A flow aware packet sampling mechanism for high speed links

Marco Canini
DIST
University of Genoa

Damien Fay
Computer Laboratory
University of Cambridge

Andrew W. Moore
Computer Laboratory
University of Cambridge

Raffaele Bolla
DIST
University of Genoa

Work done while visiting the
Cambridge University Computer Laboratory

Goal

- Enable for inline, real-time, application-centric traffic classification
  - Based upon a fixed number of packets at the start of all flows
- Permit implementation at 10+ Gbit/s

Motivation

- Studies* show that start of flow information allows for accurate identification of traffic
- Traditional packet sampling and NetFlow don’t provide required information
- Heavy-tailed nature of Internet traffic allows discarding of significant amount of data without processing

* For example:
[1] W. Li and A. W. Moore. A Machine Learning Approach for Efficient Traffic Classication. In Proceedings of the IEEE MASCOTS, October 2007.
[2] L. Bernaille, R. Teixeira, and K. Salamatian. Early Application Identication. In Proc. of ACM CoNEXT, December 2006.

Method

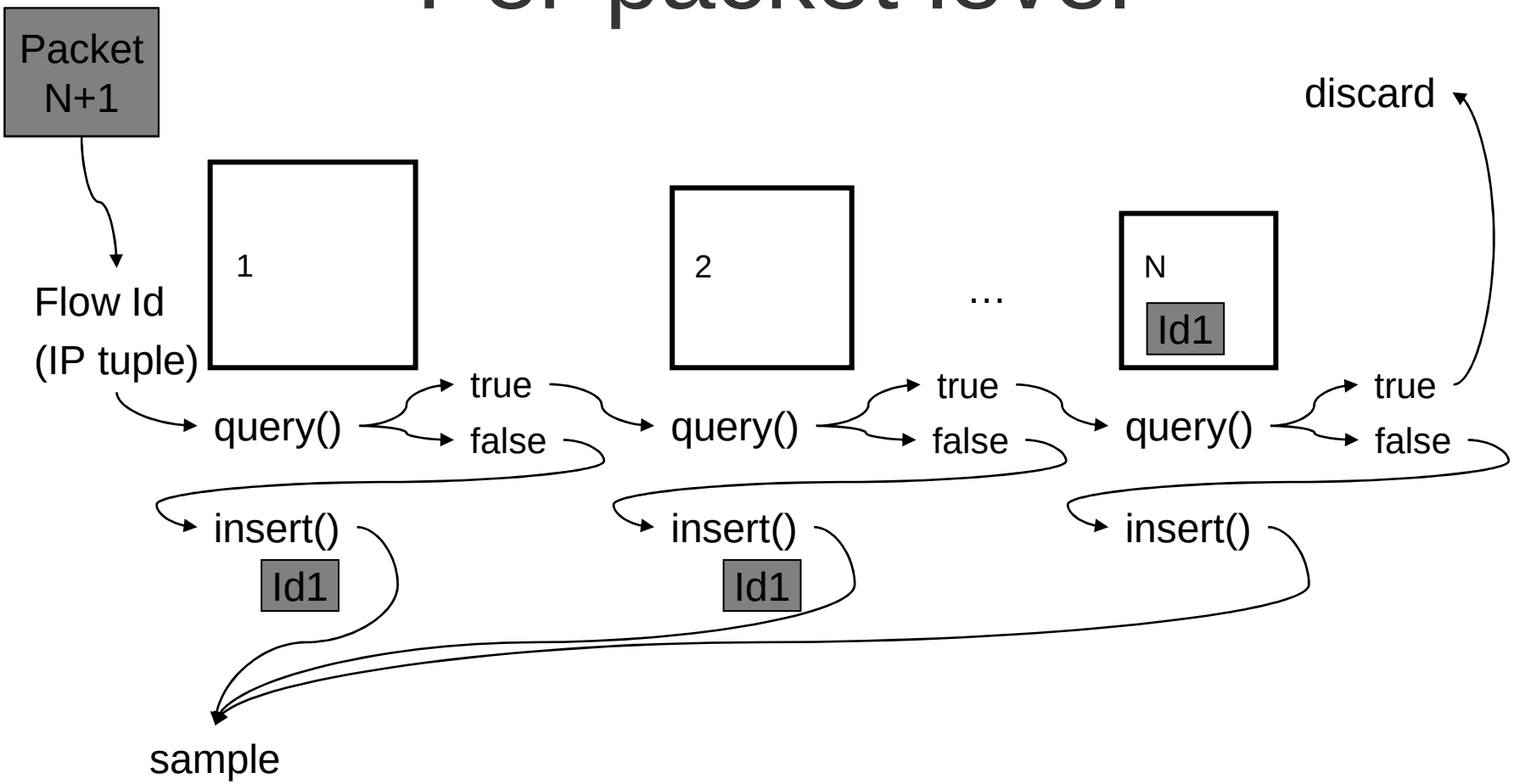
- Focus upon the sampling of a fixed number of packets at the start of all flows
- The scheme consists of two levels
- Per-packet level
  - Within a time window of length *W*, an algorithm based on Bloom filters identifies and samples the first *N* packets from each flow that occur in that window
- Per-window level
  - The memory allocation mechanism finds the parameters required in the sampling scheme

