with Query-driven telemetry systems
  - Marple [SIGCOMM’17], two variants of Sonata [SIGCOMM’18]
  - Switch resource overhead
  - Bandwidth overhead
  - Query Accuracy
- Parameter tuning
- Seaching cost
- Efficiency of searching results

Future Work

- Multi-query and distributed deployment
- More backends
  - DPU / SmartNIC
  - DPDK
- Support more underlying sketches
  - new sketch-like structure for groupby
  - adding new sketch candidates for existing operators
  - introducing new operators
- Integrate emerging sketch optimization techniques
  - SketchLib [NSDI'22], FlyMon [SIGCOMM'22], Sketchovsky [NSDI'23]
  - BitSense [SIGCOMM'23], OmniWindow [SIGCOMM'23]

Conclusion

- ➢ AutoSketch combines the strengths of
  - Strong expressiveness
  - Resource efficiency
  - Controllable accuracy
- ➢ Accuracy Intent
- ➢ Operator-level sketch mapping
- ➢ Benchmark-based sketch configuration
- ➢ Source Code Available: https://github.com/N2-Sys/AutoSketch

Thank You!