# SONATA:
# Query-Driven Network Telemetry

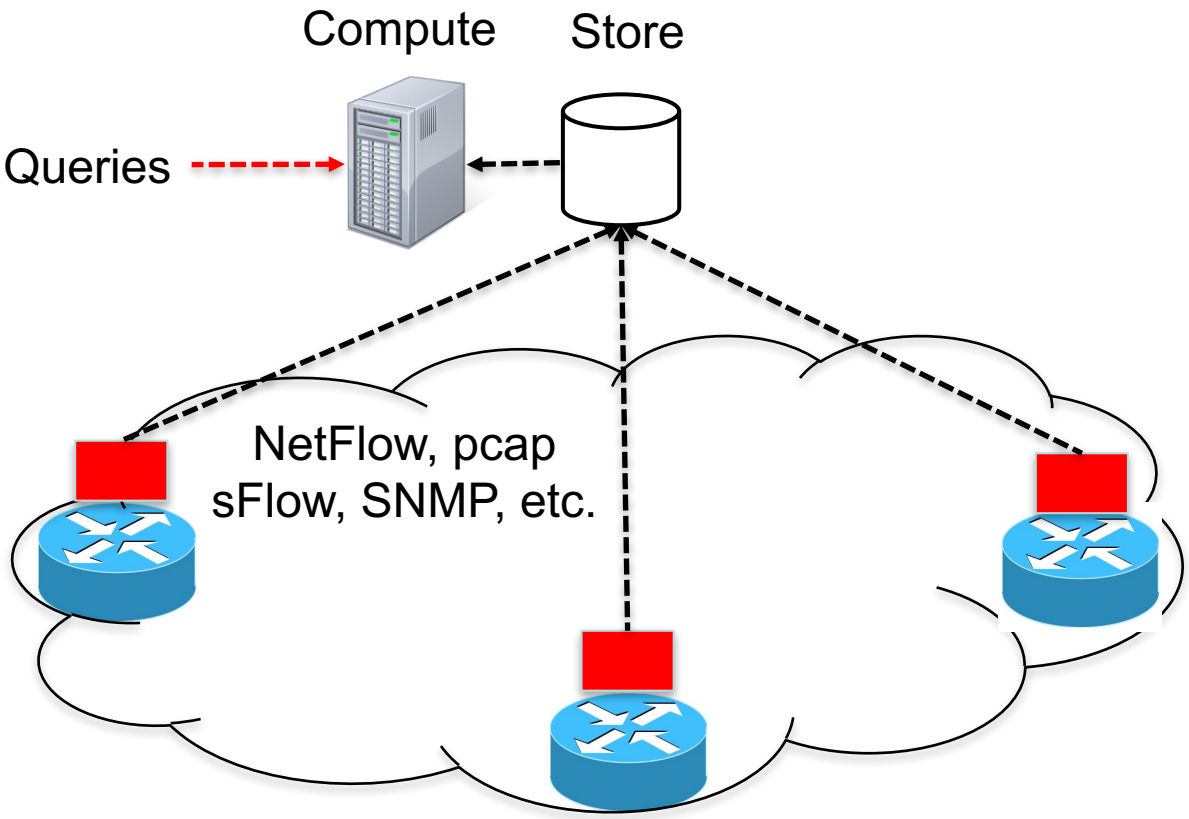

**Arpit Gupta**

**Princeton University**

*Rob Harrison, Ankita Pawar, Rüdiger Birkner,*

*Marco Canini, Nick Feamster, Jennifer Rexford, Walter Willinger*

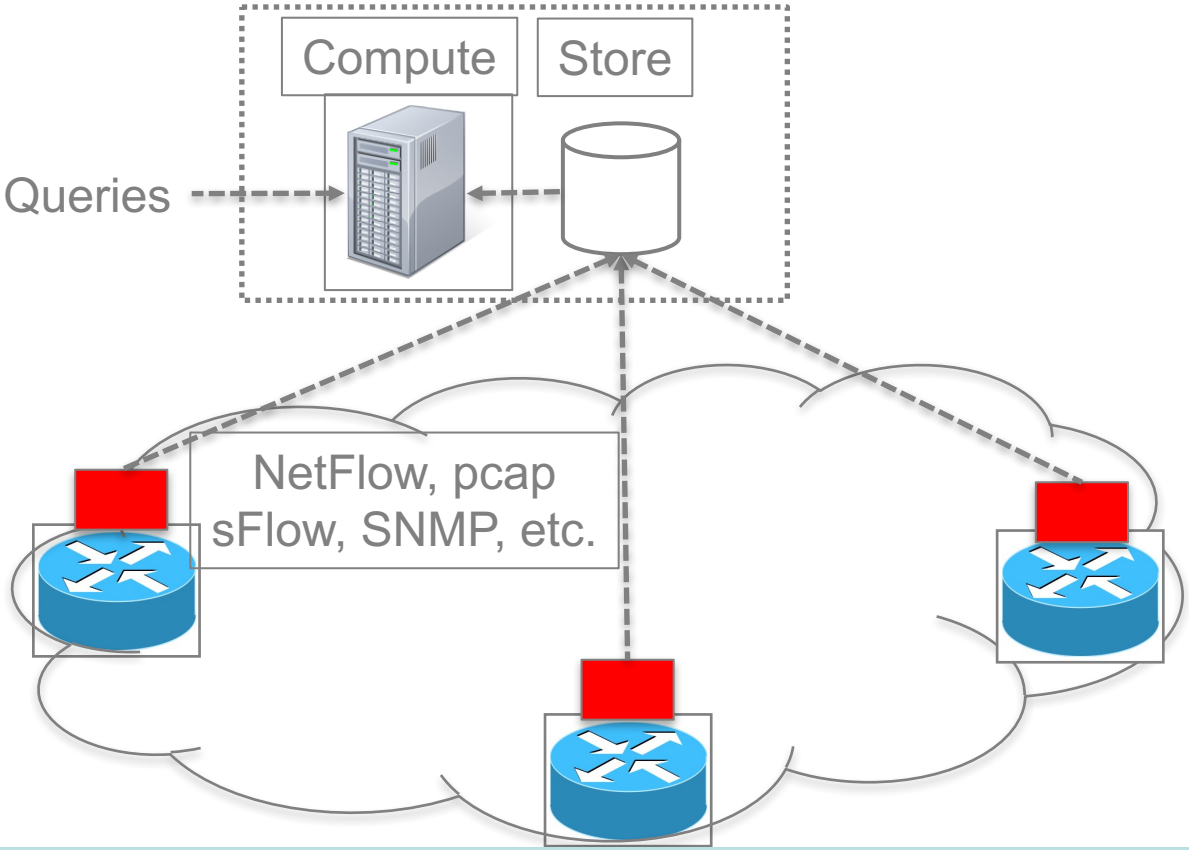
# Conventional Network Telemetry

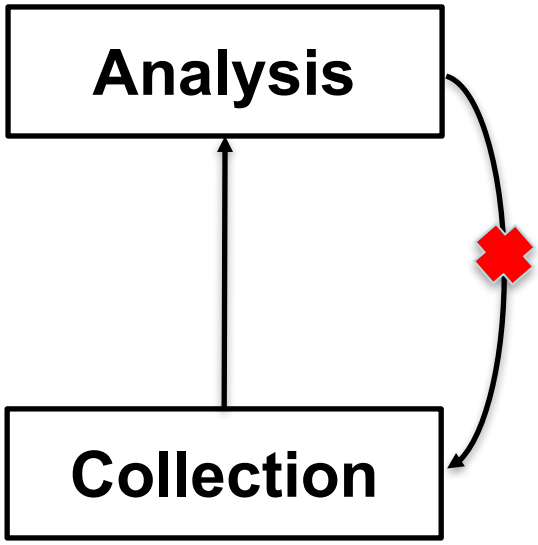



**Analysis**

**Collection**

# Conventional Network Telemetry




Collection is **not** driven by Analysis

# Problems with Status Quo

- ***Expressibility***
  - Configure collection & analysis stages separately
  - Static (and often coarse) data collection
  - Brittle analysis setup---specific to collection tools

# Problems with Status Quo

- *Expressibility*
  - Configure collection & analysis stages separately
  - Static (and often coarse) data collection
  - Brittle analysis setup---specific to collection tools

- ***Scalability***

  As Traffic Volume or # Monitoring Queries increases
  - Hard to answer queries in real-time

Hard to **express** & **scale** queries for network telemetry tasks!

