Resource Usage Modeling for Network Monitoring Applications

Josep Sanjuàs-Cuxart

Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain
jsanjuas@ac.upc.edu

EEDC'07

Outline

1. Introduction
   - Motivation
   - System Overview
   - Work Hypothesis
2. System Components
   - Feature Extraction
   - Feature Selection
   - Prediction
   - Load Shedding
3. Evaluation and Operational Results
   - Prediction Accuracy
   - Performance Results
4. Conclusions and Future Work

Outline

1. Introduction
   - Motivation
   - System Overview
   - Work Hypothesis
2. System Components
   - Feature Extraction
   - Feature Selection
   - Prediction
   - Load Shedding
3. Evaluation and Operational Results
   - Prediction Accuracy
   - Performance Results
4. 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 imposed on network monitoring have greatly increased in recent years
  - Continuous and fine-grained analysis is now a basic requirement
  - 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 imposed on network monitoring have greatly increased in recent years
  - Continuous and fine-grained analysis is now a basic requirement
  - E.g., intrusion and anomaly detection

The problem

- Existing solutions do not address efficiently overload situations
- Over-provisioning the system to handle peak rates is not viable

Case Study

- CoMo (Continuous Monitoring)[1]
  - Open-source passive monitoring system
  - Allows for fast implementation and deployment of monitoring applications
- Traffic queries are defined as *plug-in* modules written in C
  - All complex computations are contained within the plug-in module
  - Users also provide a stateless filter and the *measurement interval*

[1] `http://como.sourceforge.net`

Case Study

- CoMo (Continuous Monitoring)[1]
  - Open-source passive monitoring system
  - Allows for fast implementation and deployment of monitoring applications
- Traffic queries are defined as *plug-in* modules written in C
  - All complex computations are contained within the plug-in module
  - Users also provide a stateless filter and the *measurement interval*

Traffic queries are black boxes

- CoMo does not restrict the type of computations a query can perform nor the data structures it can use
- Load shedding must be performed without any explicit knowledge of the traffic 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

