Load Shedding in Network Monitoring Applications

P. Barlet-Ros\textsuperscript{1} G. Iannaccone\textsuperscript{2} J. Sanjuàs-Cuxart\textsuperscript{1}  
D. Amores-López\textsuperscript{1} J. Solé-Pareta\textsuperscript{1}

\textsuperscript{1}Technical University of Catalonia (UPC)  
Barcelona, Spain  
{pbarlet, jsanjuas, damores, pareta}@ac.upc.edu

\textsuperscript{2}Intel Research  
Berkeley, CA  
gianluca.iannaccone@intel.com

USENIX Annual Technical Conference, 2007
Outline

1. Introduction
   - Motivation
   - Case Study: Intel CoMo

2. Prediction Method
   - Work Hypothesis
   - Multiple Linear Regression

3. Load Shedding
   - When, Where and How Much

4. Evaluation and Operational Results
   - Performance Results
   - Accuracy Results

5. Conclusions and Future Work
Outline

1. **Introduction**
   - Motivation
   - Case Study: Intel CoMo

2. **Prediction Method**
   - Work Hypothesis
   - Multiple Linear Regression

3. **Load Shedding**
   - When, Where and How Much

4. **Evaluation and Operational Results**
   - Performance Results
   - Accuracy Results

5. **Conclusions and Future Work**
Motivation

- Building robust network monitoring applications is hard
  - Unpredictable nature of network traffic
  - Anomalous traffic, extreme data mixes, highly variable data rates

- Processing requirements have greatly increased in recent years
  - E.g., intrusion and anomaly detection
Motivation

Building robust network monitoring applications is hard
  - Unpredictable nature of network traffic
  - Anomalous traffic, extreme data mixes, highly variable data rates

Processing requirements have greatly increased in recent years
  - E.g., intrusion and anomaly detection

The problem

- Efficiently handling extreme overload situations
- Over-provisioning is not possible
Case Study: Intel CoMo

- CoMo (Continuous Monitoring)\(^1\)
  - Open-source passive monitoring system
  - Fast implementation and deployment of monitoring applications

- Traffic queries are defined as *plug-in* modules written in C
  - Contain complex computations
  - Stateless filter and *measurement interval*

\(^1\)http://como.sourceforge.net
Case Study: Intel CoMo

CoMo (Continuous Monitoring)\(^1\)
- Open-source passive monitoring system
- Fast implementation and deployment of monitoring applications

Traffic queries are defined as *plug-in* modules written in C
- Contain complex computations
- Stateless filter and *measurement interval*

Traffic queries are **black boxes**
- Arbitrary computations and data structures
- Load shedding cannot use knowledge about the queries

\(^1\)http://como.sourceforge.net
Load Shedding Approach

**Working Scenario**
- Monitoring system supporting multiple arbitrary queries
- Single resource: CPU cycles

**Approach:** Real-time modeling of the queries’ CPU usage

1. Find correlation between **traffic features** and CPU usage
   - Features are **query agnostic** with **deterministic worst case** cost
2. Exploit the correlation to predict CPU load
3. Use the prediction to guide the load shedding procedure
Load Shedding Approach

Working Scenario
- Monitoring system supporting multiple arbitrary queries
- Single resource: CPU cycles

Approach: Real-time modeling of the queries’ CPU usage
1. Find correlation between **traffic features** and CPU usage
   - Features are **query agnostic** with **deterministic worst case** cost
2. Exploit the correlation to predict CPU load
3. Use the prediction to guide the load shedding procedure

Novelty: No *a priori* knowledge of the queries is needed
- Preserves high degree of flexibility
- Increases possible applications and network scenarios
System Overview

Figure: Prediction and Load Shedding Subsystem
Work Hypothesis

Our thesis
- Cost of maintaining data structures needed to execute a query can be modeled looking at a set of traffic features.

Empirical observation
- Different overhead when performing basic operations on the state while processing incoming traffic:
  - E.g., creating or updating entries, looking for a valid match, etc.
- Cost of a query is mostly dominated by the overhead of some of these operations.
Work Hypothesis

Our thesis
- Cost of maintaining data structures needed to execute a query can be modeled looking at a set of traffic features

Empirical observation
- Different overhead when performing basic operations on the state while processing incoming traffic
  - E.g., creating or updating entries, looking for a valid match, etc.
- Cost of a query is mostly dominated by the overhead of some of these operations

Our method
Models queries’ cost by considering the right set of traffic features
Traffic Features vs CPU Usage

Figure: CPU usage compared to the number of packets, bytes and flows
Traffic Features vs CPU Usage

Figure: CPU usage versus the number of packets and flows
# Multiple Linear Regression (MLR)

## Linear Regression Model

\[
Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi} + \varepsilon_i, \quad i = 1, 2, \ldots, n.
\]