# SONATA: Query-Driven Telemetry

- ***Uniform Programming Abstraction***

  Express queries as dataflow operations over pkt. tuples

- ***Query-Driven Data Reduction***

  Execute subset of dataflow operations in data plane

- ***Coordinated Data Collection & Analysis***

  Select query plans that make best use of available resources

# Uniform Programming Abstraction

- ***Extensible Packet-tuple Abstraction***

  Queries operate over all packet tuples, at every location in the network

- ***Expressive Dataflow Operators***
  - Most telemetry applications require
    - collecting aggregate statistics over subset of traffic
    - joining results of one analysis with the other
  - Easy to express them as declarative queries composed of dataflow operators

# Example Query

## Detecting DNS Reflection Attack

Detect hosts for which # of unique source IPs sending DNS response messages exceeds threshold (Th)

```
victimIPs = pktStream(W)
            .filter(p => p.srcPort == 53)
            .map(p => (p.dstIP, p.srcIP))
            .distinct()
            .map((dstIP, srcIP) => (dstIP, 1))
```

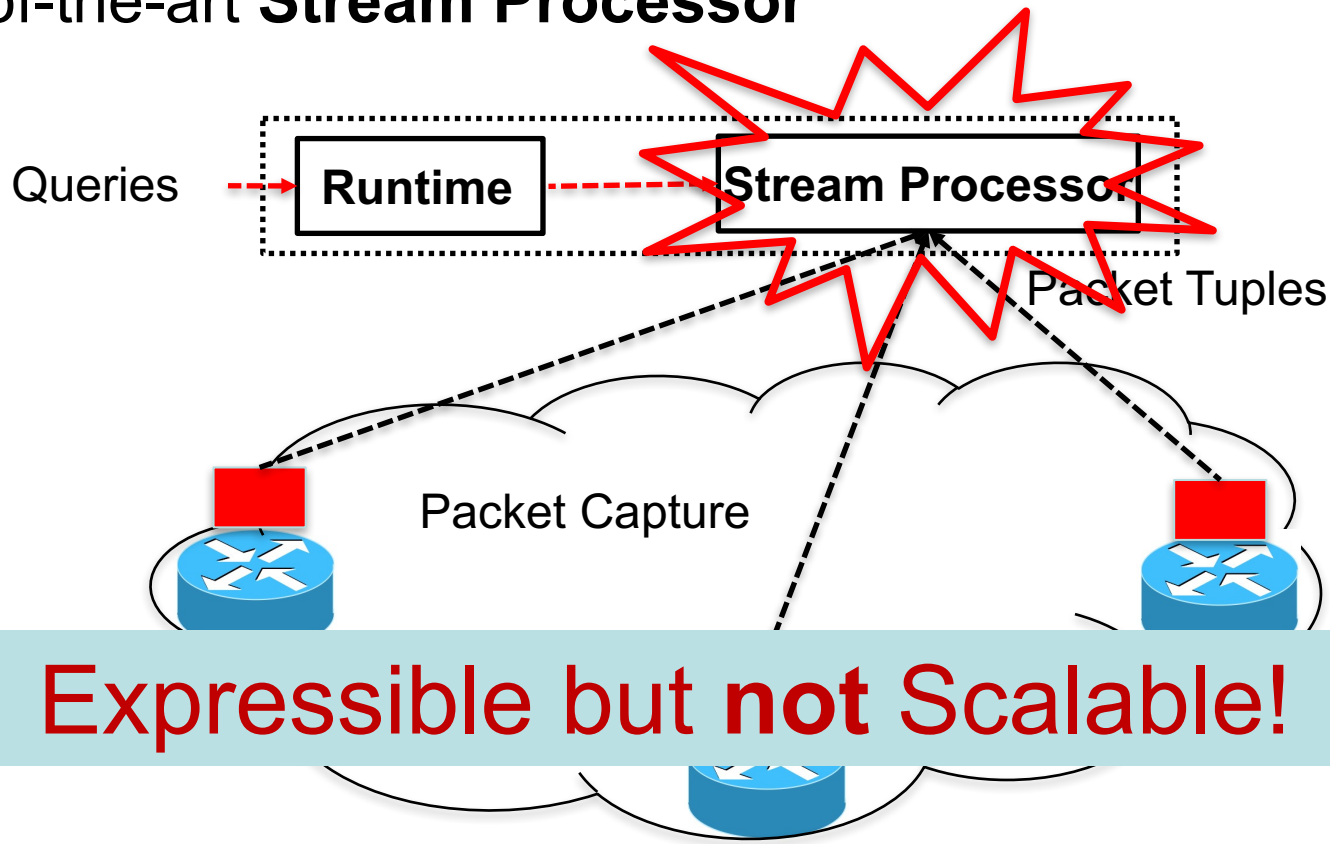
Express queries without worrying about ***where*** and ***how*** they get executed