Bloom filter refresher

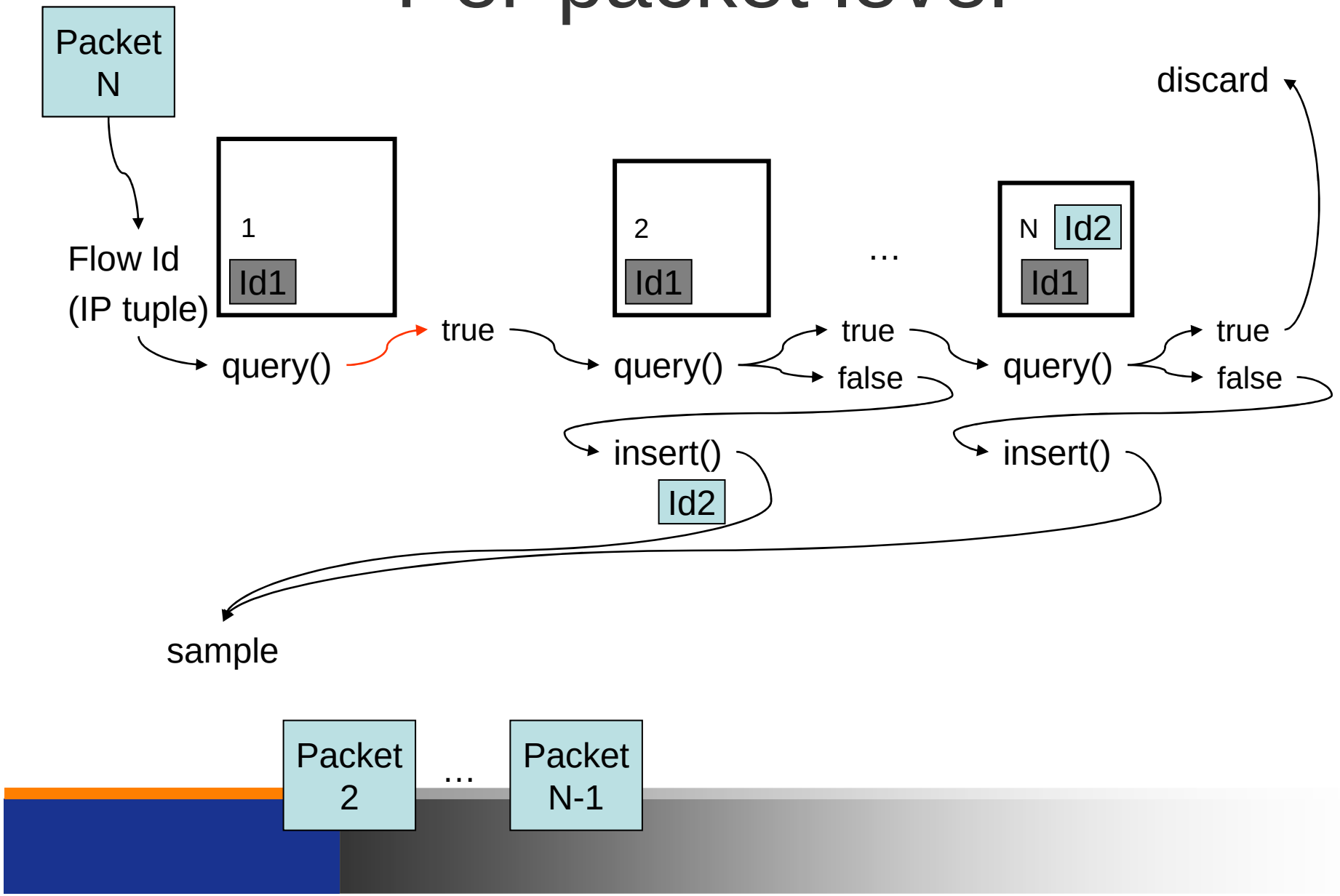
- Simple space-efficient probabilistic data structure for representing a set of elements
- Supports the insert() and query() operations
- False positives are possible
- False negatives are not possible

