ALBUS: a Probabilistic Monitoring Algorithm to Counter Burst-Flood Attacks

Simon Scherrer, Adrian Perrig
ETH Zürich

Jo Vliegen, Arish Sateesan, Nele Mentens*
KU Leuven, *Leiden University

Hsu-Chun Hsiao
National Taiwan University

SRDS 2023, Marrakech
How Do DDoS Defense Systems Work?

**Goal:** Guarantee availability of network resources under malicious traffic from many sources
How Do DDoS Defense Systems Work?

**Goal:** Guarantee availability of network resources under malicious traffic from many sources

<table>
<thead>
<tr>
<th>Poseidon</th>
<th>Ripple</th>
<th>Jaqen</th>
<th>COLIBRI</th>
<th>ACC-Turbo</th>
</tr>
</thead>
</table>
How Do DDoS Defense Systems Work?

**Goal:** Guarantee availability of network resources under malicious traffic from many sources

<table>
<thead>
<tr>
<th>System</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poseidon</td>
<td>(Zhang et al. 2020)</td>
</tr>
<tr>
<td>Ripple</td>
<td>(Xing et al. 2021)</td>
</tr>
<tr>
<td>Jaqen</td>
<td>(Liu et al. 2021)</td>
</tr>
<tr>
<td>COLIBRI</td>
<td>(Giuliari et al. 2021)</td>
</tr>
<tr>
<td>ACC-Turbo</td>
<td>(Alcoz et al. 2022)</td>
</tr>
</tbody>
</table>

**DDoS Defense System**
How Do DDoS Defense Systems Work?

**Goal:** Guarantee availability of network resources under malicious traffic from many sources

<table>
<thead>
<tr>
<th>Poseidon</th>
<th>Ripple</th>
<th>Jaqen</th>
<th>COLIBRI</th>
<th>ACC-Turbo</th>
</tr>
</thead>
</table>

DDoS Defense System

Traffic Packet = (Flow, Size, Time)
How Do DDoS Defense Systems Work?

**Goal:** Guarantee availability of network resources under malicious traffic from many sources

- **Poseidon** (Zhang et al. 2020)
- **Ripple** (Xing et al. 2021)
- **Jaqen** (Liu et al. 2021)
- **COLIBRI** (Giuliari et al. 2021)
- **ACC-Turbo** (Alcoz et al. 2022)

**Traffic**

Packet = (Flow, Size, Time)

**DDoS Defense System**

Detector

Identify suspiciously large flows

Mitigation

Restrict suspiciously large flows

Network operator

Configuration, Programming

Filtered Traffic
How Do DDoS Defense Systems Work?

**Goal:** Guarantee availability of network resources under malicious traffic from many sources

- **Poseidon** (Zhang et al. 2020)
- **Ripple** (Xing et al. 2021)
- **Jaqen** (Liu et al. 2021)
- **COLIBRI** (Giuliari et al. 2021)
- **ACC-Turbo** (Alcoz et al. 2022)

Traffic Packet = (Flow, Size, Time)

**DDoS Defense System**

- **Detection**
  - Identify suspiciously large flows

Filtered Traffic
How Do DDoS Defense Systems Work?

**Goal:** Guarantee availability of network resources under malicious traffic from many sources

- **Poseidon** (Zhang et al. 2020)
- **Ripple** (Xing et al. 2021)
- **Jaqen** (Liu et al. 2021)
- **COLIBRI** (Giuliari et al. 2021)
- **ACC-Turbo** (Alcoz et al. 2022)

Traffic Packet = (Flow, Size, Time)

**DDoS Defense System**

**Detection**
Identify suspiciously large flows

**Mitigation**
Restrict suspiciously large flows

Filtered Traffic
How Do DDoS Defense Systems Work?

**Goal:** Guarantee availability of network resources under malicious traffic from many sources

<table>
<thead>
<tr>
<th>System</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poseidon</td>
<td>(Zhang et al. 2020)</td>
</tr>
<tr>
<td>Ripple</td>
<td>(Xing et al. 2021)</td>
</tr>
<tr>
<td>Jaqen</td>
<td>(Liu et al. 2021)</td>
</tr>
<tr>
<td>COLIBRI</td>
<td>(Giuliari et al. 2021)</td>
</tr>
<tr>
<td>ACC-Turbo</td>
<td>(Alcoz et al. 2022)</td>
</tr>
</tbody>
</table>

**DDoS Defense System**

- **Detection**
  - Identify suspiciously large flows

- **Mitigation**
  - Restrict suspiciously large flows

**Traffic Packet** = (Flow, Size, Time)

**Configuration, Programming**

**Network operator**

**Filtered Traffic**
How Do DDoS Defense Systems Work?

**Goal:** Guarantee availability of network resources under malicious traffic from many sources

- **Poseidon** (Zhang et al. 2020)
- **Ripple** (Xing et al. 2021)
- **Jaqen** (Liu et al. 2021)
- **COLIBRI** (Giuliari et al. 2021)
- **ACC-Turbo** (Alcoz et al. 2022)

**DDoS Defense System**

- **Detection**
  - Identify suspiciously large flows

- **Mitigation**
  - Restrict suspiciously large flows

**Traffic Packet** = (Flow, Size, Time)

**Network operator**

**Configuration, Programming**

**Filtered Traffic**
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection.
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

+ Limited memory  
+ Efficient processing  
- Limited accuracy

For each flow:

- Estimated flow volume in period so far
- Expected flow volume over period

If expected flow volume over period > Link capacity:

- Suspicious flow!

**Elephant Flow:**

- Detected

**Burst-Flood Attack:**

- Medium-rate bursts in different flows
- Small volume of each flow
- Bursts split by resets
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

- Limited memory
- Efficient processing
- Limited accuracy

For each flow:

- Estimated flow volume in period so far
- Expected flow volume over period

If

\[
\text{Flow sending rate} \geq \text{Link capacity} \\
\text{Elephant Flow:} \quad \text{Detected}
\]

\[
\text{Burst-Flood Attack:} \quad \text{Medium-rate bursts in different flows} \\
\text{Small volume of each flow} \\
\text{Bursts split by resets}
\]
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection

+ Limited memory  
+ Efficient processing  
- Limited accuracy

For each flow:

- Estimated flow volume in period so far
- Expected flow volume over period

> Suspicious flow!