- \(Y_i\) = \(n\) observations of the response variable (measured cycles)
- \(X_{ji}\) = \(n\) observations of the \(p\) predictors (traffic features)
- \(\beta_j\) = \(p\) regression coefficients (unknown parameters to estimate)
- \(\varepsilon_i\) = \(n\) residuals (OLS minimizes SSE)

## Feature Selection

- Variant of the Fast Correlation-Based Filter\(^2\) (FCBF)
- Removes **irrelevant** and **redundant** predictors
- Reduces significantly the cost of the MLR

System Overview

Prediction and Load Shedding subsystem

1. Each 100 ms of traffic is grouped into a batch of packets
2. The traffic features are efficiently extracted from the batch (multi-resolution bitmaps)
3. The most relevant features are selected (using FCBF) to be used by the MLR
4. MLR predicts the CPU cycles required by the query to run
5. Load shedding is performed to discard a portion of the batch
6. CPU usage is measured (using TSC) and fed back to the prediction system
Outline

1. Introduction
   - Motivation
   - Case Study: Intel CoMo

2. Prediction Method
   - Work Hypothesis
   - Multiple Linear Regression

3. Load Shedding
   - When, Where and How Much

4. Evaluation and Operational Results
   - Performance Results
   - Accuracy Results

5. Conclusions and Future Work
Load Shedding

**When to shed load**
- When the prediction exceeds the available cycles
  \[ avail\_cycles = (0.1 \times CPU\ frequency) - overhead \]
  - Corrected according to prediction error and buffer space
  - Overhead is measured using the time-stamp counter (TSC)

**How and where to shed load**
- Packet and Flow sampling (hash based)
- The same sampling rate is applied to all queries

**How much load to shed**
- Maximum sampling rate that keeps CPU usage < \( avail\_cycles \)
- \( srate = \frac{avail\_cycles}{pred\_cycles} \)
Load Shedding

When to shed load
- When the prediction exceeds the available cycles
- \( \text{avail\_cycles} = (0.1 \times \text{CPU frequency}) - \text{overhead} \)
  - Corrected according to prediction error and buffer space
  - Overhead is measured using the time-stamp counter (TSC)

How and where to shed load
- Packet and Flow sampling (hash based)
- The same sampling rate is applied to all queries

How much load to shed
- Maximum sampling rate that keeps CPU usage \(< \text{avail\_cycles} \)
- \( srate = \frac{\text{avail\_cycles}}{\text{pred\_cycles}} \)
Load Shedding

When to shed load

- When the prediction exceeds the available cycles
  
  \[ avail\_cycles = (0.1 \times CPU\_frequency) - overhead \]
  
  Corrected according to prediction error and buffer space
  
  Overhead is measured using the time-stamp counter (TSC)

How and where to shed load

- Packet and Flow sampling (hash based)
  
  The same sampling rate is applied to all queries

How much load to shed

- Maximum sampling rate that keeps CPU usage \( < avail\_cycles \)
  
  \[ srate = \frac{avail\_cycles}{pred\_cycles} \]
Outline

1. Introduction
   - Motivation
   - Case Study: Intel CoMo

2. Prediction Method
   - Work Hypothesis
   - Multiple Linear Regression

3. Load Shedding
   - When, Where and How Much

4. Evaluation and Operational Results
   - Performance Results
   - Accuracy Results

5. Conclusions and Future Work
**Load Shedding Performance**

**Figure**: Stacked CPU usage (Predictive Load Shedding)
Load Shedding Performance

Figure: CDF of the CPU usage per batch
Packet Loss

(a) Original CoMo  (b) Reactive Load Shedding  (c) Predictive Load Shedding

Figure: Link load and packet drops
Queries estimate their unsampled output by multiplying their results by the inverse of the sampling rate.

Errors in the query results (*mean ± stdev*)

<table>
<thead>
<tr>
<th>Query</th>
<th>original</th>
<th>reactive</th>
<th>predictive</th>
</tr>
</thead>
<tbody>
<tr>
<td>application (pkts)</td>
<td>55.38% ±11.80</td>
<td>10.61% ±7.78</td>
<td>1.03% ±0.65</td>
</tr>
<tr>
<td>application (bytes)</td>
<td>55.39% ±11.80</td>
<td>11.90% ±8.22</td>
<td>1.17% ±0.76</td>
</tr>
<tr>
<td>flows</td>
<td>38.48% ±902.13</td>
<td>12.46% ±7.28</td>
<td>2.88% ±3.34</td>
</tr>
<tr>
<td>high-watermark</td>
<td>8.68% ±8.13</td>
<td>8.94% ±9.46</td>
<td>2.19% ±2.30</td>
</tr>
<tr>
<td>link-count (pkts)</td>
<td>55.03% ±11.45</td>
<td>9.71% ±8.41</td>
<td>0.54% ±0.50</td>
</tr>
<tr>
<td>link-count (bytes)</td>
<td>55.06% ±11.45</td>
<td>10.24% ±8.39</td>
<td>0.66% ±0.60</td>
</tr>
<tr>
<td>top destinations</td>
<td>21.63 ±31.94</td>
<td>41.86 ±44.64</td>
<td>1.41 ±3.32</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- Effective load shedding methods are now a basic requirement
  
