Joltik:
Enabling Energy-Efficient “Future-Proof” Analytics on Low-Power Wide-Area Networks

Mingran Yang, Junbo Zhang, Akshay Gadre, Zaoxing Liu, Swarun Kumar, and Vyas Sekar

Carnegie Mellon University
LPWAN provides opportunity for many applications

√ Long range
√ Low power

Low-power wide-area network (LPWAN) Technologies

Industrial monitoring  Smart city  Smart farming
Generality vs. Power trade-off for LPWAN applications

- Occasional short samples
- Pre-determined summary statistics

Downstream Analytics

- Energy generated
- Power outages
- Weather events

New tasks → additional energy overhead
System design goal 1: generality

Downstream Analytics

- Energy generated
- Power outages
- Weather events
- ...

Support estimation on multiple statistics simultaneously
System design goal 2: high-fidelity

Downstream Analytics
- Energy generated
- Power outages
- Weather events
- ......

Error rate < 5%
System design goal 3: energy-efficiency

Downstream Analytics
- Energy generated
- Power outages
- Weather events

Lifetime > 5 years
## Existing solutions and limitations

<table>
<thead>
<tr>
<th>Approach</th>
<th>Energy-efficiency</th>
<th>High-fidelity</th>
<th>Generality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-sampling</td>
<td>√</td>
<td>X</td>
<td>√</td>
</tr>
<tr>
<td>Lossless compression</td>
<td>X</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Lossy compression</td>
<td>√</td>
<td>X</td>
<td>√</td>
</tr>
<tr>
<td>Sparse recovery</td>
<td>√</td>
<td>√</td>
<td>X</td>
</tr>
<tr>
<td>Data-centric aggregation</td>
<td>√</td>
<td>√</td>
<td>X</td>
</tr>
</tbody>
</table>
# Joltik vs. existing solutions

<table>
<thead>
<tr>
<th>Approach</th>
<th>Energy-efficiency</th>
<th>High-fidelity</th>
<th>Generality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-sampling</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Lossless compression</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Lossy compression</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Sparse recovery</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Data-centric aggregation</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

**Joltik**

- **Universal sketching**
  - Support the estimation on broad class of statistics
  - No requirements on raw data / network topology
Joltik vs. existing solutions
Outline for this talk

• Motivation
• System workflow and challenges
• System design
  • Low memory footprint - reducing memory footprint
  • Low power - reducing communication footprint
  • Low CPU - reducing computation overhead
• Implementation and evaluation
• Conclusions
Outline for this talk

• Motivation
• System workflow and challenges
• System design
  • Low memory footprint - reducing memory footprint
  • Low power - reducing communication footprint
  • Low CPU - reducing computation overhead
• Implementation and evaluation
• Conclusions
Joltik system workflow

Solar sensor → Sensing → Raw sensed data → Practical realization of universal sketching → Sketch summaries → Wireless Communication → Sketch summaries

Pre deployment Configuration

Query:
- Energy generated
- Power outages
- Weather events
... (new statistics)
Background on universal sketching

• Sketching algorithm:
  
  Input Data Stream
  
  [Diagram: Sketches arrow to Sketches]
  
  Query to estimate certain statistics
  
  e.g., count-min sketch serves as a frequency table of events in a data stream

• Universal Sketching:

  Input Data Stream
  
  [Diagram: Universal Sketches arrow to Universal Sketches]
  
  Query to estimate multiple statistics
Why universal sketching is a promising solution?

Universal sketching can estimate **many already known (generality)**, or even **possibly unforeseen (future-proof)** statistics at the same time.