**Burst-Flood Attack**:

- Medium-rate bursts in different flows
- Small volume of each flow
- Bursts split by resets

**Network Security Group**

Department of Computer Science
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

- Limited memory
- Efficient processing
- Limited accuracy

For each flow:

- Estimated flow volume in period so far
- Expected flow volume over period

If:

- Expected flow volume > Link capacity

Then:

- Burst-Flood Attack:
  - Medium-rate bursts in different flows
  - Small volume of each flow
  - Bursts split by resets

reset period

Network Security Group
Department of Computer Science
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

- Limited memory
- Efficient processing
- Limited accuracy

For each flow:

- Estimated flow volume in period so far
- Expected flow volume over period

**Suspicious flow!**

**Network Security Group**

**Department of Computer Science**
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection

+ Limited memory  + Efficient processing  - Limited accuracy

For each flow:

- Estimated flow volume in period so far
- Expected flow volume over period

**Suspicious flow!**

- **Elephant Flow:** Detected
- **Burst-Flood Attack:** Medium-rate bursts in different flows
- Small volume of each flow
- Bursts split by resets

**Network Security Group**
**Department of Computer Science**
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection

- Limited memory
- Efficient processing
- Limited accuracy

**For each flow:**

- **Estimated** flow volume in period so far
- **Expected** flow volume over period

**Time**

- Reset
- Reset period

Network Security Group
Department of Computer Science
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection

- Limited memory
- Efficient processing
- Limited accuracy

For each flow:

- Estimated flow volume in period so far
- Expected flow volume over period

Suspicious flow!

Network Security Group
Department of Computer Science
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

- **Limited memory**
- **Efficient processing**
- **Limited accuracy**

For each flow:

**Estimated** flow volume in period so far

**>**

**Expected** flow volume over period

**Suspicious flow!**
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

+ Limited memory
+ Efficient processing
- Limited accuracy

**For each flow:**

- **Estimated** flow volume in period so far
- **Expected** flow volume over period

If

<table>
<thead>
<tr>
<th>Estimated</th>
<th>Expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow volume</td>
<td>flow volume</td>
</tr>
<tr>
<td>in period so far</td>
<td>over period</td>
</tr>
</tbody>
</table>

...Suspicious flow!

**Burst-Flood Attack:**
- Medium-rate bursts in different flows
- Small volume of each flow
- Bursts split by resets

**Elephant Flow:** Detected

**Network Security Group**
**Department of Computer Science**
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

- **CountMin-Sketch**: Limited memory, Efficient processing, Limited accuracy
- **Count-Sketch**: Limited memory, Efficient processing, Limited accuracy

For each flow:

- Estimated flow volume in period so far
- Expected flow volume over period

Suspicous flow!

**Network Security Group**
Department of Computer Science
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

- **Limited memory**
- **Efficient processing**
- **Limited accuracy**

For each flow:

- *Estimated* flow volume in period so far
- *Expected* flow volume over period

If *Estimated flow volume* > *Expected flow volume*

*Suspicious flow!*

**Network Security Group**
Department of Computer Science
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

- **CountMin-Sketch** and **Count-Sketch**
  - Limited memory
  - Efficient processing
  - Limited accuracy

For each flow:

- **Estimated** flow volume in period so far
- **Expected** flow volume over period

**Suspicious flow!**

**Burst-Flood Attack:**
Medium-rate bursts in different flows
Small volume of each flow

Network Security Group
Department of Computer Science
How Do DDoS Defense Systems Detect Suspicious Flows?

DDoS defense systems mostly use the algorithms **CountMin-Sketch** and **Count-Sketch** for detection:

- Limited memory
- Efficient processing
- Limited accuracy

**For each flow:**

- **Estimated** flow volume in period so far
- **Expected** flow volume over period

**Suspicious flow!**

**Burst-Flood Attack:**
Medium-rate bursts in different flows
Small volume of each flow
Bursts split by resets
How Do Sketch Algorithms Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed
How Do Sketch Algorithms Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed

Recall = \( \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \)

Precision = \( \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \)
How Do Sketch Algorithms Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed

Recall = \( \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \)

Precision = \( \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \)

Trade-off between recall and precision

Reset period must fit burst width ⇒ Evasion

Network Security Group
Department of Computer Science
How Do Sketch Algorithms Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed

Recall = \[ \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \]

Precision = \[ \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \]

Trade-off between recall and precision

Reset period must fit burst width ⇒ Evasion
How Do Sketch Algorithms Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed

Recall = \( \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \)

Recall

100%

Precision = \( \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \)

Precision

100%

Trade-off between recall and precision

Reset period must fit burst width

⇒ Evasion
How Do Sketch Algorithms Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed

Recall = \( \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \)

Recall

![Graph showing recall and precision](image)

Report if:

\( \text{Estimated flow volume} > \text{Expected flow volume} \) in period so far over period

Precision = \( \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \)

Precision

![Graph showing recall and precision](image)

Trade-off between recall and precision

Reset period must fit burst width \( \Rightarrow \) Evasion

Network Security Group

Department of Computer Science

ETH Zürich

3/19
How Do Sketch Algorithms Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed

Recall = \( \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \)

Precision = \( \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \)

Report if:

- Estimated flow volume in period so far > 50% of Expected flow volume over period

**Trade-off** between recall and precision

Reset period must fit burst width \( \Rightarrow \) Evasion

Network Security Group
Department of Computer Science

ETH Zürich
How Do Sketch Algorithms Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed

\[
\text{Recall} = \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}}
\]

\[
\text{Precision} = \frac{\text{Reported malicious bursts}}{\text{Reported bursts}}
\]

- **Recall** vs. Reset period [seconds]
  - CountMin-Sketch (Threshold 50%)
  - 100%

- **Precision** vs. Reset period [seconds]
  - CountMin-Sketch (Threshold 50%)
  - 100%
**How Do Sketch Algorithms Perform Under Burst-Flood Attacks?**

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed

<table>
<thead>
<tr>
<th>Recall</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Recall} = \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} )</td>
<td>( \text{Precision} = \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} )</td>
</tr>
</tbody>
</table>