  - Rapidly increasing data rates, number of users and complexity of analysis methods

- Load shedding operates without knowledge of the traffic queries
  
  - Quickly adapts to overload situations by gracefully degrading accuracy via packet and flow sampling

- Operational results in a research ISP network show that:
  
  - The system is robust to severe overload
  - The impact on the accuracy of the results is minimized

- Limitations and Future work
  
  - Load shedding methods for queries non robust against sampling
  - Load shedding strategies to maximize the overall system utility
  - Other system resources (memory, disk bandwidth, storage space)
Availability

- The source code of our load shedding system is publicly available at http://loadshedding.ccaba.upc.edu
- The CoMo monitoring system is available at http://como.sourceforge.net

Acknowledgments

- This work was funded by a University Research Grant awarded by the Intel Research Council and the Spanish Ministry of Education under contract TEC2005-08051-C03-01
- Authors would also like to thank the Supercomputing Center of Catalonia (CESCA) for giving them access the Catalan RREN
Outline

Backup Slides

- Load Shedding Algorithm
- Testbed Scenario
- Related Work
Load shedding algorithm (simplified version)

\[
\text{pred\_cycles} = 0; \\
\textbf{foreach} \ q_i \ \textbf{in} \ Q \ \textbf{do} \\
\begin{align*}
& f_i = \text{feature\_extraction}(b_i); \\
& s_i = \text{feature\_selection}(f_i, h_i); \\
& \text{pred\_cycles} += \text{mlr}(f_i, s_i, h_i); \\
\end{align*}
\]

\[
\textbf{if} \ \text{avail\_cycles} < \text{pred\_cycles} \times (1 + \hat{\text{error}}) \ \textbf{then} \\
\begin{align*}
\textbf{foreach} \ q_i \ \textbf{in} \ Q \ \textbf{do} \\
& b_i = \text{ampling}(b_i, q_i, \text{srate}); \\
& f_i = \text{feature\_extraction}(b_i); \\
\end{align*}
\]

\[
\textbf{foreach} \ q_i \ \textbf{in} \ Q \ \textbf{do} \\
\begin{align*}
& \text{query\_cycles}_i = \text{run\_query}(b_i, q_i, \text{srate}); \\
& h_i = \text{update\_mlr\_history}(h_i, f_i, \text{query\_cycles}_i); \\
\end{align*}
\]
Testbed Scenario

- Equipment and network scenario
  - 2 × Intel® Pentium™ 4 running at 3 GHz
  - 2 × Endace® DAG 4.3GE cards
  - 1 × Gbps link connecting Catalan RREN to Spanish NREN

- Executions

<table>
<thead>
<tr>
<th>Execution</th>
<th>Date</th>
<th>Time</th>
<th>Link load (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>mean/max/min</td>
</tr>
<tr>
<td>predictive</td>
<td>24/Oct/06</td>
<td>9am:5pm</td>
<td>750.4/973.6/129.0</td>
</tr>
<tr>
<td>original</td>
<td>25/Oct/06</td>
<td>9am:5pm</td>
<td>719.9/967.5/218.0</td>
</tr>
<tr>
<td>reactive</td>
<td>05/Dec/06</td>
<td>9am:5pm</td>
<td>403.3/771.6/131.0</td>
</tr>
</tbody>
</table>

- Queries (from the standard distribution of CoMo)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>Port-based application classification</td>
</tr>
<tr>
<td>counter</td>
<td>Traffic load in packets and bytes</td>
</tr>
<tr>
<td>flows</td>
<td>Per-flow counters</td>
</tr>
<tr>
<td>high-watermark</td>
<td>High watermark of link utilization</td>
</tr>
<tr>
<td>pattern search</td>
<td>Finds sequences of bytes in the payload</td>
</tr>
<tr>
<td>top destinations</td>
<td>List of the top-10 destination IPs</td>
</tr>
<tr>
<td>trace</td>
<td>Full-payload collection</td>
</tr>
</tbody>
</table>
Related Work

### Network Monitoring Systems
- Only consider a pre-defined set of metrics
- Filtering, aggregation, sampling, etc.

### Data Stream Management Systems
- Define a declarative query language (small set of operators)
- Operators’ resource usage is assumed to be known
- Selectively discard tuples, compute summaries, etc.
Related Work

Network Monitoring Systems
- Only consider a pre-defined set of metrics
- Filtering, aggregation, sampling, etc.

Data Stream Management Systems
- Define a declarative query language (small set of operators)
- Operators’ resource usage is assumed to be known
- Selectively discard tuples, compute summaries, etc.

Limitations
- Restrict the type of metrics and possible uses
- Assume explicit knowledge of operators’ cost and selectivity