- Implemented with an array of *m* bits and uses *k* hash functions to map elements to [0,…,*m*-1]

Per-packet level



Per-packet level



False positives

- False positives introduce sampling errors
  - Packets that should have been sampled get discarded
  - Discarded are only ever at the end
- Reduce false positives by
  - Resetting the filters periodically
    - (at the end of each time window)
  - Optimize the memory based upon the traffic

Theory 1

After *n* elements have been inserted, the probability that a specific bit is still 0 is

$$p' = \left(1 - \frac{1}{m}\right)^{kn}$$

The probability of a false positive is approximated with

$$f' = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k}$$ that is minimized for $$k = \ln(2) \cdot \frac{m}{n}$$

In this application we are interested in the number of false positives that occur as the Bloom filter is being filled. The expected number of false positives is

$$E[F] = \sum_{i=1}^{n} \left(1 - \left(1 - \frac{1}{m}\right)^{\ln(2)\cdot(m/n)\cdot i}\right)^{\ln(2)\cdot(m/n)}$$

where *n* is the expected number of elements that will be inserted within the current window

And considering a bank of *N* Bloom filters

$$E[F] = \sum_{j=1}^{N} \sum_{i=1}^{n_j} \left(1 - \left(1 - \frac{1}{m_j}\right)^{\ln(2)\cdot(m_j/n_j)\cdot i}\right)^{\ln(2)\cdot(m_j/n_j)}$$

Theory 2 : Per-window level

- Aim to divide a block of memory *M* into *N* different portions optimally
- The allocation of m_j is given by a system of linear equations

$$\begin{cases} \sum_{j=1}^{N} m_j = M \\ m_j - \alpha \cdot n_j = 0 \quad \forall j = 1 \ldots N \end{cases}$$

- However the number of elements that will be inserted in the filters, n_j, are unknown and have to be estimated (in our case using an AR model)

$$\hat{n}_j(k) = \theta_1 \hat{n}_j(k-1) + \sum_{i=2}^{r} \theta_i n_j(k-i) + \varepsilon_j(k)$$

