NAP: Programming Data Planes with Approximate Data Structures

Mengying Pan, Hyojoon Kim, Jennifer Rexford, and David Walker

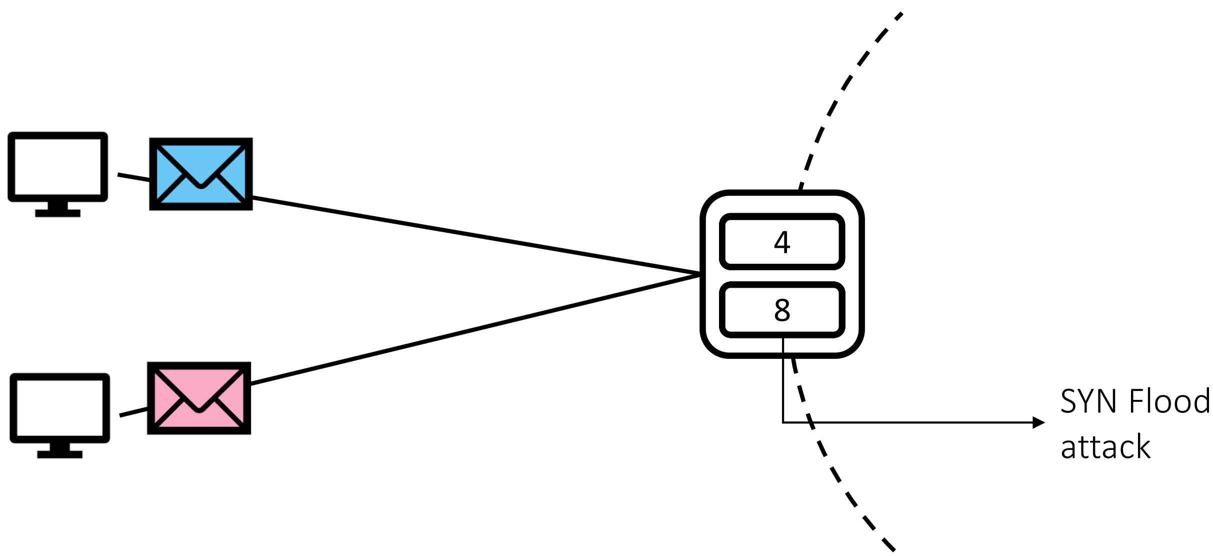
PRINCETON UNIVERSITY

Stateful applications

- Many applications keep **stateful information** on the data plane.
- The **limited memory** on the data plane makes it impossible to keep **exact per-flow state** in data structures.

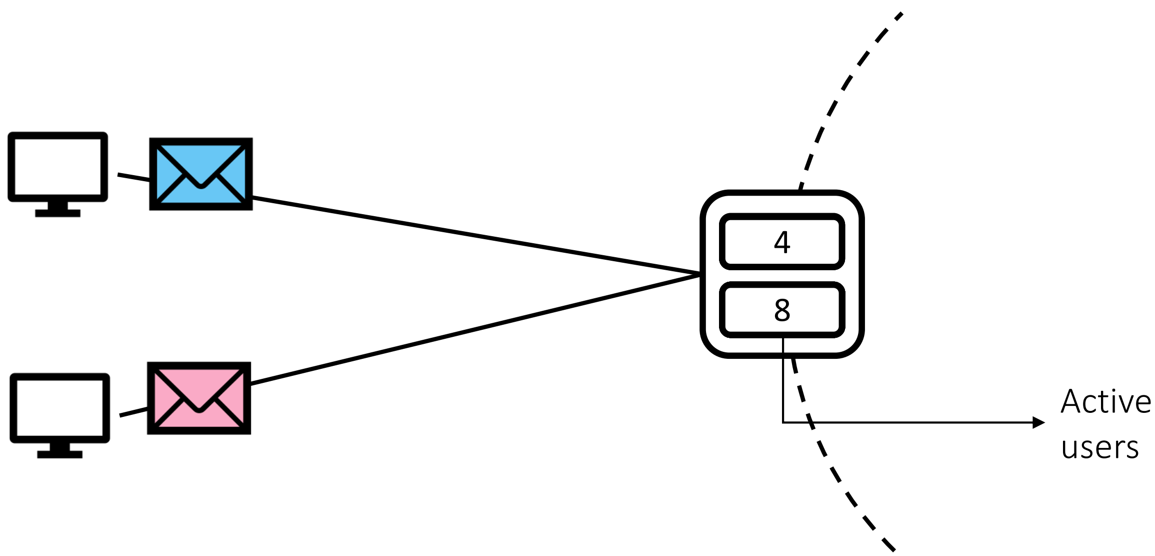
Approximate stateful application

- Many applications tolerate **errors of a specific direction**.
  - SYN flood detector: overapproximate the counts



