Quantitative Network Monitoring with NetQRE

Yifei Yuan\textsuperscript{1,2}, Dong Lin\textsuperscript{1,3}, Ankit Mishra\textsuperscript{1}, Sajal Marwaha\textsuperscript{1}, Rajeev Alur\textsuperscript{1}, and Boon Thau Loo\textsuperscript{1}

\textsuperscript{1}University of Pennsylvania
\textsuperscript{2}Carnegie Mellon University
\textsuperscript{3}LinkedIn Inc.
Network Monitoring is Important

- Security
  - Heavy hitter
  - Super spreader
  - Syn flood
  - Slowloris
  - …

- Performance
  - Traffic matrix
  - Application usage
  - …
Today’s Low-level Programming Abstraction

Packet stream

```
state s = s_0;
Upon receiving packet p {
    s = update(s, p);
    output d = decide(s);
}
```

Streaming algorithm

Monitoring results
Motivating Example: VoIP Monitoring

Example Policy:
1. Monitor average number of VoIP calls per user
2. Alert a user, if her/his number of calls exceeds a threshold

Stateful: Need to maintain state to track VoIP sessions with each incoming packet
Quantitative: Need to compute numerical aggregate based on metrics of past history and across users

What low-level state to maintain? How to update it?
NetQRE Overview

- Stream-level abstraction

- Policy Specification in NetQRE

- NetQRE Compiler
  - Automatic state inference
  - Efficient state update
  - Automatic optimization

- Modular programming

- Monitoring query implementation

- Monitoring results
Outline

• Motivation
• NetQRE language
• NetQRE compiler
• Implementation
• Evaluation
Outline

• Motivation
• NetQRE language
• NetQRE compiler
• Implementation
• Evaluation
Modular Programming of VoIP Monitoring

• Input: packet stream from all users
• Output: average number of VoIP calls per user
• Procedure:
  • Step 1: Focus on the packet stream from an arbitrary user \( x \)
  • Step 2: View the stream as a sequence of calls, and identify each call
  • Step 3: Aggregate across all calls in the stream of the user
  • Step 4: Aggregate across all users
Step 1: Focus on Packet Stream of User x

Legend
- Blue: Packets from user x
- Orange: Packets from other users

filter(usr=x)
Step 2: Identify A Call

```
VoIP Call  VoIP Call  ..........  VoIP Call
```

Legend
- Packets from user x
- Packets from other users

How to specify the pattern of a VoIP call?
Step 2: Identify A Call

filter(usr=x)

Legend
- Blue: Packets from user x
- Orange: Packets from other users

Regular pattern:

\[ \text{re\_call} = [\text{invite}] [200] [\text{ack}] [\text{data}]^* [\text{BYE}] [200] \]
Step 2: Identify A Call

How to associate a numerical value with each pattern?
Step 2: Identify A Call

call = re_call?1 call call

filter(usr=x)

Legend
- Packets from user x
- Packets from other users
Step 3: Aggregate over All Calls

VoIP Call VoIP Call .......... VoIP Call

filter(usr=x)

call = re_call?1 call call

iter(call, sum)
Stream Iteration: \textbf{iter}(f, \text{aggop})

- \(f\) is a NetQRE function
- \text{aggop} is an aggregation operator, such as \text{sum}, \text{avg}, \text{max}, \text{min}.
- Split the stream into multiple substreams \(s_1, \ldots, s_n\) such that \(f\) is defined on each \(s_i\)
- Returns \(\text{aggop}(f(s_1), \ldots, f(s_n))\)
Step 4: Aggregation over All Users

Legend
- Blue: Packets from user x
- Orange: Packets from other users

\[ \text{call}_\text{usr}(x) = \text{iter}(\text{call}, \text{sum}) \]

\[ \text{avg}\{ \text{call}_\text{usr}(x) \mid \text{User } x \} \]
## Requirements & Key Ideas

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Key Ideas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern matching for recognizing traffic patterns</td>
<td>Regular expression (RE) for pattern matching</td>
</tr>
<tr>
<td>Handle arbitrary &amp; unknown value</td>
<td>Parametric extension to RE</td>
</tr>
<tr>
<td>Quantitative aggregations</td>
<td>Quantitative extension to RE</td>
</tr>
</tbody>
</table>
### NetQRE Language

#### Regular Expression
- **Atoms**: letters
  - E.g. a, b, ...
- **Base RE**: atoms
- **Union**: $f \mid g$
- **Concatenation**: $f \circ g$
- **Kleene star**: $f^*$

