NAP: Programming Data Planes with Approximate Data Structures

Mengying Pan, Hyojoon Kim, Jennifer Rexford, and David Walker
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
- Count-Min Sketch
- Cache
- BeauCoup
- CocoSketch
- Cache with Fingerprint
Challenges

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

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

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

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

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

**NAP**: Network Approximate Data Structure Programming 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
- Count-Min Sketch
- Cache
- BeauCoup
- CocoSketch
- Cache with Fingerprint
Abstraction: approximate dictionary

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

Sliding window: `within(lo, hi)`
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

![Diagram showing sliding and tumbling windows](image)
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

![Diagram showing sliding and tumbling windows with parameters](current time)
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

---

Sliding window: within(lo, hi)

Tumbling window:
- since(intv)
- last(intv)
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**
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**

```haskell
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
A library of P4 data structures
NAP program
Hardware resources

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

---

<table>
<thead>
<tr>
<th></th>
<th>CountDict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>Exact array</td>
</tr>
<tr>
<td>Over</td>
<td>Count-min sketch</td>
</tr>
<tr>
<td>Under</td>
<td>Cache with full fingerprint</td>
</tr>
<tr>
<td>Approx</td>
<td>All of above, Cache w. partial fingerprint</td>
</tr>
</tbody>
</table>

type key = {int sip}

```java
CountDict<key> counters = CountDict.create (over, within(sec(60),sec(90)), CountDict())
```
Compiler: size data structures

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

<table>
<thead>
<tr>
<th>Minimize:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected error of count-min sketch</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constrained by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Memory constraints</td>
</tr>
<tr>
<td>• Computational constraints</td>
</tr>
<tr>
<td>• Architectural constraints</td>
</tr>
</tbody>
</table>
Evaluation

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

<table>
<thead>
<tr>
<th>Applications</th>
<th>LoC</th>
<th>Compile Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NAP</td>
<td>P4</td>
</tr>
<tr>
<td><strong>Single Dictionary</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stateful firewall</td>
<td>15</td>
<td>555</td>
</tr>
<tr>
<td>DNS amplification mitigation</td>
<td>15</td>
<td>582</td>
</tr>
<tr>
<td>FTP monitoring</td>
<td>20</td>
<td>798</td>
</tr>
<tr>
<td>Heavy hitter detection</td>
<td>8</td>
<td>595</td>
</tr>
<tr>
<td>Traffic rate measurement by IP/8</td>
<td>12</td>
<td>466</td>
</tr>
<tr>
<td>TCP out-of-order monitoring</td>
<td>19</td>
<td>559</td>
</tr>
<tr>
<td><strong>Multiple Dictionaries</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP superspreader detection</td>
<td>20</td>
<td>842</td>
</tr>
<tr>
<td>TCP SYN flood detection</td>
<td>20</td>
<td>842</td>
</tr>
<tr>
<td>NetCache</td>
<td>22</td>
<td>802</td>
</tr>
</tbody>
</table>
Evaluation

• **Generalizability**

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

<table>
<thead>
<tr>
<th>Applications</th>
<th>LoC</th>
<th>Compile Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NAP</td>
<td>P4</td>
</tr>
<tr>
<td><strong>Single Dictionary</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stateful firewall</td>
<td>15</td>
<td>555</td>
</tr>
<tr>
<td>DNS amplification mitigation</td>
<td>15</td>
<td>582</td>
</tr>
<tr>
<td>FTP monitoring</td>
<td>20</td>
<td>798</td>
</tr>
<tr>
<td>Heavy hitter detection</td>
<td>8</td>
<td>595</td>
</tr>
<tr>
<td>Traffic rate measurement by IP/8</td>
<td>12</td>
<td>466</td>
</tr>
<tr>
<td>TCP out-of-order monitoring</td>
<td>19</td>
<td>559</td>
</tr>
<tr>
<td><strong>Multiple Dictionaries</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP superspreader detection</td>
<td>20</td>
<td>842</td>
</tr>
<tr>
<td>TCP SYN flood detection</td>
<td>20</td>
<td>842</td>
</tr>
<tr>
<td>NetCache</td>
<td>22</td>
<td>802</td>
</tr>
</tbody>
</table>
Evaluation

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

<table>
<thead>
<tr>
<th>Applications</th>
<th>LoC</th>
<th>Compile Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NAP</td>
<td>P4</td>
</tr>
<tr>
<td>Single Dictionary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stateful firewall</td>
<td>15</td>
<td>555</td>
</tr>
<tr>
<td>DNS amplification mitigation</td>
<td>15</td>
<td>582</td>
</tr>
<tr>
<td>FTP monitoring</td>
<td>20</td>
<td>798</td>
</tr>
<tr>
<td>Heavy hitter detection</td>
<td>8</td>
<td>595</td>
</tr>
<tr>
<td>Traffic rate measurement by IP/8</td>
<td>12</td>
<td>466</td>
</tr>
<tr>
<td>TCP out-of-order monitoring</td>
<td>19</td>
<td>559</td>
</tr>
<tr>
<td>Multiple Dictionaries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP superspreader detection</td>
<td>20</td>
<td>842</td>
</tr>
<tr>
<td>TCP SYN flood detection</td>
<td>20</td>
<td>842</td>
</tr>
<tr>
<td>NetCache</td>
<td>22</td>
<td>802</td>
</tr>
</tbody>
</table>
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