Approximate stateful application

- Many applications tolerate **errors of a specific direction**.
  - SYN flood detector: overapproximate the counts
  - Rate limiter: underapproximate the counts



Approximate data structures

Network applications need to use **approximate data structures** to represent information compactly.

Bloom Filter	BeauCoup
Count-Min Sketch	CocoSketch
Cache	Cache with Fingerprint

Challenges

- **Selecting** the data structures:
  Which approximate data structure supports the desired state and the error direction?

Bloom Filter	BeauCoup
Count-Min Sketch	CocoSketch
Cache	Cache with Fingerprint

Challenges

- **Selecting** the data structures:
  Which approximate data structure supports the desired state and the error direction?

Data Structure

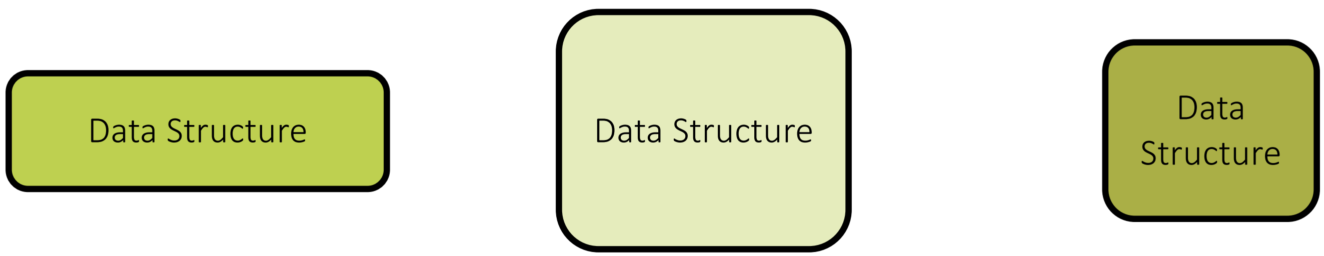
Challenges

- **Selecting** the data structures
- **Sizing** the data structures
  How to size the data structures to minimize the approximation error?

Data Structure

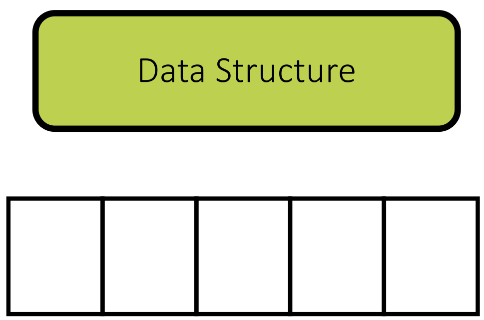
Challenges

- **Selecting** the data structures
- **Sizing** the data structures
  How to size the data structures to minimize the approximation error?



Challenges

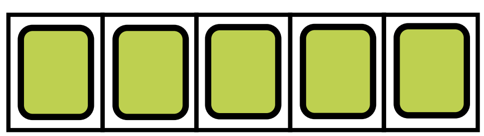
- **Selecting** the data structures
- **Sizing** the data structures
- **Tailoring** the data structures:
  How to implement the data structure to fit within the architecture?



Data Plane

Challenges

- **Selecting** the data structures
- **Sizing** the data structures
- **Tailoring** the data structures:
  How to implement the data structure to fit within the architecture?



