Quantitative Policies over Streaming Data

Rajeev Alur
University of Pennsylvania
Thanks to Collaborators

Dana Fisman
Sanjeev Khanna
Boon Thau Loo

Kostas Mamouras
Mukund Raghothaman
Yifei Yuan
Real-time Decision Making in IoT Applications

data  ➔  Controller  ➔  decisions
Variable Tolling

Adjust toll rate at each toll booth dynamically based on time of day and congestion conditions in road segments.

Reference: Linear road benchmark for stream management systems.
Network Traffic Engineering

Dynamic network management for traffic engineering
Real-time response to emerging attacks / security threats

Software Defined Networking (SDN)
Opportunity for increased programmability/functionality
Example network policy:

if number of packets in current VoIP session exceeds the average over past VoIP sessions by a threshold $T$, then drop the packet

Stateful: Need to maintain state and update it with each item
Quantitative: Based on numerical aggregate metrics of past history
Design and Implementation of Policies

Which policies are effective?
Based on traffic models and domain specific insights

How to specify and implement policies?
Focus of this talk!
Streaming Algorithm

state $s = \text{initialize}$;
for each packet $p$ {
    $s = \text{update} (s, p)$;
    output $d = \text{decide} (s)$
}
High-level Abstractions over Data Streams ??

Example network policy:
if number of packets in current VoIP session exceeds the average over past VoIP sessions by a threshold T then drop the packet

Low-level programming:
What state to maintain? How to update it?

Desired high-level abstraction: Beyond packet sequence
Modular Monitoring of VoIP Sessions

1. Focus on traffic between a specific source and destination

2. View data stream as a sequence of VoIP session

3. View a VoIP session as split in three phases

4. Aggregate cost over call phase during a session, and aggregate cost across sessions

Session Initiation Protocol
Design Goals for Policy Language

Efficiency critical: Key parameters
1. Time to process each packet
2. State that needs to maintained
Ideally both should be constant or logarithmic in length of data stream

Programming abstractions for processing data stream ??

Theoretical foundations
Expressiveness
Optimization

Policy spec
Policy compiler
Policy code
decisions
Talk Outline

✓ Motivation

✓ Quantitative Regular Expressions (QRE)

☐ QRE Compilation

☐ Implementation and Evaluation in SDN Platform

☐ Research Opportunities
Illustrative Example: Bank Transactions

Data stream: Sequence of bank transactions of a single customer

Data items:
- Transaction T with a positive/negative numerical amount
- End-of-day marker D
- End-of-month marker M

Compute maximum over months,
average net-deposit per day during a month
Daily Transactions

What pattern specifies substream of transactions during a single day?

Regular expressions:

Excellent match for specifying patterns of substreams of data
Net Deposit During a Day

Regular Expressions + Aggregation operators

Base function $T$ maps an item, if it’s a transaction, to its amount
Expression $f$ maps a sequence of transactions to its sum

Base function $D$ maps an item, if it’s end-of-day marker, to 0
Expression $g$ maps daily transactions to net deposit during that day

$$f = \text{iter}(T, +)$$
$$g = \text{split}(f, D, +)$$
Average Daily Net-deposit In a Month

Expression $k$ matches streams with the pattern $(T^* \cdot D)^* \cdot M$ and maps it to average daily net-deposit.

Expression $k$ matches streams with the pattern $(T^* \cdot D)^* \cdot M$ and maps it to average daily net-deposit.
Completing the Query

Compute maximum over months,
average daily net-deposit per day during a month
Quantitative Regular Expressions

- Each QRE maps a sequence of data items to a cost value
- Definition parameterized by operations over costs
- Example: Set of integers with min, max, sum, average
- Domain of QRE = Sequences for which it is defined
  Characterized by a regular expression
- Base QRE: Maps individual data items to costs
- QREs combined using a few operators such as split, iter, else ...
Split Combinator: split(f, g, op)

f and g are QREs and op is a cost operation such as +, max, ...
Divide input data stream s into two parts s₁ and s₂ such that
  f is defined on s₁ and g is defined on s₂ and return op(f(s₁), g(s₂))
Key requirement: split must be unique (unambiguous)

Domain of f : Streams ending with a high-risk transaction (amount > 500)
Domain of g : Stream without high-risk transactions
Iterated Composition: \( \text{iter}(f, \text{op}) \)

- \( f \) is a QRE and \( \text{op} \) is an aggregation operation such as +, max, average...

- Divide input data stream \( s \) into multiple parts \( s_1, s_2, \ldots s_k \) such that \( f \) is defined on each \( s_i \) and return \( \text{op}( f(s_1), f(s_2), \ldots f(s_k)) \)

- Splitting must be unique!