# Changing Status Quo

- ***Expressibility***
  - Express dataflow queries over packet tuples
  - Not tied to low-level (3rd party/platform-specific) APIs
  - Trivial to add new queries and change collection tools

# Query Execution

Process all (or subset of) captured packet tuples using state-of-the-art **Stream Processor**




Expressible but **not** Scalable!

# PISA Targets for Data Reduction

- ***Programmable parsing***

  Allow new query-specific header fields for parsing

- ***State in packets & registers***

  Support simple stateful computations

- ***Customizable hash functions***

  Support hash functions over flexible set of fields

- ***Flexible match/action table pipelines***

  Support match/action tables with prog. actions

# Compiling Dataflow Operators

- ***Map, Filter & Sample***

  Apply sequence of match-action tables

- ***Distinct & Reduce***
  - Compute index, & read value from hash tables
  - Apply function (e.g., bit_or for distinct) & then update the hash table
  - Use sketches, e.g. reduce(sum) → CM Sketches

- ***Limitations***
  - Complex transformations, e.g. log, regex, etc.

# Compiling Dataflow Queries

- ***Compiling a Single Query***
  - Generate & update query-specific metadata fields
  - Apply operator's match-action tables in sequence
  - Clone packet if ***report bit*** set

- ***Compiling Multiple Queries***
  - Generate & update metadata fields for all queries
  - Apply operators for all queries in sequence
  - Clone packet if ***report bit*** is set for at least one query

# Coordinated Data Coll. & Analysis

- ***Query Partitioning***
  - Execute subset of dataflow operators in data plane
  - Reduce packet tuples at the cost of additional state in the data plane

- ***Iterative Refinement***
  - Iteratively zoom-in on traffic of interests
  - Reduce state at the cost of additional detection delay

How to select the best query plan?

# Query Planning

- ***Reflection Attack Query***

```
pktStream(W)
.filter(p => p.srcPort == 53)
.map(p => (p.dstIP, p.srcIP))
.distinct()
.map((dstIP, srcIP)=>(dstIP,1))
.reduceByKey(sum)
.filter((dstIP,count)=>count>Th)
.map((dstIP, count) => dstIP)
```

- ***Partitioning Plans***

  Plan 1: Data Plane only

  Plan 2: Stream Processor only

- ***Refinement Plans***
  - Refinement key: dstIP
  - Refinement levels: {/8, /32}