Trade-off between recall and precision:
- Reset period must fit burst width

---

**Network Security Group**
**Department of Computer Science**
How Do Sketch Algorithms Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed

Recall = \( \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \)

Precision = \( \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \)

**Trade-off between recall and precision**

Reset period must fit burst width \( \Rightarrow \) Evasion
How Can We Better Detect Malicious Bursts?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:
How Can We Better Detect Malicious Bursts?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:

- Time-window flexibility
- Memory efficiency
- Processing efficiency
How Can We Better Detect Malicious Bursts?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:

- **Time-window flexibility**
- **Memory efficiency**
How Can We Better Detect Malicious Bursts?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:

- **Time-window flexibility**
- **Memory efficiency**
- **Processing efficiency**
How Can We Better Detect Malicious Bursts?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:

- **Time-window flexibility**
- **Memory efficiency**
- **Processing efficiency**

Techniques such as CountMin-Sketch and Count-Sketch can be used to achieve these requirements.
How Can We Better Detect Malicious Bursts?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:

- **Time-window flexibility**
- **Memory efficiency**
- **Processing efficiency**

CountMin-Sketch

Count-Sketch

ALBUS
How Does ALBUS Work?

**ALBUS**: Adaptive Leaky-Bucket Undulation Sensor
How Does ALBUS Work?

**ALBUS**: Adaptive Leaky-Bucket Undulation Sensor

Flows: $f_1$ $f_2$ $f_3$ $f_4$ $f_5$ $f_6$ $f_7$ $f_8$ ... $f_{1475286}$
How Does ALBUS Work?

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

Flows: $f_1$  $f_2$  $f_3$  $f_4$  $f_5$  $f_6$  $f_7$  $f_8$  ...  $f_{1475286}$
How Does ALBUS Work?

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

Flows: \( f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7 \quad f_8 \quad \ldots \quad f_{1475286} \)

Diagram:
- Leaky-bucket module
- Background-counting module
How Does the Leaky-Bucket Algorithm Work?

General idea:

Bucket volume $\beta$

Leakage rate $\gamma$

Fill rate from monitored traffic flow

Net inflow = Inflow − Outflow × Overflow
How Does the Leaky-Bucket Algorithm Work?

General idea:
How Does the Leaky-Bucket Algorithm Work?

General idea:

Network Security Group
Department of Computer Science
How Does the Leaky-Bucket Algorithm Work?

General idea:

- Bucket volume \( \beta \)
- Leakage rate \( \gamma \)

Net inflow = Inflow - Outflow × Overflow
How Does the Leaky-Bucket Algorithm Work?

General idea:

Net inflow = Inflow - Outflow (Overflow)

Bucket volume $\beta$

Leakage rate $\gamma$

Fill rate from monitored traffic flow
How Does the Leaky-Bucket Algorithm Work?

General idea:

Net inflow = Inflow − Outflow

Bucket volume $\beta$

Leakage rate $\gamma$

Fill rate from monitored traffic flow
How Does the Leaky-Bucket Algorithm Work?

General idea:

Net inflow = Inflow − Outflow

Bucket volume $\beta$

Fill rate from monitored traffic flow

Leakage rate $\gamma$

Net inflow = Inflow − Outflow
How Does the Leaky-Bucket Algorithm Work?

General idea:

Net inflow = Inflow - Outflow

Bucket volume $\beta$

Leakage rate $\gamma$

Fill rate from monitored traffic flow
How Does the Leaky-Bucket Algorithm Work?

General idea:

Net inflow = Inflow − Outflow

Bucket volume \( \beta \)

Leakage rate \( \gamma \)

Fill rate from monitored traffic flow

\( \times \) Overflow
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than $\gamma w + \beta$ during a time window with arbitrary width $w$. Here, $\gamma$ is the leakage rate, $\beta$ is the flow base rate, $\gamma w$ is the time-window flexibility, and $\beta > \gamma w$. But:

- A leaky bucket can only monitor a single flow at a time!
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than $\gamma w + \beta$ during a time window with arbitrary width $w$. 

But: A leaky bucket can only monitor a single flow at a time!
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than \( \gamma w + \beta \) during a time window with arbitrary width \( w \).

\[ \gamma = \text{Leakage rate} \quad \beta = \text{Flow base rate} \]
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than \( \gamma w + \beta \) during a time window with arbitrary width \( w \)

- \( \gamma \) = Leakage rate = Flow base rate
- \( \beta \) = Bucket volume = Burstiness allowance
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than $\gamma w + \beta$ during a time window with arbitrary width $w$

- $\gamma = \text{Leakage rate}$
- $= \text{Flow base rate}$

- $\beta = \text{Bucket volume}$
- $= \text{Burstiness allowance}$
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than \( \gamma w + \beta \) during a time window with arbitrary width \( w \).

- \( \gamma \) = Leakage rate = Flow base rate
- \( \beta \) = Bucket volume = Burstiness allowance

But:

A leaky bucket can only monitor a single flow at a time!
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than $\gamma w + \beta$ during a time window with arbitrary width $w$.

- $\gamma$ = Leakage rate
- $\beta$ = Bucket volume

Send rate

\[\gamma \leq \frac{\gamma w}{w_1} \leq \gamma w + \beta \leq \frac{\gamma w}{w_3} \leq \beta\]

But:

- A leaky bucket can only monitor a single flow at a time!
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than $\gamma w + \beta$ during a time window with arbitrary width $w$.

- $\gamma$ = Leakage rate
- $\beta$ = Bucket volume
- $\alpha$ = Flow base rate
- $\beta$ = Burstiness allowance

---

**Diagram:**

- Send rate
- Time

- $\gamma$ = Leakage rate
- $\gamma w_1$ = Burstiness allowance
- $\gamma w_2$ = Time-window flexibility
- $\gamma w_3$ = Memory efficiency

---

But: A leaky bucket can only monitor a single flow at a time!
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than \( \gamma w + \beta \) during a time window with arbitrary width \( w \)

- \( \gamma \) = Leakage rate = Flow base rate
- \( \beta \) = Bucket volume = Burstiness allowance

\[ \frac{\gamma}{w_1} > \beta \]
\[ \frac{\gamma}{w_2} < \beta \]
\[ \frac{\gamma}{w_3} > \beta \]
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than $\gamma w + \beta$ during a time window with arbitrary width $w$

- $\gamma$ = Leakage rate
- $\beta$ = Bucket volume

But: A leaky bucket can only monitor a single flow at a time!
What Properties Are Offered by the Leaky-Bucket Algorithm?

