On-line Predictive Load Shedding for Network Monitoring

P. Barlet-Ros[1] D. Amores-López[1] G. Iannaccone[2]
J. Sanjuàs-Cuxart[1] J. Solé-Pareta[1]

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

[2]Intel Research
Berkeley, CA
gianluca.iannaccone@intel.com

IFIP Networking, 2007

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Outline

Outline

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

Proposed Method

Our solution

- On-line predictive load shedding scheme for network monitoring
  - Quickly adapts to overload situations
  - Gracefully degrades accuracy of analysis methods

Novelty

- No *a priori* knowledge of the monitoring applications is needed
  - Monitoring system preserves a high degree of flexibility
  - The range of possible applications and network scenarios is increased

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`

Contributions

Previous work[2]

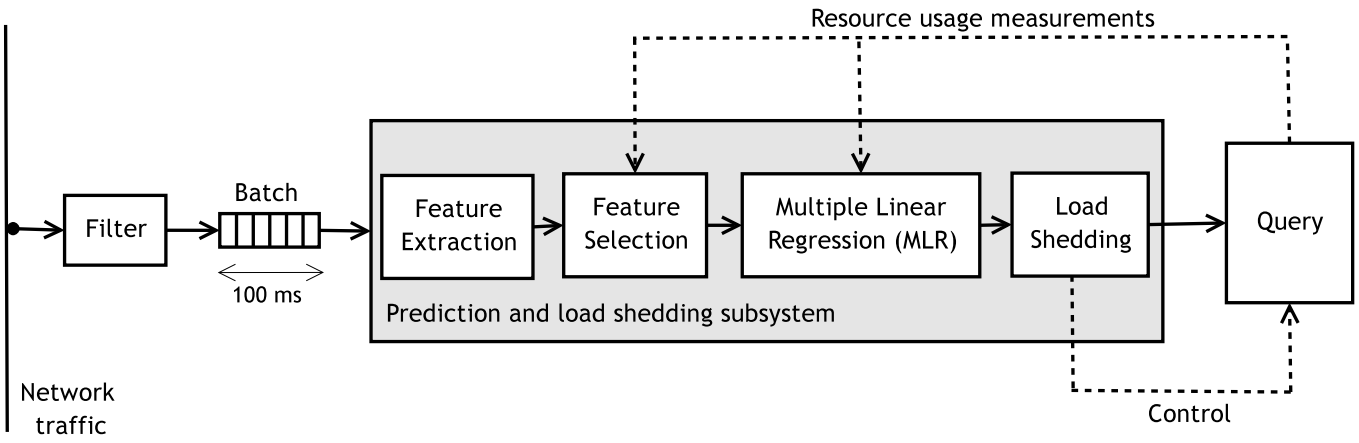
- We defined a method capable of predicting the resource usage of arbitrary network traffic queries
  - Automatically identifies the traffic features that best model the cost of each query from small sequences of packets
  - Uses these features to accurately predict the query's CPU usage

Contributions of this paper

- Use this prediction to guide the monitoring system on deciding
  1. When to shed load
  2. Where to shed load
  3. How much load to shed
- Impact of overload on the accuracy of the results is minimized

[2] http://loadshedding.ccaba.upc.edu

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
3. The most relevant features are selected 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 and fed back to the prediction system

Outline

When to Shed Load

How do we detect overload?

- When the prediction exceeds the available cycles

How do we measure the available cycles?

- We compute the cycles available in 100*ms*
- Overhead is measured using the time-stamp counter (TSC)
  - Overhead of our prediction method
  - Overhead of other CoMo tasks (e.g., memory management)
- $avail\_cycles = (0.1 \times CPU\ frequency) - overhead$

We correct the *avail_cycles* according to . . .

- Observed prediction error in previous batches
- Space available in the buffers (e.g., in the capture devices)

Where and How to Shed Load

Approach

Adaptively reducing the size of the batch

Method

1. Packet sampling
2. Flow sampling
   - Hash-based *Flowwise Sampling* samples entire flows without caching the flow keys

Where to shed load

The same sampling rate is applied to all queries

How Much Load to Shed

Magnitude of load shedding

- Maximum sampling rate that keeps the CPU usage below *avail_cycles*: $srate = \frac{avail\_cycles}{pred\_cycles}$

Assumptions

- The CPU usage is proportional to the number of packets/flows
- Prediction error and load shedding cost is 0