<table>
<thead>
<tr>
<th>Estimation tasks in solar sensor</th>
<th>Statistics supported by universal sketching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy generated</td>
<td>L1-norm</td>
</tr>
<tr>
<td>Power outages</td>
<td>Zero-draw time</td>
</tr>
<tr>
<td>Weather events</td>
<td>Change</td>
</tr>
<tr>
<td>Anomaly detection</td>
<td>Entropy</td>
</tr>
<tr>
<td>Voltage volatility</td>
<td>L2-norm</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

Table: Statistics relevant to solar farm
Data structure in universal sketching algorithm

Input Data Stream → Sub-streams

Layer 1

Layer 2

Layer $\log N$

Sketches

Universal Sketches

Heavy-Hitters

$: 3$, $\bigcirc: 2$

$: 2$, $\bigcirc: 1$

$: 1$
Data structure in universal sketching algorithm

Input Data Stream → Sub-streams

- Layer 1
- Layer 2
- Layer \( \log N \)

Sketch counter arrays → Sketches → Heavy-hitters Heap

Single layer:
- \( \square:3, \square:2 \)
- \( \square:2, \square:1 \)
- \( \square:1 \)
How sensed value updates a single layer

Sensed Value k

Key: k  Value: 6
(median of all updated counters)

Storing Heavy Hitters

`s` counters per array

`d` arrays of counters
How sensed value updates the entire structure
Joltik system challenges

- Challenge 1: Reduce memory footprint of universal sketches

  Native Universal Sketching

  Hundreds of KBs memory

  e.g., 300 KB to achieve 95% accuracy on a real-world dataset

- Limited on-chip memory

  e.g., the sensor board in our testbed only has 128 KB memory
Joltik system challenges

• Challenge 2: Reduce communication footprint
Joltik system challenges

- Challenge 3: Reduce computation overhead of sketch update

Universal sketch operations

<table>
<thead>
<tr>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash computations</td>
</tr>
<tr>
<td>Arithmetic calculations</td>
</tr>
<tr>
<td>Counter updates</td>
</tr>
<tr>
<td>Heap updates</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Outline for this talk

• Motivation
• System workflow and challenges
• System design
  • Low memory footprint - reducing memory footprint
  • Low power - reducing communication footprint
  • Low CPU - reducing computation overhead
• Implementation and evaluation
• Conclusions
Outline for this talk

- Motivation
- System workflow and challenges
- System design
  - Low memory footprint - reducing memory footprint
  - Low power - reducing communication footprint
  - Low CPU - reducing computation overhead
- Implementation and evaluation
- Conclusions
Reducing memory footprint: “Inverted Pyramid”

Gradually reduce number of columns
Insight of “Inverted Pyramid” structure

- Lower layers need to handle much smaller number of samples

\[
\sqrt{H_2(k)} \quad \sqrt{H_3(k)} \quad \sqrt{H_n(k)}
\]

Sensed Value k

Layer 1 100%
Layer 2 50%
Layer 3 25%
Layer n \(\frac{1}{2^{n-1}}\)
Outline for this talk

• Motivation
• System workflow and challenges
• System design
  • Low memory footprint - reducing memory footprint
  • Low power - reducing communication footprint
  • Low CPU - reducing computation overhead
• Implementation and evaluation
• Conclusions
Reducing communication footprint
Observation on sketch counters

- Only small part of counters have large values

A sketch counter example

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Indoor Solar</th>
<th>Sensor Scope</th>
<th>LoRa Farm</th>
<th>Joltik</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required bit size</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>17%</td>
<td>75%</td>
<td>6%</td>
<td>2%</td>
</tr>
<tr>
<td></td>
<td>62%</td>
<td>23%</td>
<td>11%</td>
<td>4%</td>
</tr>
<tr>
<td></td>
<td>43%</td>
<td>41%</td>
<td>16%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>53%</td>
<td>9%</td>
<td>37%</td>
<td>0%</td>
</tr>
</tbody>
</table>

A sketch counter example

<table>
<thead>
<tr>
<th>Counters</th>
<th>5</th>
<th>3</th>
<th>7930</th>
<th>9</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>30</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>184</td>
<td>11</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>2859</td>
<td>3</td>
</tr>
</tbody>
</table>

Only small part of counters have large values.
Lossless encoding of sketch counters