Theory 3 : Window length

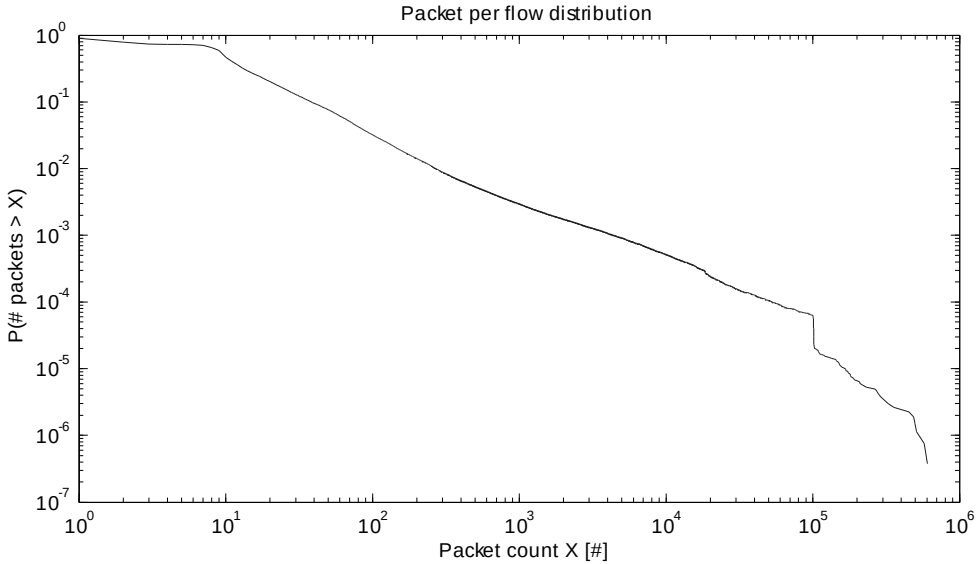
- Given an acceptable level of false positives the appropriate value of *W* can be estimated by use of a search algorithm as the number of false positives is a function of the window size

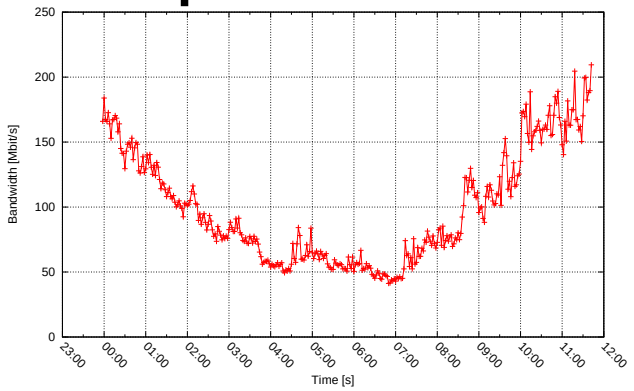
Theory Summary

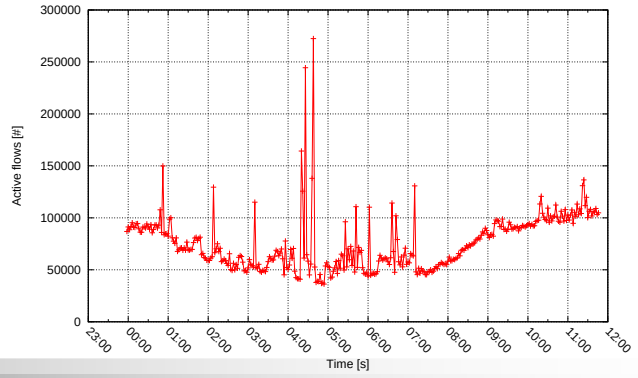
- Memory limits number of samples and size of Bloom filters
- False positive rate allows near-ideal sizing of Bloom filters (for a given amount of memory)
- False positive rate is estimated via AR
- Window length sets rate at which filters are reset