- In full generality, QRE takes arguments and iteration is chained

  E.g. \( f(x) \) computes balance at end of a day using \( x \) as initial balance

  \( \text{iter}(f)(x) \) splits transaction stream into chunks corresponding to days
  \( f \) is applied to first day using initial balance \( x \)

  Result of this is used as initial balance to apply \( f \) to next day and so on
Conditional: $f$ else $g$

Given a stream $s$, if $f(s)$ is defined, return it, else return $g(s)$

Example: $f$ makes decisions for a stream that does not contain high-risk transactions (e.g. with amount $> threshold$), and $g$ makes decisions for streams that do contain such transactions

Benefit: Test based on a global property of stream
Parameterized QREs

Suppose stream contains transactions of Alice and Bob

Suppose we want to compute average net deposit during a day for each customer

\[ f(ID) \] maps a stream to average net deposit during a day by customer ID

Specified using QRE with ID as the parameter

\[ f(Alice) \] looks at only her transactions

\[ f(Bob) \] looks at only his transactions
Composing QREs

Suppose stream contains transactions of Alice and Bob

Compute the maximum over all customers,
average net deposit during a day by that customer

f(ID) maps a stream to average net deposit during a day by customer ID

Desired query : max ( f ( Alice ), f ( Bob ) )

Composition: op(f, g) where op is cost operation and f and g are QREs
This is the analog of “intersection” operation in regular expressions
Expressiveness

Do we have enough operators?
Is expressiveness of QREs robust?

Regular languages

- Regular expressions
- Deterministic finite automata
- Monadic second-order logic MSO

Beautiful well-understood theory

Regular functions parameterized by cost operations

- Quantitative regular expressions
- Cost register automata (CRA)
- MSO-definable string to term transformations

Emerging theory (open problems...)

Talk Outline

✓ Motivation

✓ Quantitative Regular Expressions (QRE)

❖ QRECompilation

☐ Implementation and Evaluation in SDN Platform

☐ Research Opportunities
Goals for Compiler

QRE

QRE compiler

state s = initialize;
for each packet p {
  s = update (s, p);
  output d = decide (s)
}

data

decisions

Optimize bits needed to store state and time for update
Ideally independent of length of data stream
Computing $f(s)$, where $f$ is a QRE and $s$ is input stream, amounts to evaluating an expression tree of size linear in length of $s$. 

Average net-deposit per day: $\text{iter (split (iter (T, +), D, +), average)}$
Incremental Construction of Expressions

Having seen a substream, desired result is an expression with variables

Variables $x$ and $y$ are placeholders to be expanded as more items arrive

Need a way to rewrite expressions to get a compressed representation
Expression Compression

Consider expressions with variable $x$, constants, and operators $\min$ and $+$

Rule 1: Fully evaluate each variable-free subexpression to a constant
Expression Compression

Consider expressions with variable $x$, constants, and operators min and +

Rule 2: Distribute + over min
Expression Compression

Consider expressions with variable $x$, constants, and operators $\text{min}$ and $+$. 

Rule 1: Fully evaluate each variable-free subexpression to a constant

Rule 2: Distribute $+$ over $\text{min}$

Every expression with a single variable $x$, constants, and operators $\text{min}$ and $+$ can be reduced to a canonical compressed form: $\text{min}(x + c, d)$

Generalizes to terms with multiple variables and operators of $+$, $\text{min}$, $\text{max}$, and average
Approximation

Suppose we want to compute average of numbers in a streaming fashion.

Need to remember total sum (73) and count of items (5) so far.

Suppose we want to compute median of numbers.

To ensure exact answer, must remember all numbers seen so far.

Approximation algorithm:

Map each number \( n \) to bucket \( k \) such that \((1+\epsilon)^k \leq n < (1+\epsilon)^{k+1}\).

Maintain for each bucket, count of numbers mapped to that bucket.

Space needed: \( \log U \), where \( U \) is the range of numerical values.

Approximation error: Multiplicative factor of \( \epsilon \).
Online Computation of Split Points

To process \( \text{split}(f,g,+)) \), find the position where \( f \) ends and \( g \) starts

Domain of \( f \): Streams ending with high-risk transaction (amount > 500)

Need to maintain multiple parallel computations of same subexpression initialized at different positions in input stream

Insight: number of parallel copies is bounded (bound depends on query)
QRE Compiler Summary

- Given a QRE, compiler first checks all typing rules are met (e.g. when split is applied, the splitting must be unambiguous)
- Then it compiles it into an executable streaming algorithm
- General case: Memory used is linear in length of stream
- If numerical operators are min, max, sum, average, then constant memory and constant per item processing time
- If, in addition, median is also used, then
  - log U memory, where U is (dynamically updated) range of values
  - constant time to process each item
  - user specified multiplicative factor of approximation error
Talk Outline

✓ Motivation

✓ Quantitative Regular Expressions (QRE)

✓ QRE Compilation

☒ Implementation and Evaluation in SDN Platform

☐ Research Opportunities
Software Defined Networking

<table>
<thead>
<tr>
<th>Dst</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Distributed Protocols**: e.g. POX, NOX, Floodlight
- **Openflow**: APIs
- **Programmability**: APIs
- **Controller**: Match | Action
  - Src=A | drop
  - ... | ...