Data Plane

A high-level language

- **Selecting** the data structures ☑
- **Sizing** the data structures ☑
- **Tailoring** the data structures ☑

NAP: **N**etwork **A**pproximate Data Structure **P**rogramming Language

- A simple and intuitive **abstraction** for approximate data structures
- An optimizing **compiler** that generates data plane implementation

Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information

Bloom Filter

BeauCoup

Count-Min Sketch

CocoSketch

Cache

Cache with Fingerprint

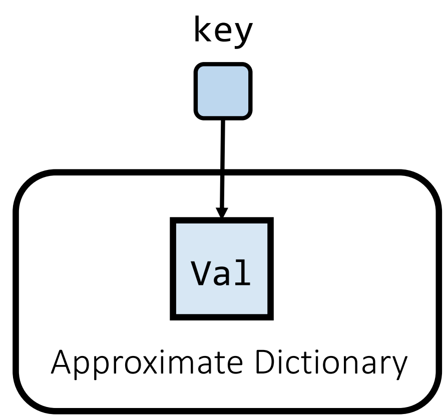
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information

Approximate Dictionary

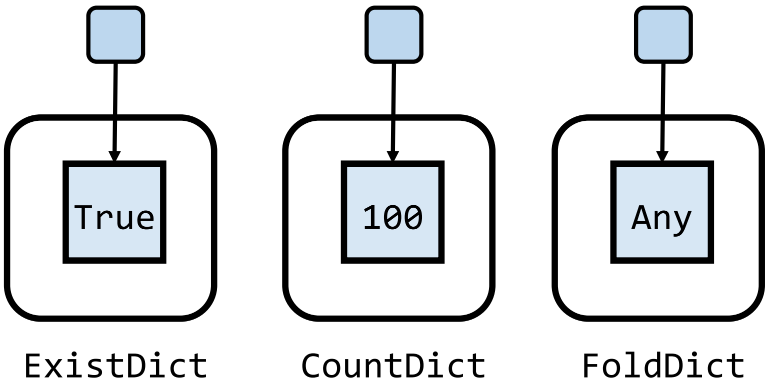
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`



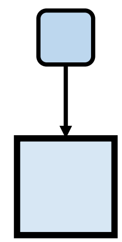
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
  - `Exist: Query(key) -> Bool`
  - `Count: Query(key) -> Int`
  - `Fold: Query(key) -> Any`



Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters**:
  - **Error direction**: inclusion approximation

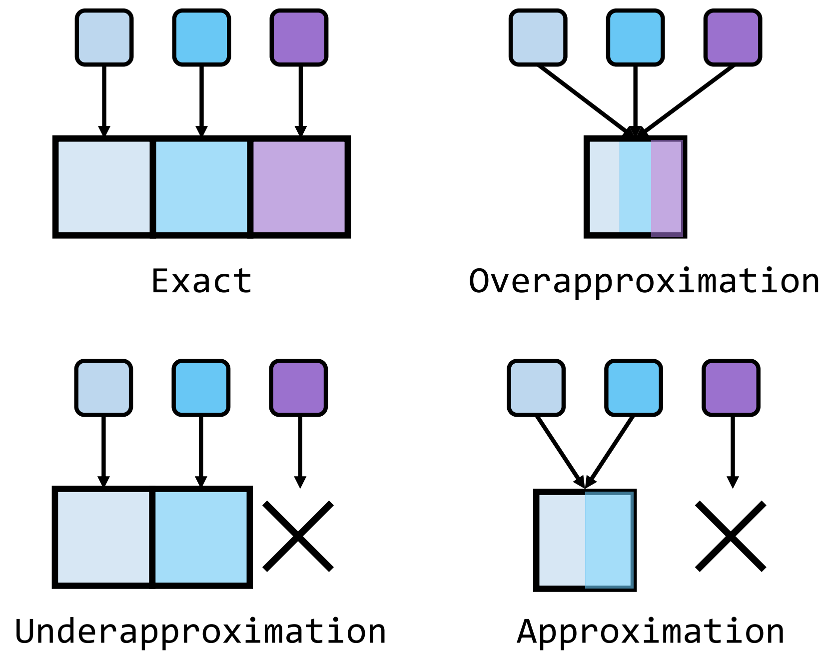


Many applications have a preference on the **error direction**.