Results

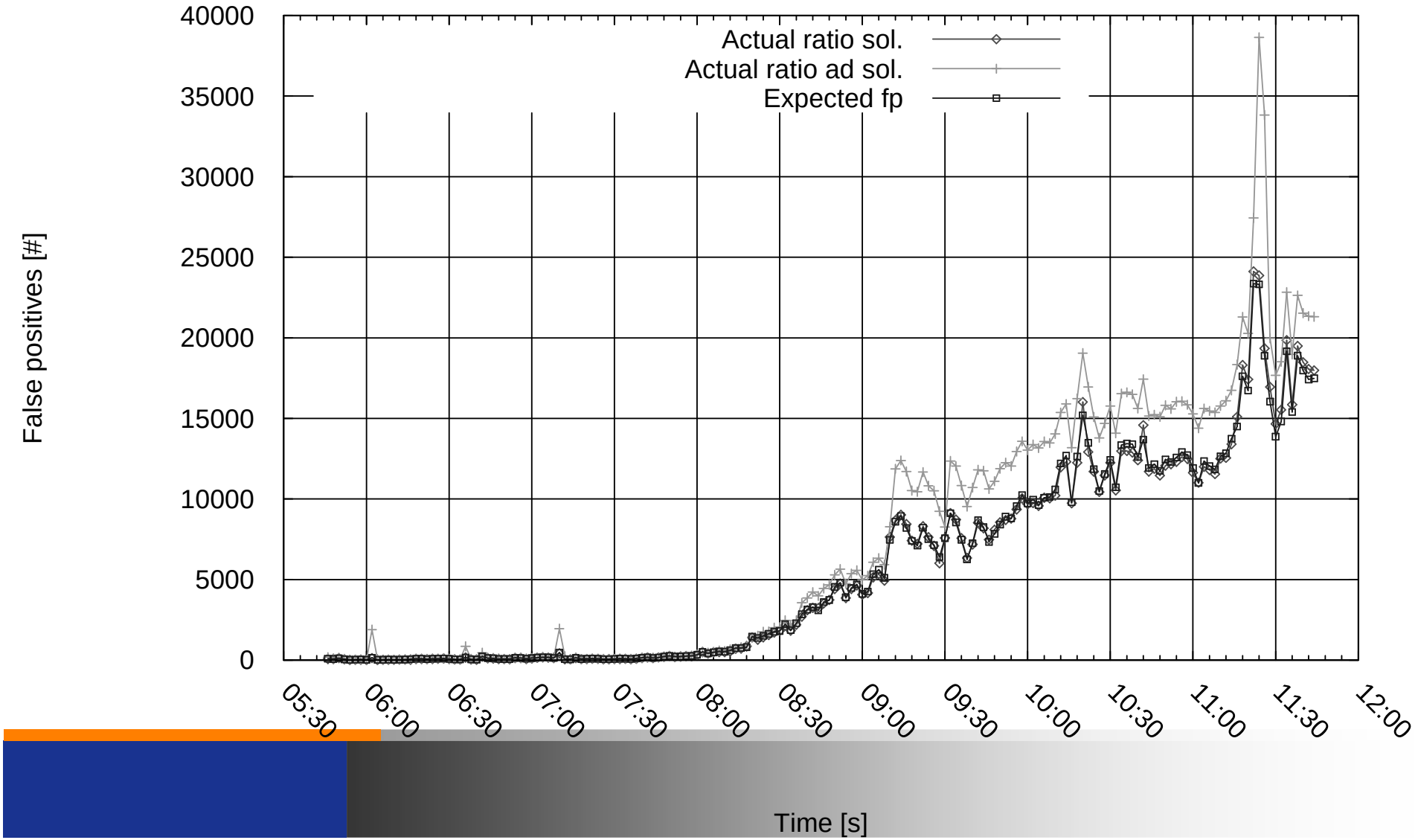
- Software implementation
- 12 hours trace from the edge of a research institute connected via a full-duplex 1 Gbit/s link