A leaky bucket reports a flow that sends more than $\gamma w + \beta$ during a time window with arbitrary width $w$.

- $\gamma = \text{Leakage rate}$
- $= \text{Flow base rate}$
- $\beta = \text{Bucket volume}$
- $= \text{Burstiness allowance}$

**But:** A leaky bucket can only monitor a single flow at a time!
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?

Leaky-bucket counters:

<table>
<thead>
<tr>
<th>LB 1</th>
<th>LB 2</th>
<th>LB 3</th>
<th>LB 4</th>
<th>LB 256</th>
</tr>
</thead>
</table>

Eviction criterion: If LB net inflow turns negative, send rate.

Report and evict f8 on overflow!
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?

Flows: \( f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7 \quad f_8 \quad \ldots \quad f_{1475286} \)

<table>
<thead>
<tr>
<th>LB 1</th>
<th>LB 2</th>
<th>LB 3</th>
<th>LB 4</th>
<th>LB 256</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Leaky-bucket counter 1" /></td>
<td><img src="image2.png" alt="Leaky-bucket counter 2" /></td>
<td><img src="image3.png" alt="Leaky-bucket counter 3" /></td>
<td><img src="image4.png" alt="Leaky-bucket counter 4" /></td>
<td><img src="image5.png" alt="Leaky-bucket counter 256" /></td>
</tr>
</tbody>
</table>

Leaky-bucket counters:

Hashing:

Monitored flows:

\( f_4 \quad f_7 \quad f_5 \quad \ldots \quad f_8 \quad f_3 \quad \times \quad \) Report and evict \( f_8 \) on overflow!

Eviction criterion:

If LB net inflow turns negative
Send rate
Time \( \gamma \)
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?

Flows: $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, \ldots, f_{1475286}$

Hashing:

Leaky-bucket counters:
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?

Flows: $f_1$ $f_2$ $f_3$ $f_4$ $f_5$ $f_6$ $f_7$ $f_8$ $\ldots$ $f_{1475286}$

Hashing:

Leaky-bucket counters:

Monitored flows:

- LB 1
- LB 2
- LB 3
- LB 4
- LB 256
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?

Leaky-bucket counters:

Flows:

Hashing:

Monitored flows:
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?

Flows: $f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7 \quad f_8 \quad \ldots \quad f_{1475286}$

Hashing:

Leaky-bucket counters:

Monitored flows:

- LB 1
- LB 2
- LB 3
- LB 4
- LB 256

Eviction criterion: If LB net inflow turns negative, send rate Time $\gamma$.
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?

Flows: $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, \ldots, f_{1475286}$

Hashing:

Leaky-bucket counters:

Monitored flows:

Report and evict $f_8$ on overflow!
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?

Flows: $f_1$, $f_2$, $f_3$, $f_4$, $f_5$, $f_6$, $f_7$, $f_8$, ..., $f_{1475286}$

Hashing: $f_4$, $f_3$, $f_7$, $f_5$, ..., $f_8$

Leaky-bucket counters:

<table>
<thead>
<tr>
<th>LB 1</th>
<th>LB 2</th>
<th>LB 3</th>
<th>LB 4</th>
<th>LB 256</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_4$</td>
<td>$f_3$</td>
<td>$f_7$</td>
<td>$f_5$</td>
<td>$f_8$</td>
</tr>
</tbody>
</table>

Monitored flows: If LB net inflow turns negative

Eviction criterion: Report and evict $f_8$ on overflow!
How Can We Apply the LB Algorithm in a Memory-Efficient Fashion?

Flows: \( f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, \ldots, f_{1475286} \)

Hashing:

Leaky-bucket counters:

Monitored flows:

Eviction criterion:

Send rate

If LB net inflow turns negative

Report and evict \( f_8 \) on overflow!

Send rate \( \gamma \)
How Does ALBUS Work?

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

Flows: $f_1$ $f_2$ $f_3$ $f_4$ $f_5$ $f_6$ $f_7$ $f_8$ ... $f_{1475286}$
How Does ALBUS Work?

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

Flows: $f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7 \quad f_8 \quad \cdots \quad f_{1475286}$

Hashing:

Leaky-bucket counters:

<table>
<thead>
<tr>
<th>LB 1</th>
<th>LB 2</th>
<th>LB 3</th>
<th>LB 4</th>
<th>\cdots</th>
<th>LB 256</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_4$</td>
<td>$f_3$</td>
<td>$f_7$</td>
<td>$f_5$</td>
<td>$\cdots$</td>
<td>$f_8$</td>
</tr>
</tbody>
</table>

Background-counting module
How Does ALBUS Work?

ALBUS: Adaptive Leaky-Bucket Undulation Sensor

Flows: $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, \ldots, f_{1475286}$

Hashing:

Leaky-bucket counters:

Overflow: Report & Evict

Net inflow $< 0$: Evict

On evict: ?

Background-counting module
**How Does ALBUS Work?**

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

**Flows:** $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, \ldots, f_{1475286}$

**Hashing:**

**Leaky-bucket counters:**

<table>
<thead>
<tr>
<th>LB 1</th>
<th>LB 2</th>
<th>LB 3</th>
<th>LB 4</th>
<th>\ldots</th>
<th>LB 256</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_4$</td>
<td>$f_3$</td>
<td>$f_7$</td>
<td>$f_5$</td>
<td>\ldots</td>
<td>$f_8$</td>
</tr>
</tbody>
</table>

**Overflow:** Report & Evict

**Net inflow** $< 0$: Evict

**Background-counting module**
How Does ALBUS Work?

**ALBUS**: Adaptive Leaky-Bucket Undulation Sensor

**Flows**: $f_1 \ f_2 \ f_3 \ f_4 \ f_5 \ f_6 \ f_7 \ f_8 \ \ldots \ \ f_{1475286}$

**Hashing**: 

**Leaky-bucket counters**: 

<table>
<thead>
<tr>
<th>LB 1</th>
<th>LB 2</th>
<th>LB 3</th>
<th>LB 4</th>
<th>$\ldots$</th>
<th>LB 256</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_4$</td>
<td>$f_3$</td>
<td>$f_7$</td>
<td>$f_5$</td>
<td>$\ldots$</td>
<td>$f_8$</td>
</tr>
</tbody>
</table>

**Overflow**: Report & Evict

**Net inflow $< 0$**: Evict

**On evict**: ?

Background-counting module

Network Security Group
Department of Computer Science
How Does ALBUS Work?