- **SYN Flood detector**: overapproximation
- **Rate limiter**: underapproximation

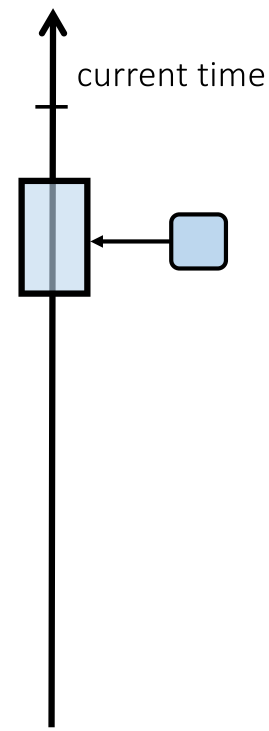
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters**:
  - **Error direction**: inclusion approximation



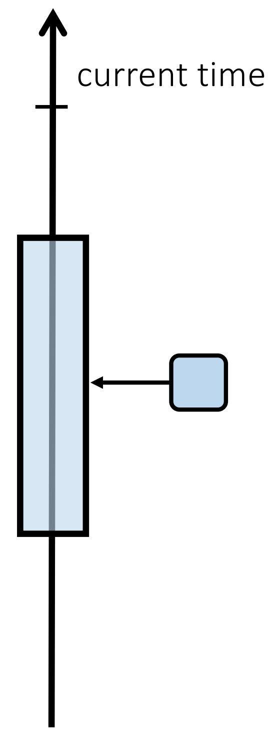
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters**:
  - **Error direction**: inclusion approximation
  - **Time window**: temporal approximation



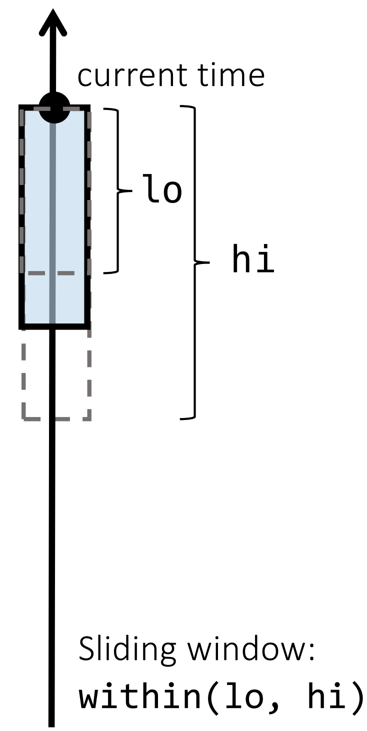
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters:**
  - **Error direction**: inclusion approximation
  - **Time window**: temporal approximation



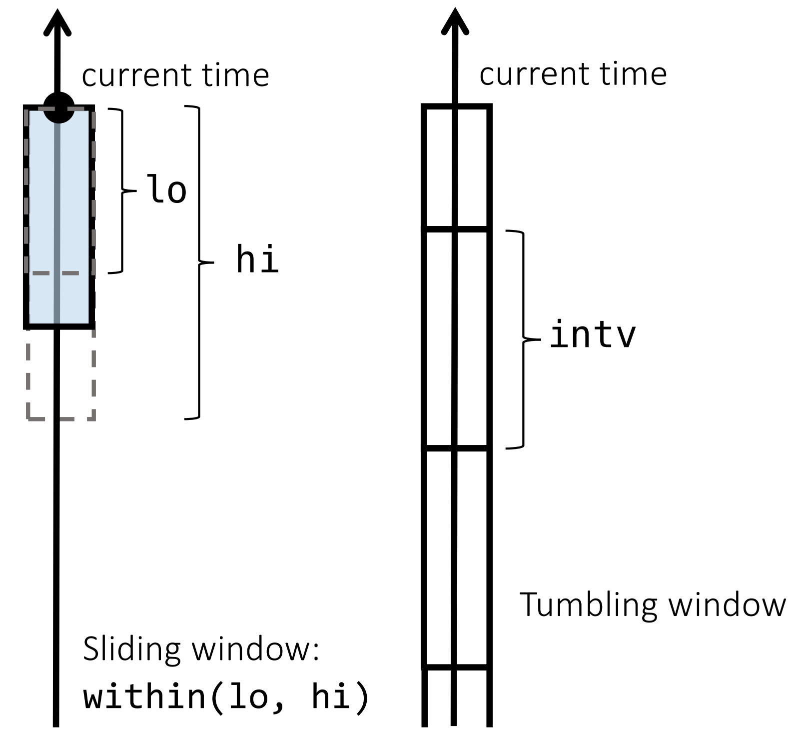
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters**:
  - **Error direction**: inclusion approximation
  - **Time window**: temporal approximation