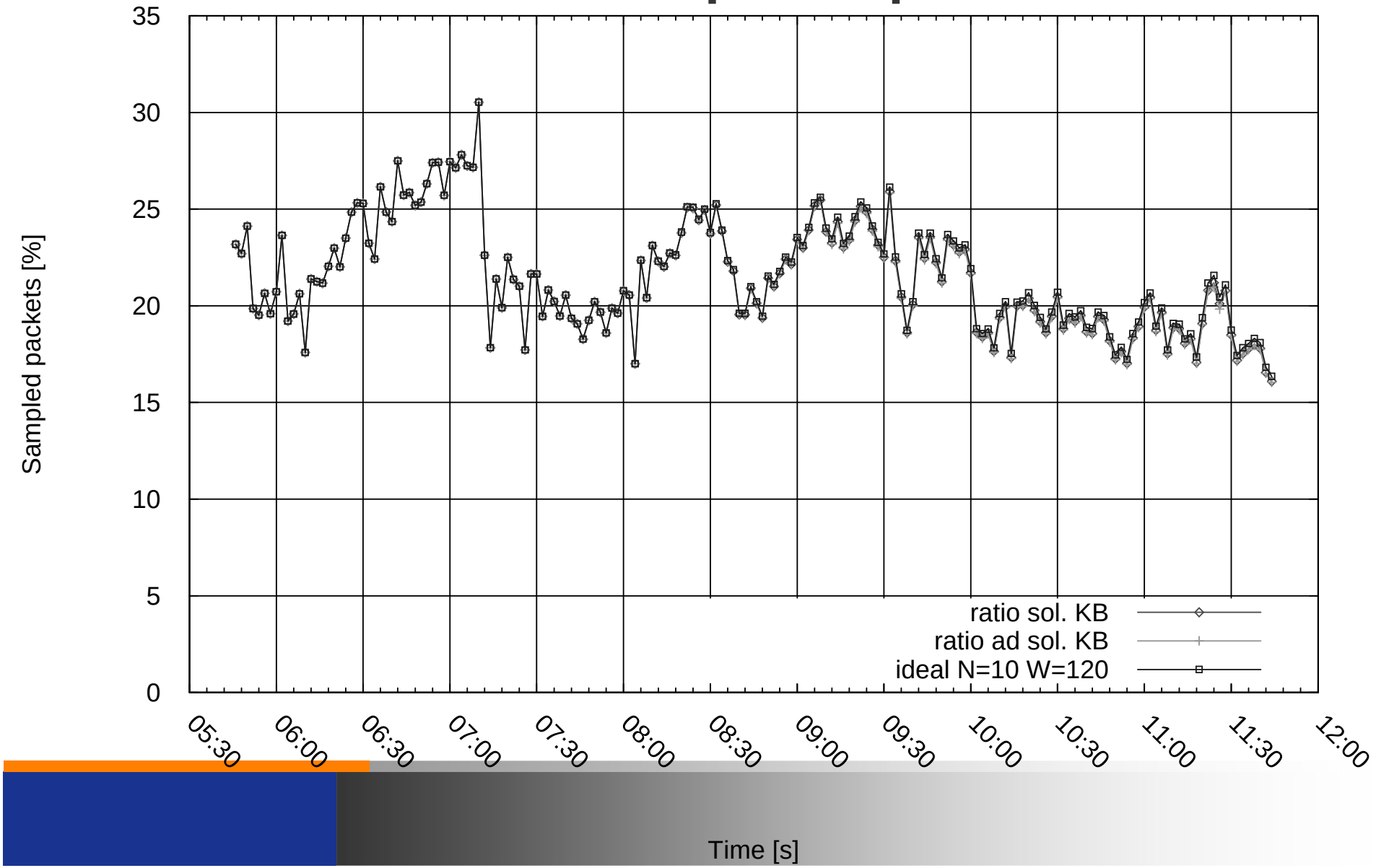