# Query Planning

- ***Reflection Attack Query***

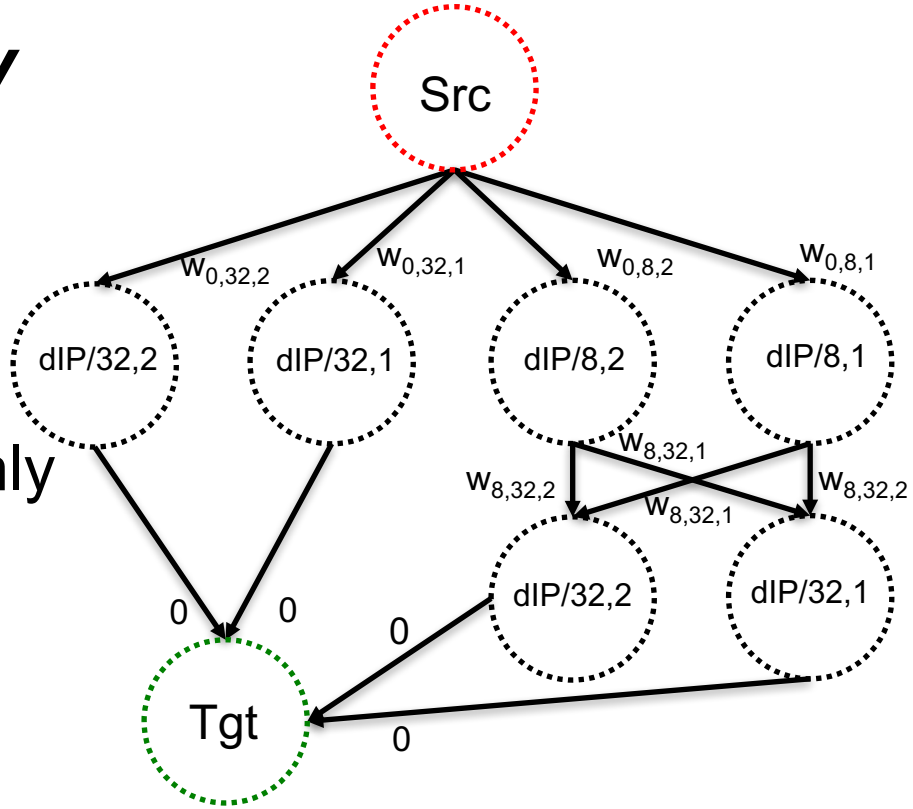
- ***Partitioning Plans***

  Plan 1: Data Plane only

  Plan 2: Stream Processor only

- ***Refinement Plans***
  - Refinement key: dstIP
  - Refinement levels: {/8, /32}