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

Flows: $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, \ldots, f_{1475286}$

Hashing:

Leaky-bucket counters:

<table>
<thead>
<tr>
<th>LB 1</th>
<th>LB 2</th>
<th>LB 3</th>
<th>LB 4</th>
<th>\ldots</th>
<th>LB 256</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_4$</td>
<td>$f_3$</td>
<td>$f_7$</td>
<td>$f_5$</td>
<td>\ldots</td>
<td>$f_8$</td>
</tr>
</tbody>
</table>

Overflow: Report & Evict

Net inflow $< 0$: Evict

On evict: ?

Background-counting module
What to Do When a Flow Is Evicted?
What to Do When a Flow Is Evicted?

Leaky-bucket counter

Question: How to find the most bursty flow among the background flows of a leaky-bucket counter?

Packet:

Flow: \( f_2 \)

Size: \( s_2 \)

Count: \( c + s_2 \)

Flow: \( f_3 \)

Size: \( s_3 \)

Probability: \( p \)

else

Count: \( c - s_3 \)

Flow: \( f_2 \)

else

Count: \( c - s_3 \)

Flow: \( f_3 \)

Effective at finding largest background flow!
What to Do When a Flow Is Evicted?

Leaky-bucket counter

Leaky-bucket counter

$\text{LB } n$

Evicted

$\text{Evicted}$
What to Do When a Flow Is Evicted?

Leaky-bucket counter

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?
What to Do When a Flow Is Evicted?

Leaky-bucket counter

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?
What to Do When a Flow Is Evicted?

Leaky-bucket counter

Background counter

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?
What to Do When a Flow Is Evicted?

**Question:**
How to find the most bursty flow among the background flows of a leaky-bucket counter?

**Probabilistic decay** (Yang et al. 2019)
What to Do When a Flow Is Evicted?

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?

Probabilistic decay (Yang et al. 2019)
Packet: Flow $f_2$, Size $s_2$
What to Do When a Flow Is Evicted?

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?

Probabilistic decay (Yang et al. 2019)

Packet: Flow $f_2$, Size $s_2$

Count: $c + s_2$

Flow: $f_2$
What to Do When a Flow Is Evicted?

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?

Probabilistic decay (Yang et al. 2019)
Packet: Flow $f_2$, Size $s_2$
Packet: Flow $f_3$, Size $s_3$

Count: $c + s_2$
Flow: $f_2$

Effective at finding largest background flow!
What to Do When a Flow Is Evicted?

Leaky-bucket counter

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?

Probabilistic decay (Yang et al. 2019)

Packet: Flow $f_2$, Size $s_2$

Packet: Flow $f_3$, Size $s_3$

Probability $p$

Count: $c - s_3$

Flow: $f_2$

Count: $c + s_2$

Flow: $f_2$

Background counter
What to Do When a Flow Is Evicted?

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?

Probabilistic decay (Yang et al. 2019)

Packet: Flow $f_2$, Size $s_2$
Count: $c + s_2$
Flow: $f_2$

Packet: Flow $f_3$, Size $s_3$
Probability $p$
Count: $c - s_3$
else
Count: $c$
Flow: $f_2$
What to Do When a Flow Is Evicted?

Leaky-bucket counter

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?

Probabilistic decay (Yang et al. 2019)

Packet: Flow \( f_2 \), Size \( s_2 \)

Count: \( c + s_2 \)
Flow: \( f_2 \)

Packet: Flow \( f_3 \), Size \( s_3 \)

Probability \( p \)
Count: \( c - s_3 \)
Flow: \( f_3 \)

else
Count: \( c \)
Flow: \( f_2 \)

Effective at finding largest background flow!
What to Do When a Flow Is Evicted?

**Question:**
How to find the most bursty flow among the background flows of a leaky-bucket counter?

**Probabilistic decay** (Yang et al. 2019)

- **Packet:** Flow $f_2$, Size $s_2$
  - **Count:** $c + s_2$
  - **Flow:** $f_2$

- **Packet:** Flow $f_3$, Size $s_3$
  - Probability $p$
  - **Count:** $c - s_3$
  - **Flow:** $f_3$
  - else
  - **Count:** $c - s_3 ≤ 0$
  - **Flow:** $f_3$
  - else
  - **Count:** $c$
  - **Flow:** $f_2$

Effective at finding largest background flow!
What to Do When a Flow Is Evicted?

Leaky-bucket counter

Background counter

Probabilistic decay (Yang et al. 2019)

Packet: Flow \( f_2 \), Size \( s_2 \)

Count: \( c + s_2 \)
Flow: \( f_2 \)

Packet: Flow \( f_3 \), Size \( s_3 \)

Probability \( p \)

Count: \( c - s_3 \)\( \leq 0 \) else

Count: \( c - s_3 \)
Flow: \( f_2 \)

Count: \( c \)
Flow: \( f_2 \)

Question:
How to find the most bursty flow among the background flows of a leaky-bucket counter?

Effective at finding largest background flow!
What to Do When a Flow Is Evicted?

Leaky-bucket counter

Background counter

**Question:** How to find the most bursty flow among the background flows of a leaky-bucket counter?

**Probabilistic decay** (Yang et al. 2019)

Packet: Flow \( f_2 \), Size \( s_2 \)

Packet: Flow \( f_3 \), Size \( s_3 \)

Probability \( p \)

Count: \( c + s_2 \)  
Flow: \( f_2 \)

Count: \( s_3 \)  
Flow: \( f_3 \)

Count: \( c - s_3 \)  
Flow: \( f_2 \)

Count: \( c \)  
Flow: \( f_2 \)

Effective at finding largest background flow!
What to Do When a Flow Is Evicted?