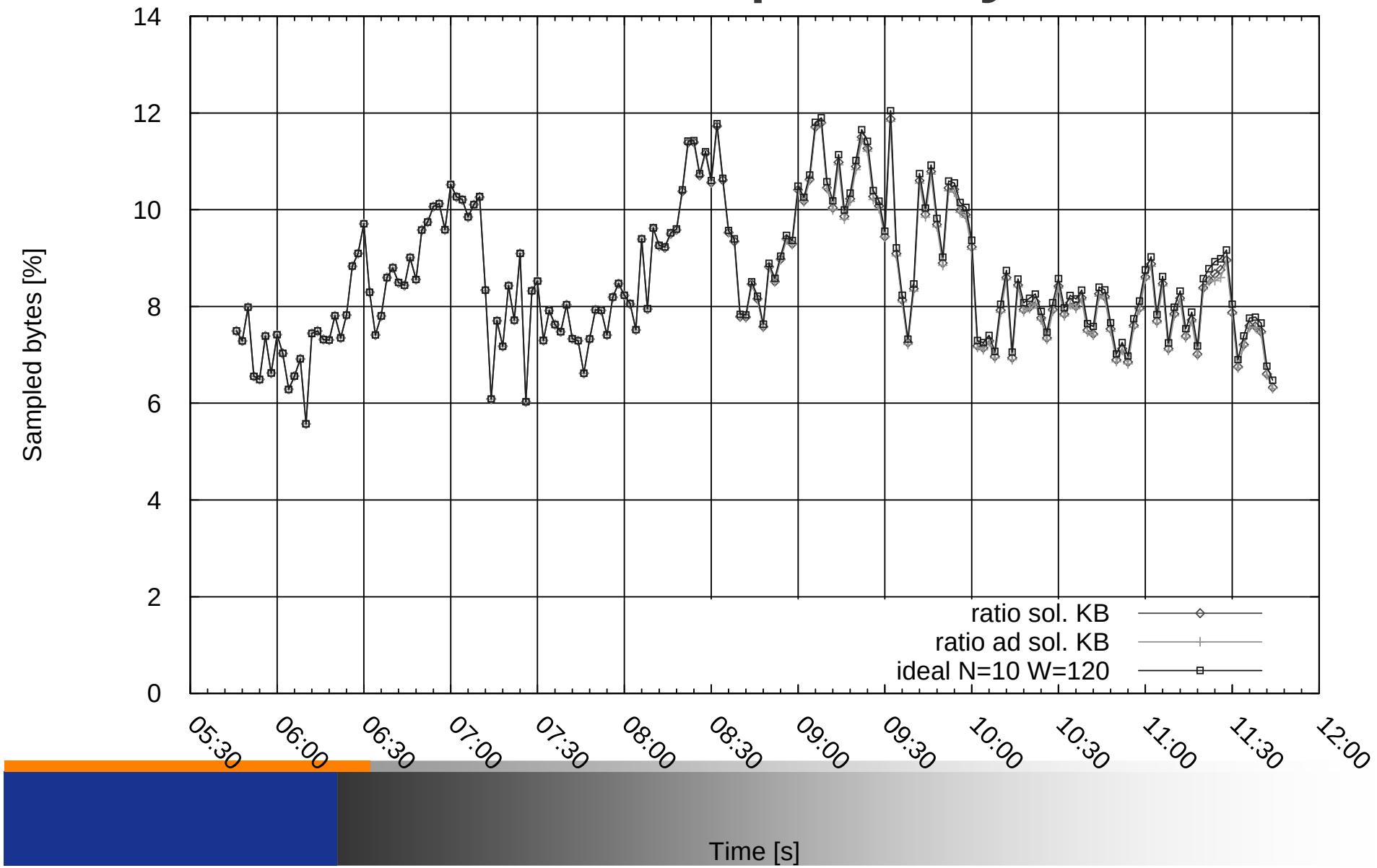
Results: false positives