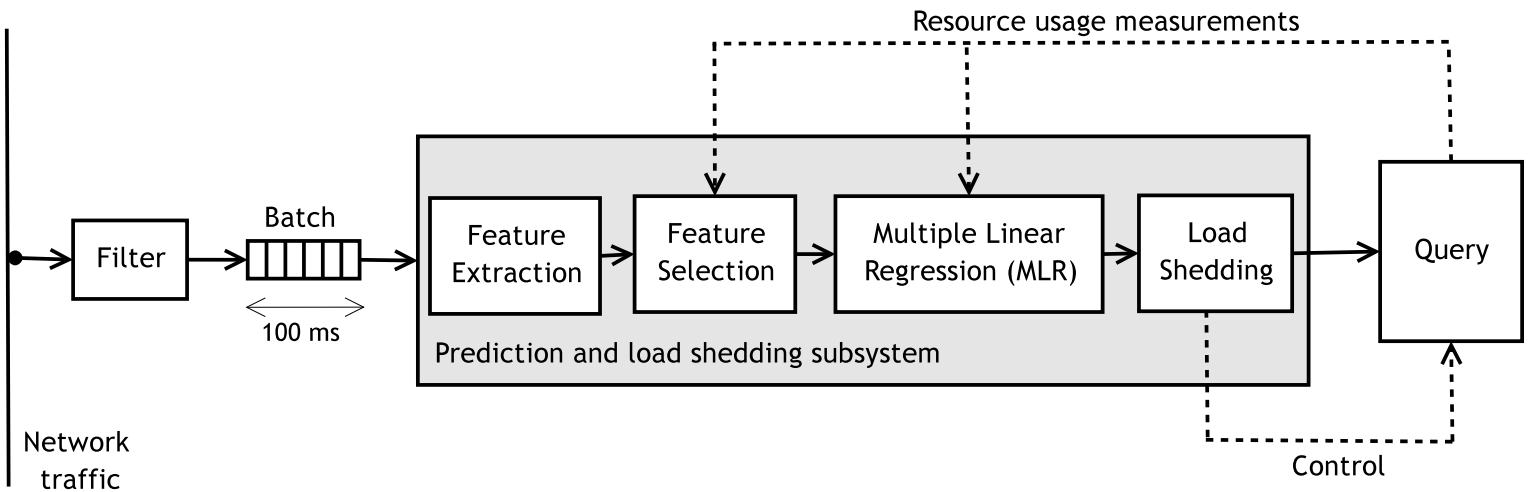

Figure: Prediction and Load Shedding Subsystem

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

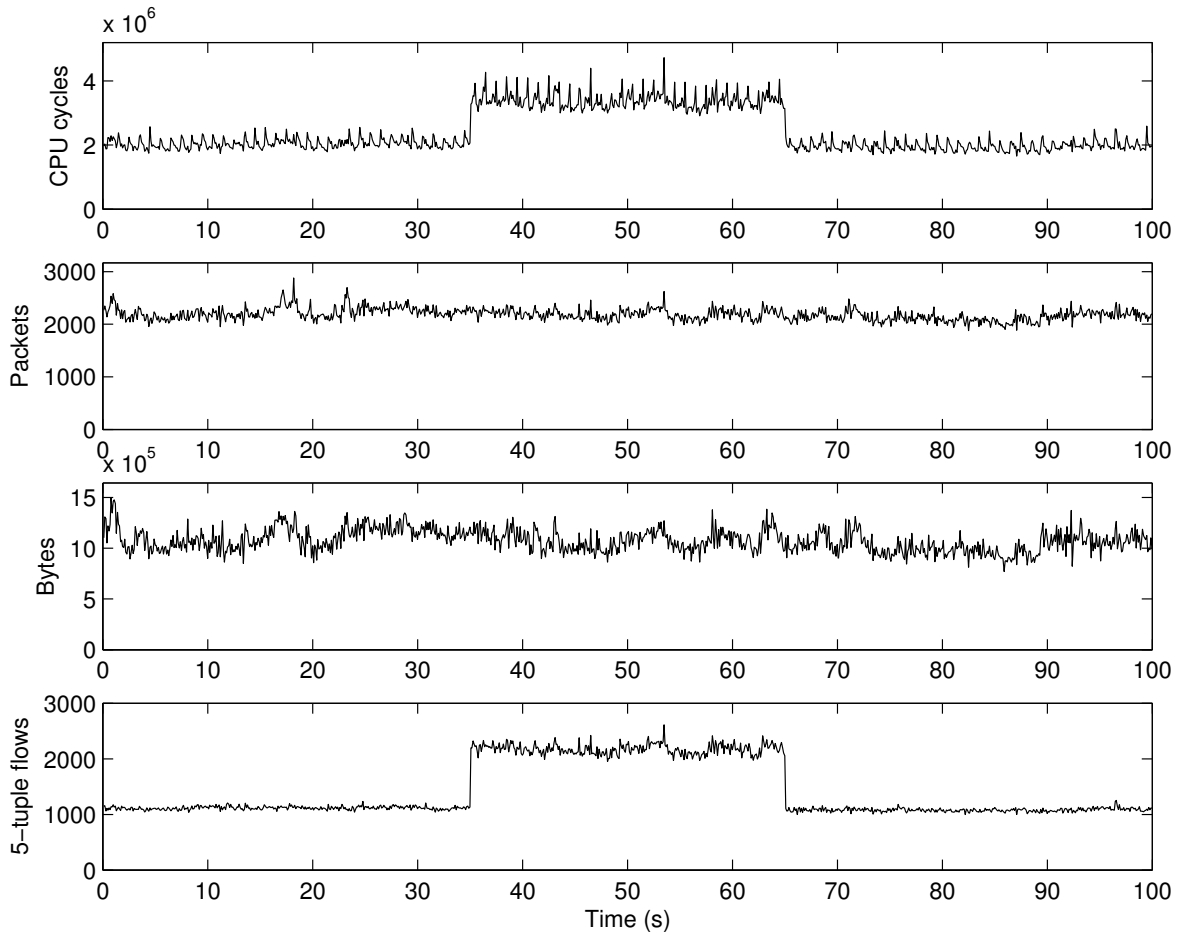

Figure: CPU usage compared to the number of packets, bytes and flows

Outline

1. Introduction
   - Motivation
   - System Overview
   - Work Hypothesis
2. System Components
   - Feature Extraction
   - Feature Selection
   - Prediction
   - Load Shedding
3. Evaluation and Operational Results
   - Prediction Accuracy
   - Performance Results
4. Conclusions and Future Work