**Question:** How to find the most bursty flow among the background flows of a leaky-bucket counter?

**Proportional decay** (Yang et al. 2019)

- **Packet:** Flow $f_2$, Size $s_2$
  - **Count:** $c + s_2$
  - **Flow:** $f_2$
  - Probability $p$
  - Else $c - s_3 \leq 0$

- **Packet:** Flow $f_3$, Size $s_3$
  - **Count:** $c - s_3$
  - **Flow:** $f_3$
  - Else $c - s_3 > 0$
  - Else $c$

Effective at finding largest background flow!
# How Does ALBUS Work?

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

**Flows:** $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, \ldots, f_{1475286}$

**Hashing:**

<table>
<thead>
<tr>
<th>Leaky-bucket counters:</th>
<th>LB 1</th>
<th>LB 2</th>
<th>LB 3</th>
<th>LB 4</th>
<th>\ldots</th>
<th>LB 256</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$f_4$</td>
<td>$f_3$</td>
<td>$f_7$</td>
<td>$f_5$</td>
<td>\ldots</td>
<td>$f_8$</td>
</tr>
</tbody>
</table>

**Overflow:** Report & Evict

- **Net inflow** $< 0$: Evict
- **On evict:** ?

**Background-counting module**
How Does ALBUS Work?

**ALBUS**: Adaptive Leaky-Bucket Undulation Sensor

**Flows:**
- $f_1$
- $f_2$
- $f_3$
- $f_4$
- $f_5$
- $f_6$
- $f_7$
- $f_8$
- $f_{1475286}$

**Hashing:**

**Leaky-bucket counters:**
- LB 1
- LB 2
- LB 3
- LB 4
- ... (to LB 256)

**Background counters:**
- BC 1
- BC 2
- BC 3
- BC 4
- ... (to BC 256)

**Overflow:**
- Report & Evict

**Net inflow < 0:**
- Evict

**On evict:**
- Pull from BC

*Network Security Group*
*Department of Computer Science*
How Does ALBUS Work?

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

**Flows:**
- $f_1$
- $f_2$
- $f_3$
- $f_4$
- $f_5$
- $f_6$
- $f_7$
- $f_8$
- $\ldots$
- $f_{1475286}$

**Hashing:**

**Leaky-bucket counters:**
- LB 1
- LB 2
- LB 3
- LB 4
- $\ldots$
- LB 256

**Background counters:**
- BC 1
- BC 2
- BC 3
- BC 4
- $\ldots$
- BC 256

**Overflow:**
- Report & Evict

**Net inflow** $< 0$:
- Evict

**On evict:**
- Pull from BC

**Filter:**
- Probabilistic decay

**Filter:**
- Probabilistic decay
How Does ALBUS Work?

**ALBUS**: Adaptive Leaky-Bucket Undulation Sensor

Flows: \( f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8 \ldots f_{1475286} \)

Hashing:

Leaky-bucket counters:

- LB 1
- LB 2
- LB 3
- LB 4
- \ldots
- LB 256

- \( f_4 \)
- \( f_3 \)
- \( f_7 \)
- \( f_5 \)
- \ldots
- \( f_8 \)

Background counters:

- BC 1
- BC 2
- BC 3
- BC 4
- \ldots
- BC 256

- Count \( c \)
- Count \( c \)
- Count \( c \)
- Count \( c \)
- \ldots
- Count \( c \)

- Flow \( f_2 \)
- Flow \( f_6 \)
- Flow \( f_1 \)
- Flow \( \Box \)
- \ldots
- Flow \( \Box \)

Overflow:
- Report & Evict

Net inflow < 0:
- Evict

On evict:
- Pull from BC

Filter:
- Probabilistic decay

Additional benefits?
Do Background Counters Have Additional Benefits?

Background counters prevent **flow masking**
Do Background Counters Have Additional Benefits?

Background counters prevent **flow masking**
Do Background Counters Have Additional Benefits?

Background counters prevent flow masking

![Diagram showing flow send rate over time with flow masking](image)
Do Background Counters Have Additional Benefits?

Background counters prevent flow masking

\[ f_1 \text{ masks } f_2 \]
Do Background Counters Have Additional Benefits?

Background counters prevent flow masking

![Diagram showing flow send rate over time with background counters](image)

- $f_1$ and $f_2$
- LB n
- BC n
- Count $c$
- Flow

Flow send rate vs Time

$\gamma$ (Base rate)
Do Background Counters Have Additional Benefits?

Background counters prevent **flow masking**

![Diagram showing flow masking](image)
Do Background Counters Have Additional Benefits?

Background counters prevent flow masking

Flow send rate

Time

γ (Base rate)
Do Background Counters Have Additional Benefits?

Background counters prevent **flow masking**

- $f_1$
- $f_2$
- LB n
- BC n
- Count $c$
- Flow $f_2$

Flow send rate

- $f_1$
- $f_2$
- $\gamma$ (Base rate)

Time
Do Background Counters Have Additional Benefits?

Background counters prevent **flow masking**

- $f_1$ masks $f_2$
- $f_2$ send rate
- Time

Network Security Group
Department of Computer Science
Do Background Counters Have Additional Benefits?

Background counters prevent flow masking

**Flow send rate**

- $f_1$
- $f_2$

- $\gamma$ (Base rate)

- $f_1$ masks $f_2$

- $f_1$ will be high!

**Check** $c > $Threshold

**Network Security Group**

**Department of Computer Science**
Do Background Counters Have Additional Benefits?

Background counters prevent flow masking

\[ f_1 \text{ masks } f_2 \]

Flow send rate

Time

\[ \gamma \text{ (Base rate)} \]

Network Security Group
Department of Computer Science
Do Background Counters Have Additional Benefits?

Background counters prevent flow masking

\[ f_1 \text{ masks } f_2 \]

\[ \text{Time} \]

\[ \gamma \text{ (Base rate)} \]

\[ f_1 \]

\[ f_2 \]

Flow send rate

Push

\[ \text{will be high!} \]

Check \( c > \text{Threshold} \)
How Does ALBUS Work?