- **Control plane**: Data plane
NetQRE Language

Domain-specific extension/adaptation of core QRE

Basic types: ports, IP addresses, tests of packet fields
Actions on packets: drop, flood, forward, augmentation with fields...
Reference to time windows (e.g. stream of packets in last 5 sec)

Basic functions on packets (written in C) +
QRE combinators (else, split, iter, max, min, sum, average) +
Parameterization (by IP addresses, current time, threshold values ...)

(source IP, dest IP, payload)

Switch

drop / forward to port X / alert controller
Implementation and Evaluation

NetQRE Compiler
  + NetQRE Runtime system (to process packets and update state)

1. Can network policies be expressed in concise and intuitive manner?

2. Is compiled code efficient for throughput and memory footprint?

3. Can our system be used for real-time monitoring and alerting?

Flow-level traffic measurements
  e.g. detection of heavy hitters, super spreaders

TCP state monitoring
  e.g. aggregate statistics of TCP connections, detect SYN flood attack

Application level monitoring
  e.g. collect statistics about VoIP sessions
Monitoring of VoIP Sessions

Detect if current VoIP session is using excessive bandwidth compared to past average.

Modular specification using parameterized expressions:
- Split and Iter constructs
- Aggregation across users
- Aggregation across sessions

18 lines of NetQRE code (vs 100s of lines C++ code).

Session Initiation Protocol
Throughput and Memory Footprint

How does NetQRE generated code compare with hand-crafted code?

Example: Detection of heavy hitters
   (a source IP address has consumed > K bandwidth in past T sec)
Workload: CAIDA traffic trace of ~ 50 million packets

Throughput (million packets per second)
   Manual: 10.50 vs NetQRE: 10.45

Memory: Manual: 14 MB vs NetQRE: 15.1 MB

Summary for other queries (measured for 20 queries)
   Throughput within 4% overhead
   SYN flood attack: NetQRE uses twice as much memory
Real-Time Response

- Experimental setup:
  - Network of two clients and one SDN switch
  - SDN Controller based on POX
  - Network emulated by Mininet with link bandwidth 100 Mbps

- How long does it take to detect an attack and block traffic?
  - Note: correction requires SDN controller to update rules on switch

- Incomplete TCP handshake:
  - SYN packet, followed by matching SYNACK, but no subsequent ACK

- SYN flood attack: Too many incomplete TCP handshakes
SYN Flood Attack

Attack starts

Attack detected and corrected by updating rules in switch
Talk Outline

✓ Motivation

✓ Quantitative Regular Expressions (QRE)

✓ QRE Compilation

✓ Implementation and Evaluation in SDN Platform

.handleClick { Research Opportunities }
Real-time Decision Making

- One research question:
  How to specify quantitative policies over data streams?

- One solution: Quantitative Regular Expressions (QRE)
  Modular high-level specifications
  Theoretically robust expressiveness
  Guaranteed space/time requirements of generated code
  Evaluation for network traffic engineering
Query: Max over CarID  \{ Average speed of CarID in past month \}

How much information about a specific car does answer to this query leak?

Research opportunity:
Anonymity / privacy guarantees for queries over streaming data
Learning ??

What traffic constitutes an attack?

Known patterns can be captured by, say, QREs, but can the switch dynamically learn the attack pattern?

Research opportunity:
Learning high-level declarative patterns, say QREs, more plausible than learning low-level code
Distributed Processing ??

Logical query on a single stream of data
Physical implementation: distributed system

How to ensure consistency ? High performance ? Resilience to errors ?

Emerging architectures: Apache STORM
Safety-critical Applications ??

Clinical diagnosis → pacing stimulus

Specification: logical query over analog signal

⇒ Implementation: discrete control software

Predictable response time critical

Key resource constraint: battery life, so need optimized code

Goal: design more effective diagnosis strategies
Principles of Real-time Decision Making

- Streaming Algorithms
- Programming Languages
- Formal Methods
- Distributed Computing
- Optimal Control Theory
- Smart Control
- Real-Time Systems
- Database Theory
- Security & Privacy