Feature Extraction

Goal

- Extract information useful to predict CPU usage
- Low overhead is a must

Traffic Features

- Number of packets, number of bytes
- Traffic aggregates: unique source addresses, unique source-destination pairs...
- Minimize overhead: approximate in linear time and with low memory

Feature Selection

Goal

- Select relevant features for each query
- Minimize the cost of building a prediction model

Algorithm

- Based on the Fast Correlation-Based Filter
- Removes features not correlated with CPU usage
- Removes inter-correlated features

Prediction

Multiple Linear Regression

- Build a model for each query
- Predictors: selected features
- Response variable: CPU cycles

Load Shedding

When to shed load

- Instrument the system to know the amount of available cycles
- Shed load when the prediction exceeds the available cycles

How much load to shed

- $sampling\_rate = \min\left(1, \frac{available\_cycles}{predicted\_cycles}\right)$

Where to shed load

- We treat all queries equally

How to shed the load

- Either packet sampling of flow-wise sampling
- Queries choose a method at configuration time
- Notify queries of their sampling rate

Outline

1. Introduction
   - Motivation
   - System Overview
   - Work Hypothesis
2. System Components
   - Feature Extraction
   - Feature Selection
   - Prediction
   - Load Shedding
3. Evaluation and Operational Results
   - Prediction Accuracy
   - Performance Results
4. Conclusions and Future Work

Prediction Accuracy

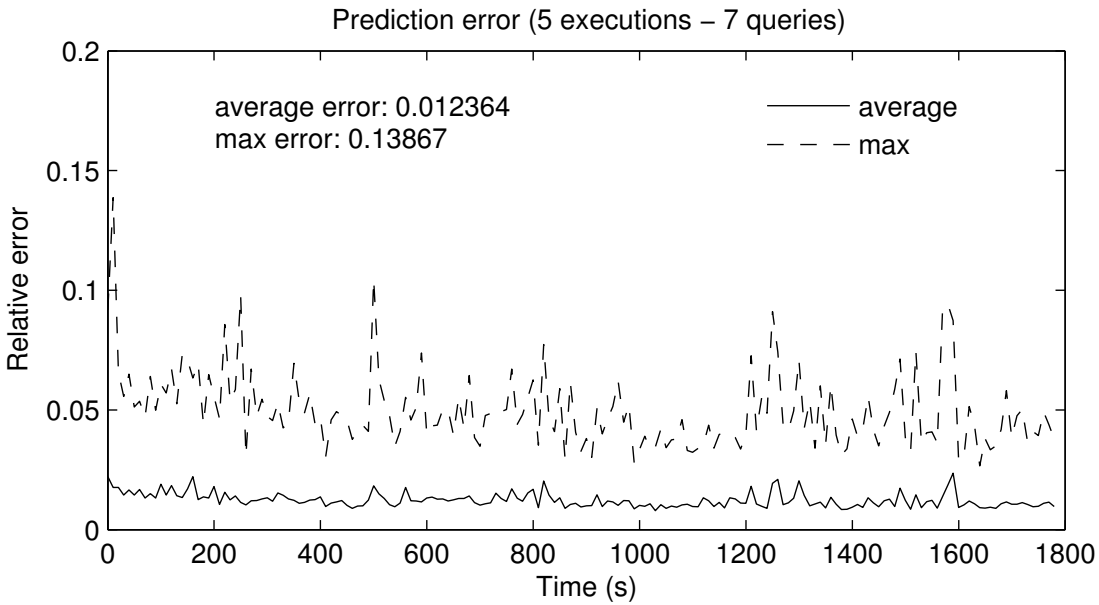

Figure: Prediction error

Observations

- Average error stable around 1%
- Error peaks at 15%, maximum error across queries around 5%