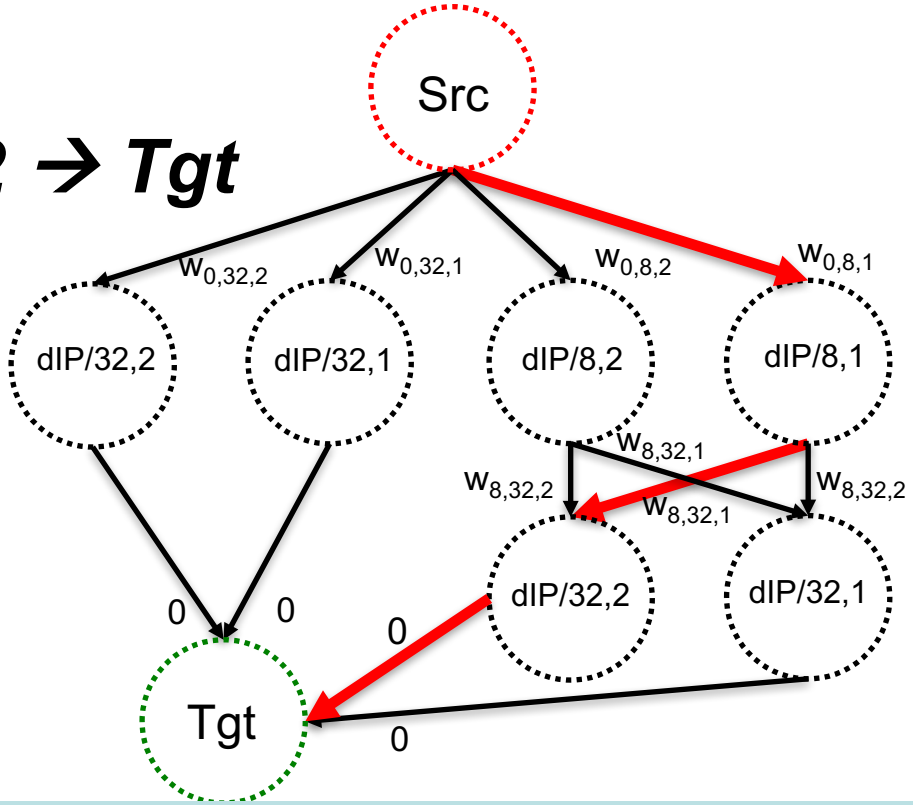
Query Plan Graph



# Query Planning

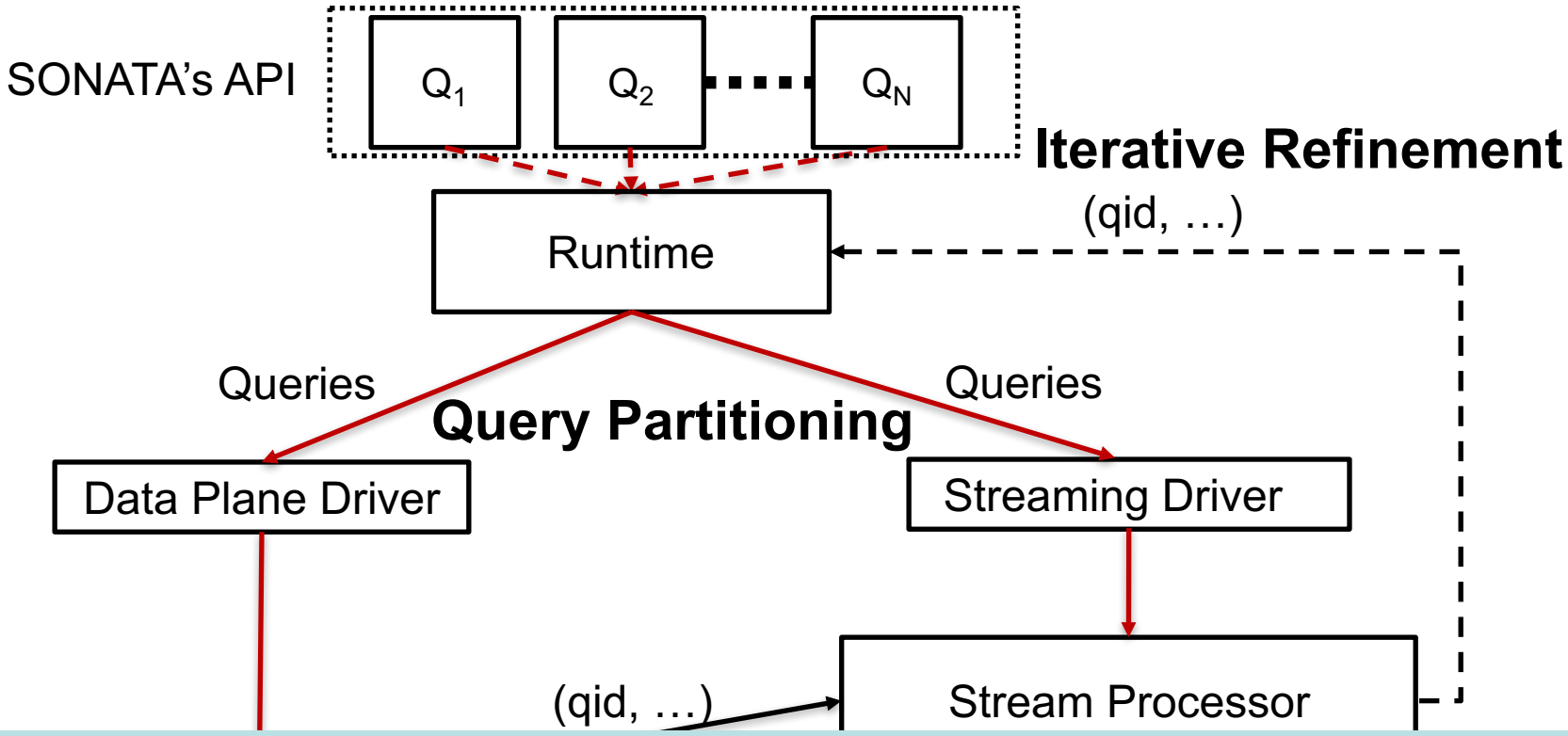
***Src → dIP/8,1 → dIP/32,2 → Tgt***

Query Plan Graph




Selects plan with smallest weighted cost

# Implementation




**Collection is now driven by Analysis!**

Data Plane Target

# Evaluation

- ***Workload***

  Large-IXP network: 2 hours long IPFIX trace, 3 Tbps peak traffic, packet sampling rate = 1/10K