<table>
<thead>
<tr>
<th>Required Bitsize</th>
<th>Prefix</th>
<th>Bitsize before/after compress</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>00</td>
<td>16 / 6</td>
</tr>
<tr>
<td>8</td>
<td>01</td>
<td>16 / 10</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>16 / 14</td>
</tr>
<tr>
<td>16</td>
<td>11</td>
<td>16 / 18</td>
</tr>
</tbody>
</table>

Example:

All counters are originally 16 bits before encoding

Rare case!
Efficiently transmitting heavy-hitters heap

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>910</td>
<td>35552</td>
</tr>
<tr>
<td>804</td>
<td>34409</td>
</tr>
<tr>
<td>983</td>
<td>29374</td>
</tr>
<tr>
<td>905</td>
<td>19395</td>
</tr>
<tr>
<td>827</td>
<td>18890</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

A heavy-hitter heap example

Transmit

Reconstruct at base station
Outline for this talk

• Motivation
• System workflow and challenges
• System design
  • Low memory footprint - reducing memory footprint
  • Low power - reducing communication footprint
  • Low CPU - reducing computation overhead
• Implementation and evaluation
• Conclusions
Bottleneck analysis

- Top two CPU performance bottlenecks:
  (a) hash computations
  (b) counter updates

(a) Hash computations

(b) Counter updates

Sensed Value k

H_1(k)

H_2(k)

H_d(k)

5  7  9  11
3  4  6  +1  10
9 +1 11  8  4
7  9 +1 10  3
8  5  3 12 +1

d arrays of counters

Storing Heavy Hitters

r counters per array
Reduced sketch update

Intuition: less samples in lower layers $\rightarrow$ less collisions $\rightarrow$ more accurate result
End-to-end deployment

- User input:
  (a) Data collection rate - $Rcl$
  (b) Period of transmission - $T$
- Parameter tuned by Joltik:
  (a) Sketch Size - $S$

<table>
<thead>
<tr>
<th>$Rcl$</th>
<th>$T$</th>
<th>$S$</th>
<th>Lifetime</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Hz</td>
<td>1 day</td>
<td>30 KB</td>
<td>2751 days</td>
<td>94.30% ± 3.27%</td>
</tr>
<tr>
<td>10 Hz</td>
<td>1 day</td>
<td>60 KB</td>
<td>1613 days</td>
<td>96.61% ± 2.25%</td>
</tr>
<tr>
<td>10 Hz</td>
<td>1 day</td>
<td>90 KB</td>
<td>1141 days</td>
<td>97.50% ± 1.92%</td>
</tr>
</tbody>
</table>

Table: Joltik deployment example
Outline for this talk

• Motivation
• System workflow and challenges
• System design
  • Low memory footprint - reducing memory footprint
  • Low power - reducing communication footprint
  • Low CPU - reducing computation overhead
• Implementation and evaluation
• Conclusions
Joltik implementation and evaluation setup

- Microcontroller: NUCLEO-L476RG
- Sensor board: X-NUCLEO-IKS01A2
- RF Frontend: SX1276 LoRa Transceiver

Real world testbed in a campus building at CMU
Joltik provides better energy-accuracy trade-off for “future-proof” analytics
Joltik supports multi-task handling and provides generality

AppSet1 = \{\text{Cardinality}\}
AppSet2 = \{\text{Cardinality, Entropy}\}

AppSet3 = \{\text{Cardinality, Entropy, L2}\}
AppSet4 = \{\text{Cardinality, Entropy, L2, HH}\}

(Positive values imply Joltik is worse and vice versa)
Summary

● Goal
  ○ Build a general, accurate, and energy-efficient sensor analytics framework

● Our approach
  ○ Propose a novel architecture by leveraging universal sketching
  ○ Low memory footprint, low power and low CPU realization of universal sketching

● Joltik: Enabling energy-efficient “future-proof” analytics on LPWAN
  ○ Support general and “future-proof” analytics
  ○ Guarantee >5-year sensor battery life and <5% error rate on a range of statistics

https://github.com/Joltik-project/Joltik
Mingran Yang mingrany@andrew.cmu.edu