#### Parametric extension

#### NetQRE
- **Atoms**: predicate over packets
  - E.g. $[\text{srcip}=x], [\text{dstip}=10.0.0.1]$
- **Base NetQRE**: $\text{re} \ ? \ \nu$
- **Choice**: $\text{re} \ ? \ f : g$
- **Split**: $\text{split}(f, g, \text{aggop})$
- **Iteration**: $\text{iter}(f, \text{aggop})$
- **Aggregation over parameter**:
  - $\text{aggop}\{ f(x) \mid \text{Type } x \}$
- **Streaming composition**: $f \gg g$

Details in the paper
Stream Split: \texttt{split}(f, g, \text{aggop})

- \(f\) and \(g\) are two NetQRE functions
- \(\text{aggop}\) is an aggregation operator, such as \texttt{sum}, \texttt{avg}, \texttt{max}, \texttt{min}.
- Split the stream into two substreams \(s_1\) and \(s_2\), such that \(f\) is defined on \(s_1\) and \(g\) is defined on \(s_2\)
- Returns \(\text{aggop}(f(s_1), g(s_2))\)
Outline

• Motivation
• NetQRE language
• NetQRE compiler
• Implementation
• Evaluation
NetQRE Compilation

- Goal: Evaluating a query online with small state
  - Independent of length of packet stream
- Insight: Leverage compilation of regular expression to DFA
- Question 1: How to handle parameters?
  - Insight: Lazy instantiation
- Question 2: How to evaluate `split(f, g, aggop)` and `iter(f, aggop)` online?
  - Insight: Keep all possible (but bounded number of) cases
  - Details in the paper
Compilation of RE with Parameters

exist_{src}(x)

srcip != x

x=0:

Eager Instantiation

Lazy Instantiation

x=2^{32}:
Implementation

- Single-node deployment
- Compiler implemented in C++
- Compiled code C++
- pcap library for packet capturing
Outline

• Motivation
• NetQRE language
• NetQRE compiler
• Implementation
• Evaluation
Evaluation

• Is the NetQRE language expressive?
• Is the NetQRE compiled implementation efficient?
## Evaluation: Expressiveness

<table>
<thead>
<tr>
<th>Function</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heavy hitter</td>
<td>6</td>
</tr>
<tr>
<td>Super spreader</td>
<td>2</td>
</tr>
<tr>
<td>Entropy estimation</td>
<td>6</td>
</tr>
<tr>
<td>Flow size distribution</td>
<td>8</td>
</tr>
<tr>
<td>Traffic change detection</td>
<td>10</td>
</tr>
<tr>
<td>Count traffic</td>
<td>2</td>
</tr>
<tr>
<td>Completed flows</td>
<td>6</td>
</tr>
<tr>
<td>SYN flood detection</td>
<td>9</td>
</tr>
<tr>
<td>Slowloris detection</td>
<td>12</td>
</tr>
<tr>
<td>Connection lifetime</td>
<td>8</td>
</tr>
<tr>
<td>Newly opened connection</td>
<td>11</td>
</tr>
<tr>
<td># duplicated ACKs</td>
<td>5</td>
</tr>
<tr>
<td># VoIP calls</td>
<td>7</td>
</tr>
<tr>
<td>VoIP usage</td>
<td>18</td>
</tr>
<tr>
<td>DNS tunnel detection</td>
<td>4</td>
</tr>
<tr>
<td>DNS amplification</td>
<td>4</td>
</tr>
</tbody>
</table>

- **Expressive**
  - 100-1000+ LoC in compiled implementation
  - 100+ LoC in manual implementation

- **Concise**
Evaluation: Throughput

- heavy hitter
- super spreader
- entropy
- syn flood
- flow num
- slowloris

Throughput (MPPS)

- Baseline
- NetQRE

9% reduction
Evaluation: Memory

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>NetQRE</th>
<th>~40% increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>heavy hitter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>super spreader</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>entropy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>syn flood</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flow num</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slowloris</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory (MB)
Conclusion

• Motivation: Network monitoring needs high-level abstractions

• Contributions:
  • Stream-level programming abstraction
  • Parametric and quantitative extension to regular expressions
  • Expressive to capture a wide range of monitoring policies
  • Compiled implementation efficient in both throughput and memory

• Future work:
  • Hardware implementation
  • Distributed deployment of NetQRE programs