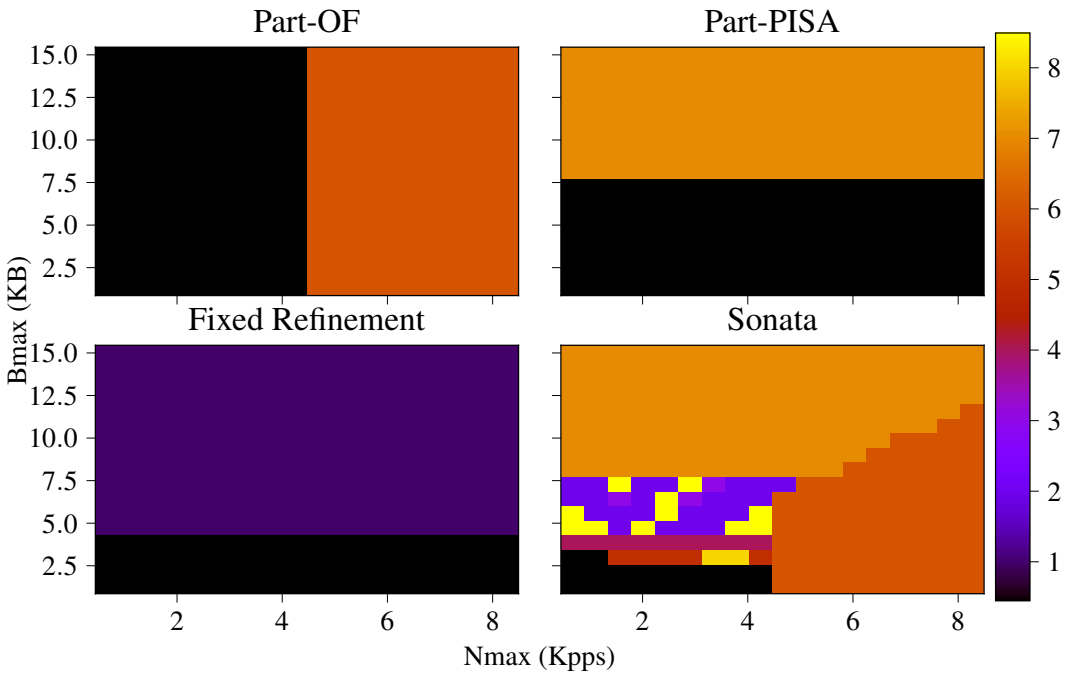
- ***Queries***

  DDoS-UDP, SSpreader, PortScan, Reflection Attack

- ***Comparisons***

  Stream-Only, Part-OF, Part-PISA, Fixed-Refinement

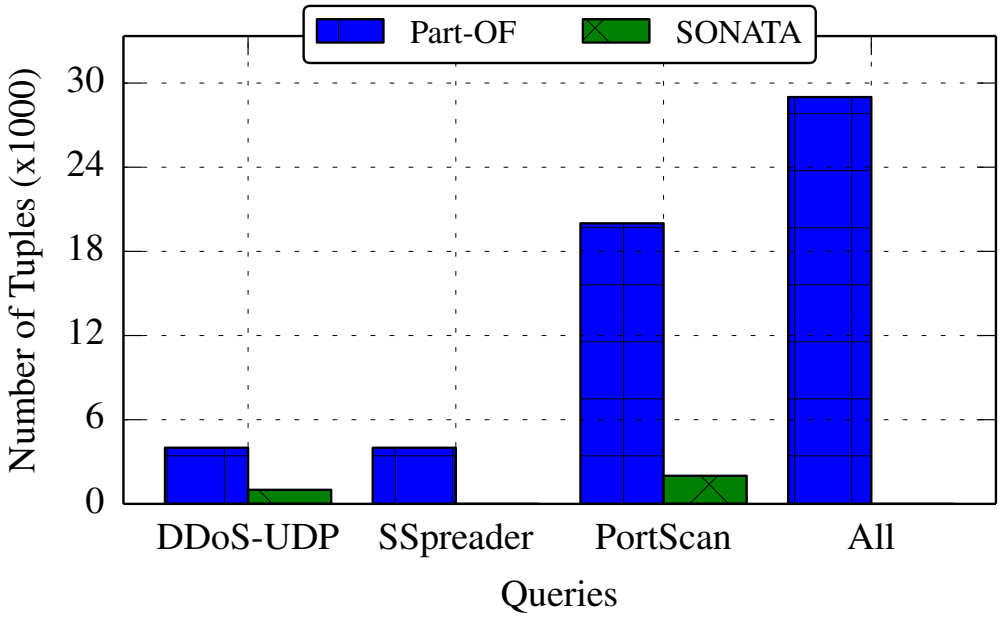
# Benefits of Query Planning




- $B_{max}$: Max. state data plane can support
- $N_{max}$: Max. pkt. tuples stream processor can process
- Each color represents a unique query plan

SONATA makes best use of available resources

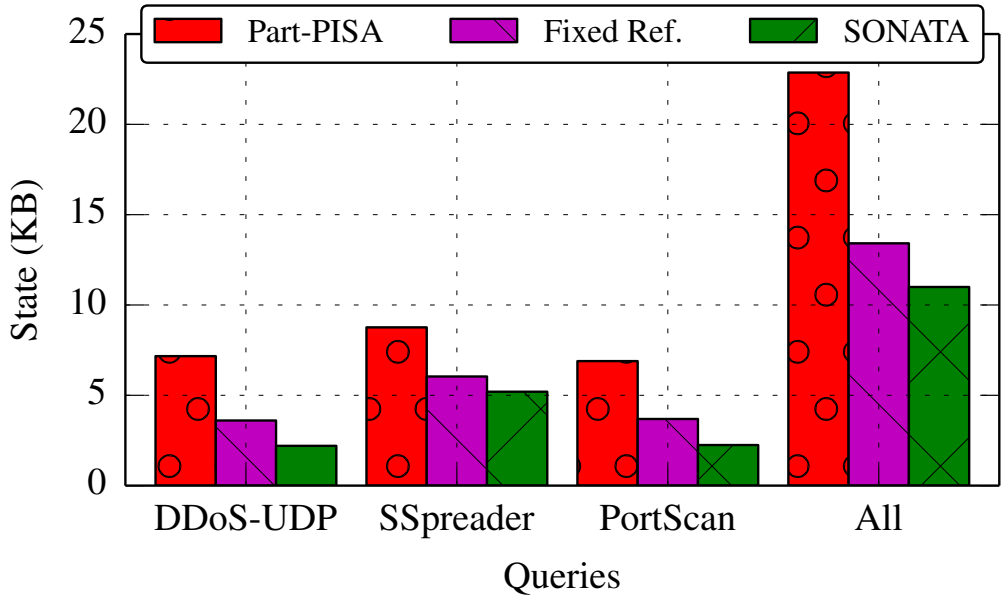
# Scaling Query Executions




Number of pkt tuples processed by Stream Processor

Executing stateful operations in data plane reduces workload on Stream Proc.

# Scaling Query Executions




State (KB) required by data plane targets

Iterative refinement reduces state required by the data plane targets

# Changing Status Quo

- *Expressibility*
  - Express Dataflow queries over packet tuples
  - Not worry about how and where the query is executed
  - Adding new queries and collection tools is trivial

- *Scalability*
  - Answers hundreds of queries in real-time for traffic volume as high as few Tb/s

Expressible & Scalable!

- tuples processed by the stream processor
- state in the data plane

# Summary

- SONATA makes it easier to **express** and **scale** network monitoring queries using
  - Programmable Data Plane
  - Scalable Stream Processor

- Running Code
  - Github: github.com/Sonata-Princeton/SONATA-DEV
  - Run test queries or express new ones

- SONATA@arxiv: arxiv.org/abs/1705.01049