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

**Flows:** $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, \ldots, f_{1475286}$

**Hashing:**

**Leaky-bucket counters:**

**Background counters:**

**Overflow:** Report & Evict

**Net inflow** $< 0$: Evict

**On evict:** Pull from BC

**Filter:** Probabilistic decay
How Does ALBUS Work?

**ALBUS:** Adaptive Leaky-Bucket Undulation Sensor

---

**Flows:**

<table>
<thead>
<tr>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>...</th>
<th>$f_{1475286}$</th>
</tr>
</thead>
</table>

**Hashing:**

**Leaky-bucket counters:**

<table>
<thead>
<tr>
<th>LB 1</th>
<th>LB 2</th>
<th>LB 3</th>
<th>LB 4</th>
<th>...</th>
<th>LB 256</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_4$</td>
<td>$f_3$</td>
<td>$f_7$</td>
<td>$f_5$</td>
<td>...</td>
<td>$f_8$</td>
</tr>
</tbody>
</table>

**Background counters:**

<table>
<thead>
<tr>
<th>BC 1</th>
<th>BC 2</th>
<th>BC 3</th>
<th>BC 4</th>
<th>...</th>
<th>BC 256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count $c$</td>
<td>Count $c$</td>
<td>Count $c$</td>
<td>Count $c$</td>
<td>...</td>
<td>Count $c$</td>
</tr>
<tr>
<td>Flow $f_2$</td>
<td>Flow $f_6$</td>
<td>Flow $f_1$</td>
<td>Flow □</td>
<td>...</td>
<td>Flow □</td>
</tr>
</tbody>
</table>

**Overflow:**

Report & Evict

**Net inflow $< 0$:**

Evict

**On evict:**

Pull from BC

**Filter:**

Probabilistic decay

**High count:**

Push to LB

---

Network Security Group

Department of Computer Science

ETH Zürich

13/19
How Does ALBUS Perform Under Burst-Flood Attacks?

Scenario:
Malicious bursts last for 200 milliseconds and are 20% larger than allowed

Recall = Reported malicious bursts
Malicious bursts

<table>
<thead>
<tr>
<th>CountMin-Sketch</th>
<th>CountMin-Sketch (Threshold 50%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset period [seconds]</td>
<td>Recalls</td>
</tr>
<tr>
<td>0.1</td>
<td>0.8</td>
</tr>
<tr>
<td>CountMin-Sketch</td>
<td>Perfect precision by design!</td>
</tr>
</tbody>
</table>

CountMin-Sketch: Reset period must fit burst width =⇒ Evasion
How Does ALBUS Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed.

Recall = \( \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \)

Precision = \( \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \)

CountMin-Sketch (Threshold 50%)

Perfect precision by design!

CountMin-Sketch: Reset period must fit burst width. ⇒ Evasion
How Does ALBUS Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed.

Recall $= \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}}$

Precision $= \frac{\text{Reported malicious bursts}}{\text{Reported bursts}}$

CountMin-Sketch (Threshold 50%)

Perfect precision by design!

CountMin-Sketch: Reset period must fit burst width $\Rightarrow$ Evasion
How Does ALBUS Perform Under Burst-Flood Attacks?

**Scenario:** Malicious bursts last for 200 milliseconds and are 20% larger than allowed.

Recall = \( \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \)

Precision = \( \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \)

Perfect precision by design!

CountMin-Sketch (Threshold 50%)
How Does ALBUS Perform Under Burst-Flood Attacks?

**Scenario**: Malicious bursts last for 200 milliseconds and are 20% larger than allowed

Recall = \( \frac{\text{Reported malicious bursts}}{\text{Malicious bursts}} \)

Precision = \( \frac{\text{Reported malicious bursts}}{\text{Reported bursts}} \)

**CountMin-Sketch** (Threshold 50%)

**ALBUS**

Perfect precision by design!

CountMin-Sketch: Reset period must fit burst width \( \Rightarrow \) Evasion
Does ALBUS Satisfy All Requirements of Effective Burst Detection?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:
Does ALBUS Satisfy All Requirements of Effective Burst Detection?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:

- Time-window flexibility
- Memory efficiency
- Processing efficiency
Does ALBUS Satisfy All Requirements of Effective Burst Detection?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:

- **Time-window flexibility**
- **Memory efficiency**
- **Processing efficiency**

ALBUS
Does ALBUS Satisfy All Requirements of Effective Burst Detection?

To withstand burst-flood attacks, a monitoring algorithm must satisfy the following requirements:

- Time-window flexibility
- Memory efficiency
- Processing efficiency

Does ALBUS satisfy all these requirements?
Does ALBUS Allow Efficient Processing?

Yes.

ALBUS has low computational complexity:

- Single hash computation
- No counter-array iterations
- No associative arrays

⇒ ALBUS is hardware-friendly

FPGA implementation for Xilinx Virtex UltraScale+ FPGA:

$\pi.c \pi.f \Sigma 0 CTE_{unused} \lambda.t CTE \Sigma \lambda.f \Sigma \lambda.c > > 3$

Hardware design of a LB-BC combination

Throughput:

200 million packets per second

$\sim 560$ Gbps!
Does ALBUS Allow Efficient Processing?

Yes. ALBUS has low computational complexity:
Does ALBUS Allow Efficient Processing?

Yes. ALBUS has low computational complexity:

- Single hash computation

Hardware design of a LB-BC combination

Throughput: 200 million packets per second

∼ 560 Gbps
Does ALBUS Allow Efficient Processing?

Yes. ALBUS has low computational complexity:

- Single hash computation
- No counter-array iterations

Hardware design of a LB-BC combination
Throughput: 200 million packets per second
∼ 560 Gbps!
Does ALBUS Allow Efficient Processing?

Yes. ALBUS has low computational complexity:

- Single hash computation
- No counter-array iterations
- No associative arrays
Does ALBUS Allow Efficient Processing?

Yes. ALBUS has low computational complexity:

- Single hash computation
- No counter-array iterations
- No associative arrays

⇒ ALBUS is hardware-friendly
Does ALBUS Allow Efficient Processing?

Yes. ALBUS has low computational complexity:

- Single hash computation
- No counter-array iterations
- No associative arrays

⇒ ALBUS is hardware-friendly

FPGA implementation for Xilinx Virtex UltraScale+ FPGA:

Throughput: 200 million packets per second ≈ 560 Gbps!
Does ALBUS Allow Efficient Processing?