Solution

- We maintain an EWMA of the prediction error
- We maintain an EWMA of the load shedding overhead
- Sampling rate is computed as: $srate = \frac{avail\_cycles - \widehat{ls\_cycles}}{pred\_cycles \times (1 + \widehat{error})}$

Outline

Load Shedding Performance

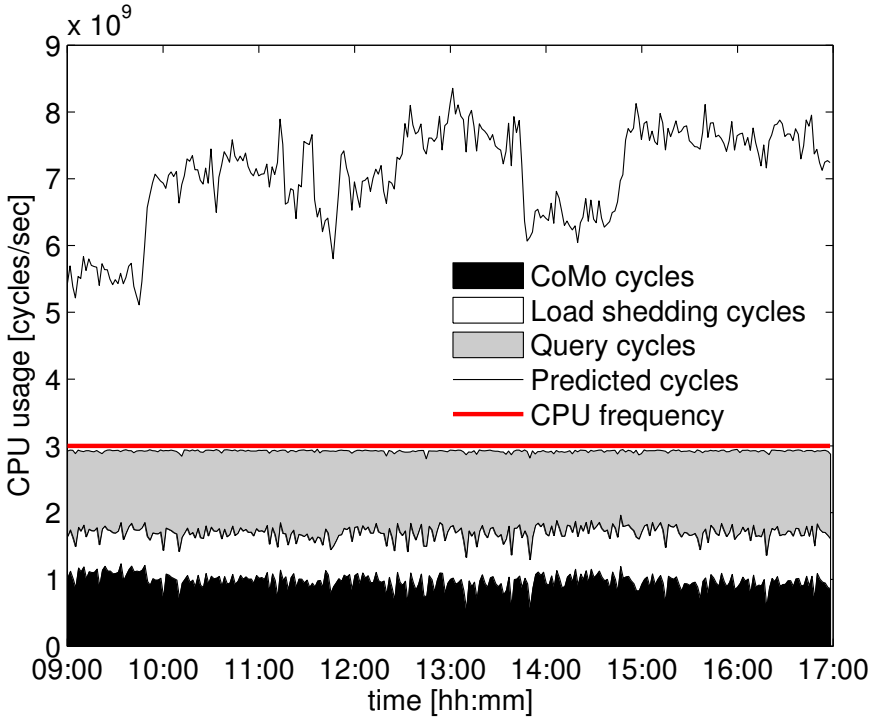
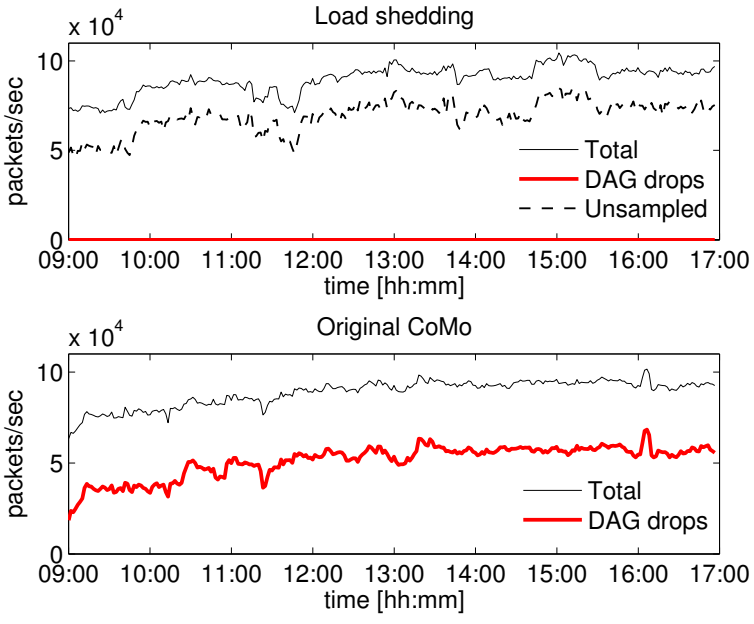


Figure: CPU usage (*load_shedding* execution)