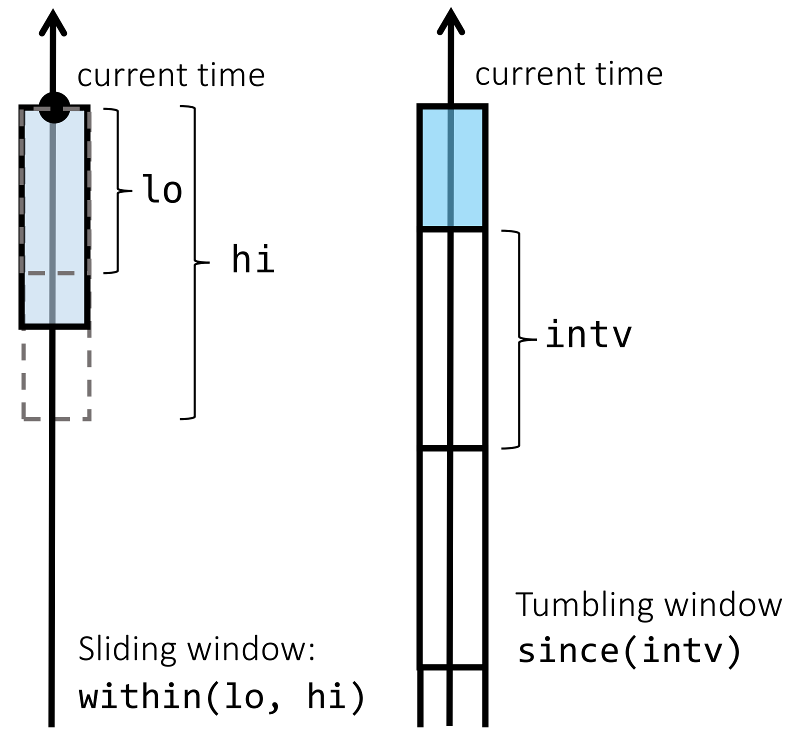
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters**:
  - **Error direction**: inclusion approximation
  - **Time window**: temporal approximation



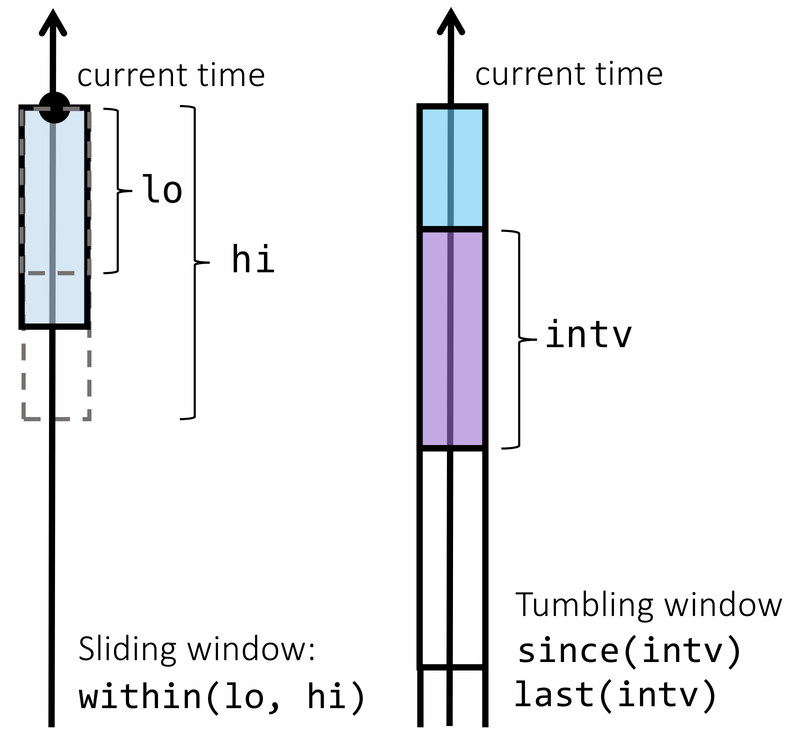
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters**:
  - **Error direction**: inclusion approximation
  - **Time window**: temporal approximation