Load Shedding Performance

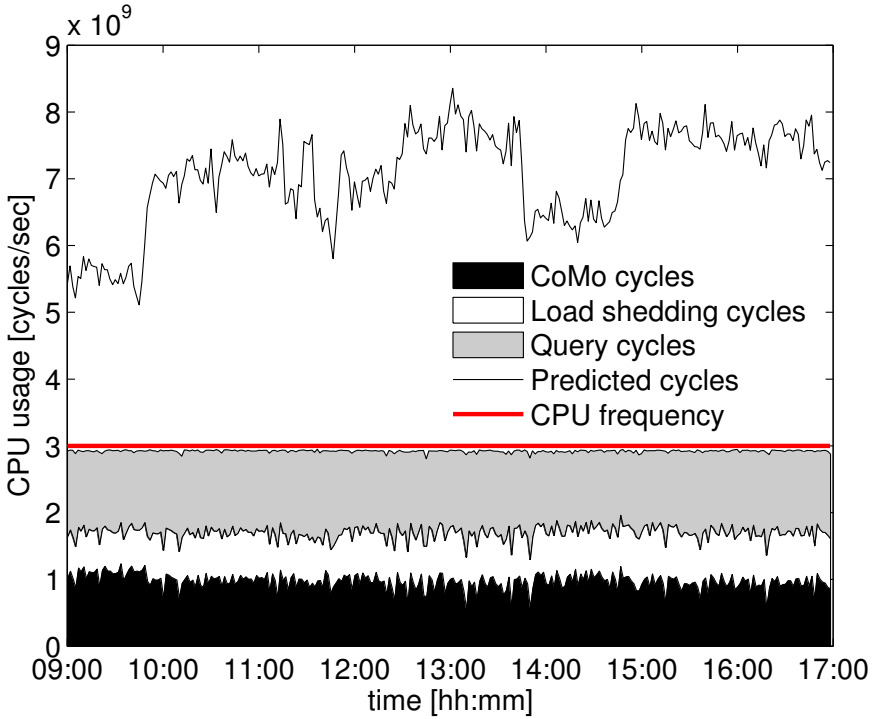

Figure: CPU usage (*load_shedding* execution)

Observations

- The system is under severe overload (more than twice the cycles available are needed)
- The system is robust to overload and load shedding overhead is stable

Packet Loss and CPU Usage

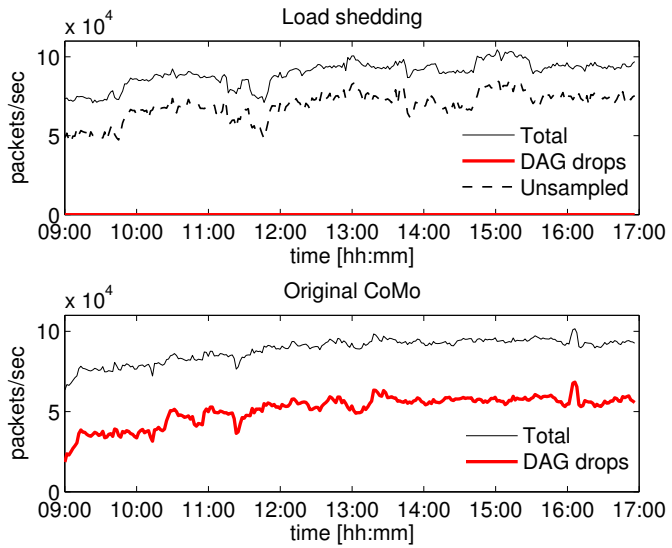

Figure: Link load and packet drops

Load Shedding robust to overload

- Not a single packet was lost
- Selects high sampling rates (see paper)

Original CoMo is not stable

- Packet loss around 50%

Outline

1. Introduction
   - Motivation
   - System Overview
   - Work Hypothesis
2. System Components
   - Feature Extraction
   - Feature Selection
   - Prediction
   - Load Shedding
3. Evaluation and Operational Results
   - Prediction Accuracy
   - Performance Results
4. Conclusions and Future Work

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
- Our predictive load shedding scheme operates without explicit 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
  - MLR assumes linear relationship between features and CPU usage
  - More complex load shedding strategies
  - Load shedding strategies to maximize the overall system utility
  - Similar approaches for other system resources (e.g., memory, disk bandwidth, storage space)