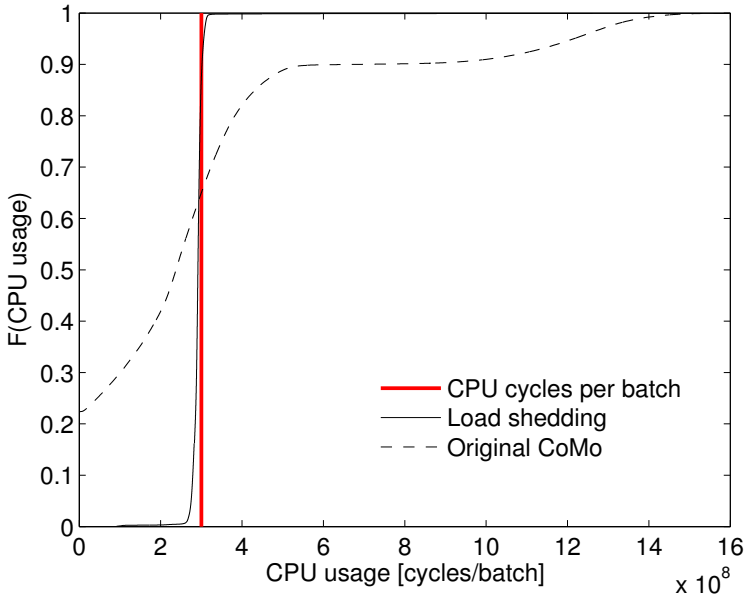
Observations

- 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



(a) Link load and packet drops



(b) CDF of the CPU usage per batch

Load Shedding CoMo is robust to overload

- Not a single packet was lost
- CPU usage around 3×10^8 cycles/batch

Original CoMo is not stable

- Probability of exceeding *avail_cycles* $> 30\%$
- Probability of losing an entire batch $> 20\%$

Accuracy Results

- Three queries estimate their unsampled output by multiplying their results by the inverse of the sampling rate
  - *Counter*: Packet sampling
  - *Flows*: Flow sampling
  - *Top destinations*: Packet sampling

- Errors in the query results (*mean* $\pm$ *stdev*)

Query	***original_como***	***load_shedding***
counter (packets)	55.03% $\pm$11.45	0.54% $\pm$0.50
counter (bytes)	55.06% $\pm$11.45	0.66% $\pm$0.60
flows	38.48% $\pm$902.13	2.88% $\pm$3.34
top destinations	21.63 $\pm$31.94	1.41 $\pm$3.32

Outline

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
  - Load shedding methods for queries non robust against sampling
  - Load shedding strategies to maximize the overall system utility
  - Similar approaches for other system resources (e.g., 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

Work Hypothesis

Our thesis

- The cost of maintaining the data structures needed to execute a query can be modeled looking at a set of traffic features
  - Characterizes the input data (e.g., unique IP addresses)

We empirically observed that . . .

- Each query incurs different overhead when performing basic operations on the state it maintains
- The time spent by a query is mostly dominated by the overhead of some of these operations

Conclusion

The cost of a query can be modeled by considering the right set of simple traffic features

Work Hypothesis

Our thesis

- The cost of maintaining the data structures needed to execute a query can be modeled looking at a set of traffic features
  - Characterizes the input data (e.g., unique IP addresses)

We empirically observed that . . .

- Each query incurs different overhead when performing basic operations on the state it maintains
- The time spent by a query is mostly dominated by the overhead of some of these operations

Conclusion

The cost of a query can be modeled by considering the right set of simple traffic features

Load Shedding Algorithm

Load shedding algorithm (simplified version)

pred_cycles = 0;
foreach q_i *in* Q **do**
    f_i = `feature_extraction(`b_i`)`;
    s_i = `feature_selection(`f_i, h_i`)`;
    pred_cycles += `mlr(`f_i, s_i, h_i`)`;

if *avail_cycles* $<$ *pred_cycles* $\times$ $(1 + \widehat{error})$ **then**
    foreach q_i *in* Q **do**
        b_i = `sampling(`b_i, q_i, *srate*`)`;
        f_i = `feature_extraction(`b_i`)`;

foreach q_i *in* Q **do**
    $query\_cycles_i$ = `run_query(`b_i, q_i, *srate*`)`;
    h_i = `update_mlr_history(`h_i, f_i, $query\_cycles_i$`)`;

Testbed Scenario

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

Execution	Date/Time	Link load (Mbps) mean/max/min
load_shedding	24/Oct/06 9:00-17:00	750.4/973.6/129.0
original_como	25/Oct/06 9:00-17:00	719.9/967.5/218.0

- Queries (from the standard distribution of CoMo)

Name	Description
application	Port-based application classification
counter	Traffic load in packets and bytes
flows	Per-flow counters
high-watermark	High watermark of link utilization
pattern search	Finds sequences of bytes in the payload
top destinations	List of the top-10 destination IPs
trace	Full-payload collection