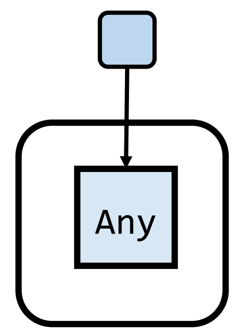
Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters**:
  - **Error direction**: inclusion approximation
  - **Time window**: temporal approximation



Abstraction: approximate dictionary

- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters**:
  - **Error direction**: inclusion approximation
  - **Time window**: temporal approximation
  - **Value state machine**



FoldDict

Example: SYN flood detector

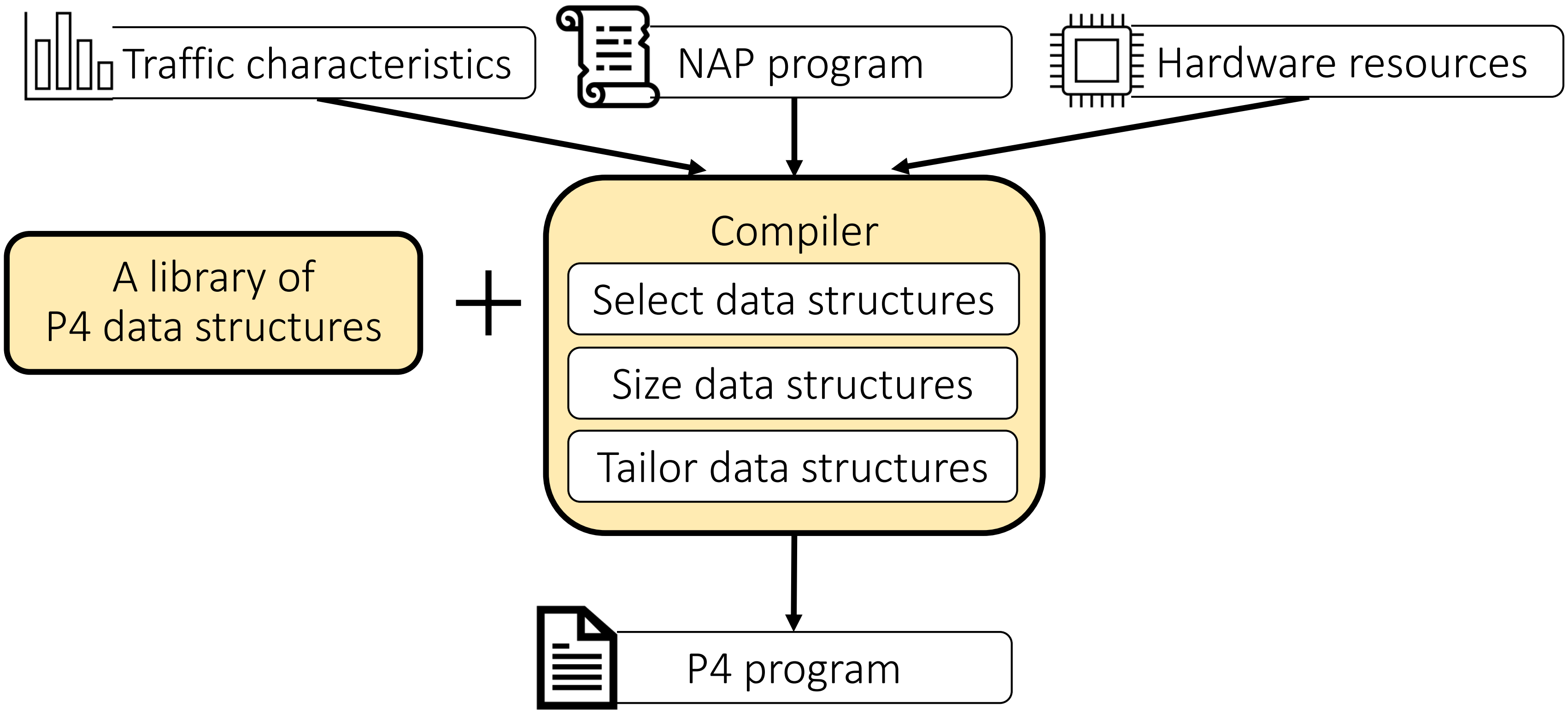
- **Key**: flow identifier
- **Value**: stateful information
- **Operations**:
  - `Create<key>(parameters)`
  - `Add(key)`
  - `Query(key)`
- **Dictionary Class**: value updates
- **Parameters:**
  - **Error direction**: inclusion approximation
  - **Time window**: temporal approximation
  - **Value state machine**

```
type key = {int sip}
CountDict<key> counters =
    CountDict.create (over,
                      within(sec(60),sec(90)),
                      CountDict())

...

if (p.tcp.flags = SYN) then {
    c = CountDict.add_query (counters,
                             {sip=p.ip.sip});
}
...
```