Yes. ALBUS has low computational complexity:

- Single hash computation
- No counter-array iterations
- No associative arrays

⇒ ALBUS is hardware-friendly

FPGA implementation for Xilinx Virtex UltraScale+ FPGA:

Throughput: 200 million packets per second ~ 560 Gbps!
Summary: What Is the Contribution of ALBUS?
Summary: What Is the Contribution of ALBUS?

ALBUS is considerably more effective under burst-flood attacks than previous monitoring algorithms.
Summary: What Is the Contribution of ALBUS?

ALBUS is considerably more effective under burst-flood attacks than previous monitoring algorithms.
Summary: What Is the Contribution of ALBUS?

ALBUS is considerably more effective under burst-flood attacks than previous monitoring algorithms.
Summary: What Is the Contribution of ALBUS?

ALBUS is considerably more effective under burst-flood attacks than previous monitoring algorithms.

**CountMin-Sketch**
- Count-Sketch
- **Fixed time-windows**
  - Time-window flexibility
- **Accuracy trade-off**

**ALBUS**
- Leaky-bucket algorithm
  - Hardware-friendly
  - Efficient processing

---

**ALBUS**
- Adaptive Leaky-Bucket Undulation Sensor
- Flows: $f_1, f_2, f_3, f_4, f_5, f_6, f_7, \ldots$

**Filter:**
- Probabilistic decay
- High count:
  - Push to LB

**Error:**
- Reported malicious bursts
- Limited memory
- Reported bursts
- Limited accuracy
- Malicious bursts last for 200 milliseconds and are 10% larger than allowed
-云端流量

**Throughput:**
- Per second
- 560 Gbps!

**Link capacity:**
- 200 million packets

**Net inflow:**
- Inflow - Outflow
- $\times$ Overflow

**CountMin-Sketch:**
- CountSketch
- **Fixed time-windows**
  - Time-window flexibility
- **Accuracy trade-off**

**ALBUS**
- Leaky-bucket algorithm
  - Hardware-friendly
  - Efficient processing
Summary: What Is the Contribution of ALBUS?

ALBUS is considerably more effective under burst-flood attacks than previous monitoring algorithms.

CountMin-Sketch
Count-Sketch

Fixed time-windows

Accuracy trade-off

Time-window flexibility

Memory efficiency

ALBUS

Leaky-bucket algorithm

ALBUS data-structure

DDoS defense systems mostly use the algorithms CountMin-Sketch and Count-Sketch for detection. ALBUS is considerably more effective under burst-flood attacks than previous monitoring algorithms. Malicious bursts last for 200 milliseconds and are 20% larger than allowed. Scenario: Malicious bursts last for 200 milliseconds and are 20% larger than allowed. CountMin-Sketch and Count-Sketch for detection. ALBUS is considerably more effective under burst-flood attacks than previous monitoring algorithms.
Summary: What Is the Contribution of ALBUS?

ALBUS is considerably more effective under burst-flood attacks than previous monitoring algorithms.
Additional Material
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]

\[ p = 1: \text{Majority algorithm (Boyer et al. 1981)} \]
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]

\[ p = 1: \text{Majority algorithm (Boyer et al. 1981)} \]

BC contains flow \( f \) with probability

\[ q_f(1) = \min \left( 1, \frac{\text{Volume of flow } f}{\text{Volume of all flows} \neq f} \right) \]
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]

\[ p = 1: \text{Majority algorithm (Boyer et al. 1981)} \]

BC contains flow \( f \) with probability

\[ q_f(1) = \min \left( 1, \frac{\text{Volume of flow } f}{\text{Volume of all flows } \neq f} \right) \]

\[ p < 1: \text{Bias BC flow towards largest flow } g \]
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]

\[ p = 1: \text{Majority algorithm (Boyer et al. 1981)} \]

BC contains flow \( f \) with probability

\[ q_f(1) = \min\left(1, \frac{\text{Volume of flow } f}{\text{Volume of all flows} \neq f}\right) \]

\[ p < 1: \text{Bias BC flow towards largest flow } g \]
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]

\[ p = 1: \text{Majority algorithm (Boyer et al. 1981)} \]

BC contains flow \( f \) with probability

\[ q_f(1) = \min \left( 1, \frac{\text{Volume of flow } f}{\text{Volume of all flows} \neq f} \right) \]

\[ p < 1: \text{Bias BC flow towards largest flow } g \]
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]

\( p = 1 \): Majority algorithm (Boyer et al. 1981)

BC contains flow \( f \) with probability

\[ q_f(1) = \min \left( 1, \frac{\text{Volume of flow } f}{\text{Volume of all flows } \neq f} \right) \]

\( p < 1 \): Bias BC flow towards \textit{largest} flow \( g \)

\[ q_g(p) \]

\[ p \]
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]

\[ p = 1: \text{Majority algorithm (Boyer et al. 1981)} \]

BC contains flow \( f \) with probability

\[ q_f(1) = \min \left( 1, \frac{\text{Volume of flow } f}{\text{Volume of all flows } \neq f} \right) \]

\[ p < 1: \text{Bias BC flow towards largest flow } g \]

\[ q_g(p) \]

\[ 0 \quad 0.001 \quad 0.01 \quad 0.1 \quad 1 \]

\[ p \]
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]

\[ p = 1: \text{Majority algorithm (Boyer et al. 1981)} \]

BC contains flow \( f \) with probability

\[ q_f(1) = \min \left( 1, \frac{\text{Volume of flow } f}{\text{Volume of all flows } \neq f} \right) \]

\[ p < 1: \text{Bias BC flow towards largest flow } g \]

\[ q_g(p) \]

\[ 0 \quad 0.001 \quad 0.01 \quad 0.1 \quad 1 \]
How Do Background Counters Help Detection Accuracy?

Probabilistic decay identifies large background flows

\[ p = \text{Count-modification probability} \]

\[ p = 1: \text{Majority algorithm (Boyer et al. 1981)} \]

BC contains flow \( f \) with probability

\[ q_f(1) = \min\left(1, \frac{\text{Volume of flow } f}{\text{Volume of all flows } \neq f}\right) \]

\[ p < 1: \text{Bias BC flow towards largest flow } g \]

\[ q_g(p) \]

Optimal \( p \)

Majority algorithm

0 0.001 0.01 0.1 1

p

Network Security Group
Department of Computer Science