Compiler


Traffic characteristics
NAP program
Hardware resources
A library of
P4 data structures
+
Compiler
Select data structures
Size data structures
Tailor data structures
P4 program

Compiler: select data structures

- **Dictionary classes**:
  - `ExistDict`
  - `CountDict`
  - `FoldDict`
- **Error directions**:
  - `Exact`
  - `Overapproximation`
  - `Underapproximation`
  - `Approximation`

	`CountDict`
`Exact`	Exact array
`Over`	Count-min sketch
`Under`	Cache with full fingerprint
`Approx`	All of above, Cache w. partial fingerptint

```
type key = {int sip}
CountDict<key> counters =
    CountDict.create (over,
                      within(sec(60),sec(90)),
                      CountDict())
```

Compiler: size data structures

- **Parametrize** the data structures
- **Constrained optimization problem**

```
Minimize:
    Expected error of count-min sketch

Constrained by:
    • Memory constraints
    • Computational constraints
    • Architectural constraints
```

Evaluation

- **Generalizability**
  a diverse set of nine example applications in network telemetry, monitoring, and control

Applications	LoC		Compile Time (s)
	NAP	P4	
Single Dictionary			
Stateful firewall	15	555	0.0055
DNS amplification mitigation	15	582	0.0056
FTP monitoring	20	798	0.0035
Heavy hitter detection	8	595	0.0049
Traffic rate measurement by IP/8	12	466	0.0040
TCP out-of-order monitoring	19	559	0.0043
Multiple Dictionaries			
TCP superspreader detection	20	842	0.0130
TCP SYN flood detection	20	842	0.0130
NetCache	22	802	0.0394

Evaluation

- **Generalizability**
- **Simplicity**
  - All example applications expressed within 30 LoC
  - A reduction of 25X to 50X in LoC

Applications	LoC		Compile Time (s)
	NAP	P4	
Single Dictionary			
Stateful firewall	15	555	0.0055
DNS amplification mitigation	15	582	0.0056
FTP monitoring	20	798	0.0035
Heavy hitter detection	8	595	0.0049
Traffic rate measurement by IP/8	12	466	0.0040
TCP out-of-order monitoring	19	559	0.0043
Multiple Dictionaries			
TCP superspreader detection	20	842	0.0130
TCP SYN flood detection	20	842	0.0130
NetCache	22	802	0.0394

Evaluation

- **Generalizability**
- **Simplicity**
- **Fast compilation**
  - All examples compiled to P4 for the Intel Tofino target within 0.1 second

Applications	**LoC**		**Compile Time (s)**
	NAP	**P4**	
Single Dictionary			
Stateful firewall	15	555	0.0055
DNS amplification mitigation	15	582	0.0056
FTP monitoring	20	798	0.0035
Heavy hitter detection	8	595	0.0049
Traffic rate measurement by IP/8	12	466	0.0040
TCP out-of-order monitoring	19	559	0.0043
Multiple Dictionaries			
TCP superspreader detection	20	842	0.0130
TCP SYN flood detection	20	842	0.0130
NetCache	22	802	0.0394

Conclusions

- NAP is a **general and simple** language for approximate data structures.
- NAP **selects, sizes and tailors** approximate data structures.
- Future directions:
  - More dictionary classes
  - Multi-pipeline
  - Multi-target