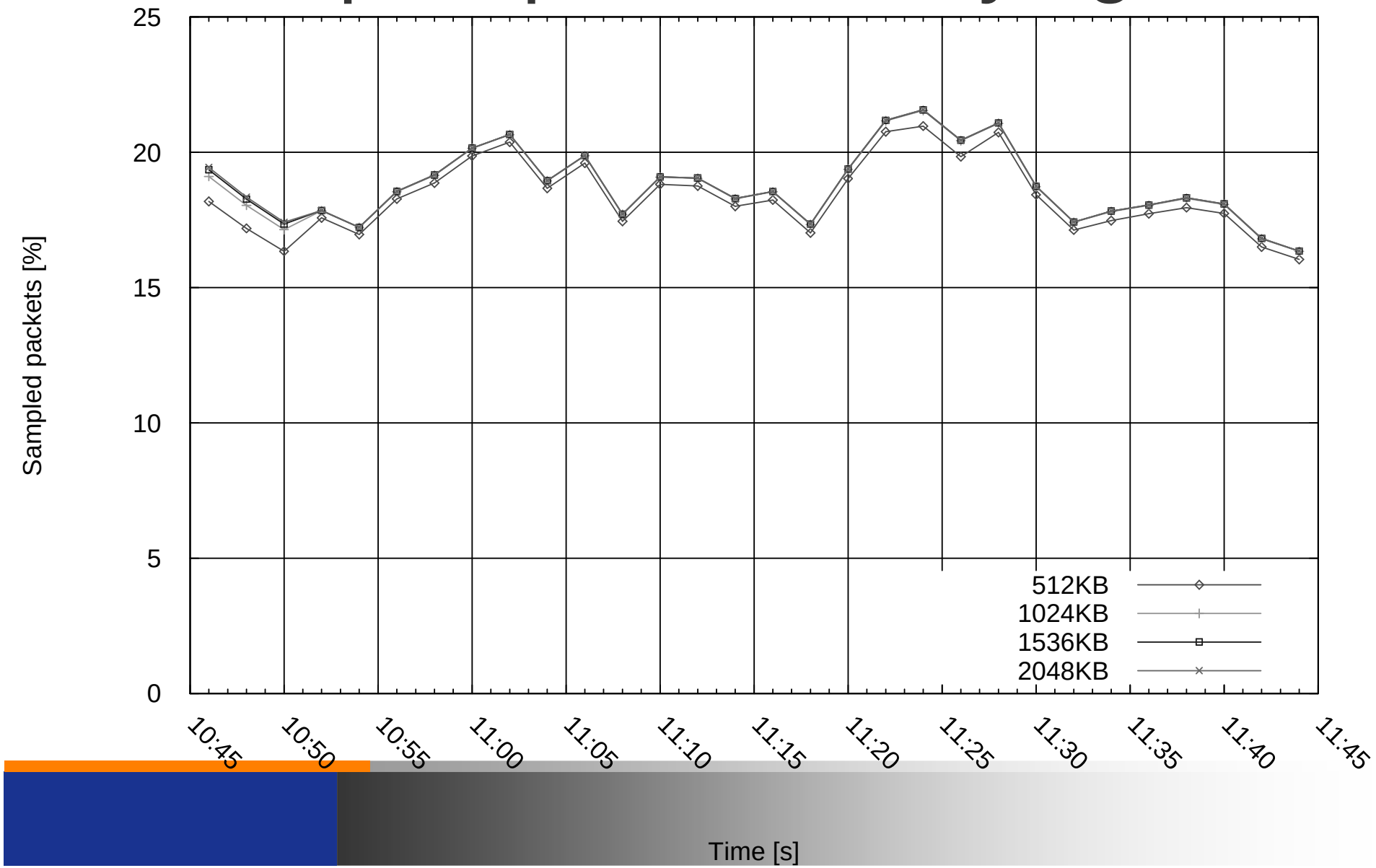
Results: sampled packets



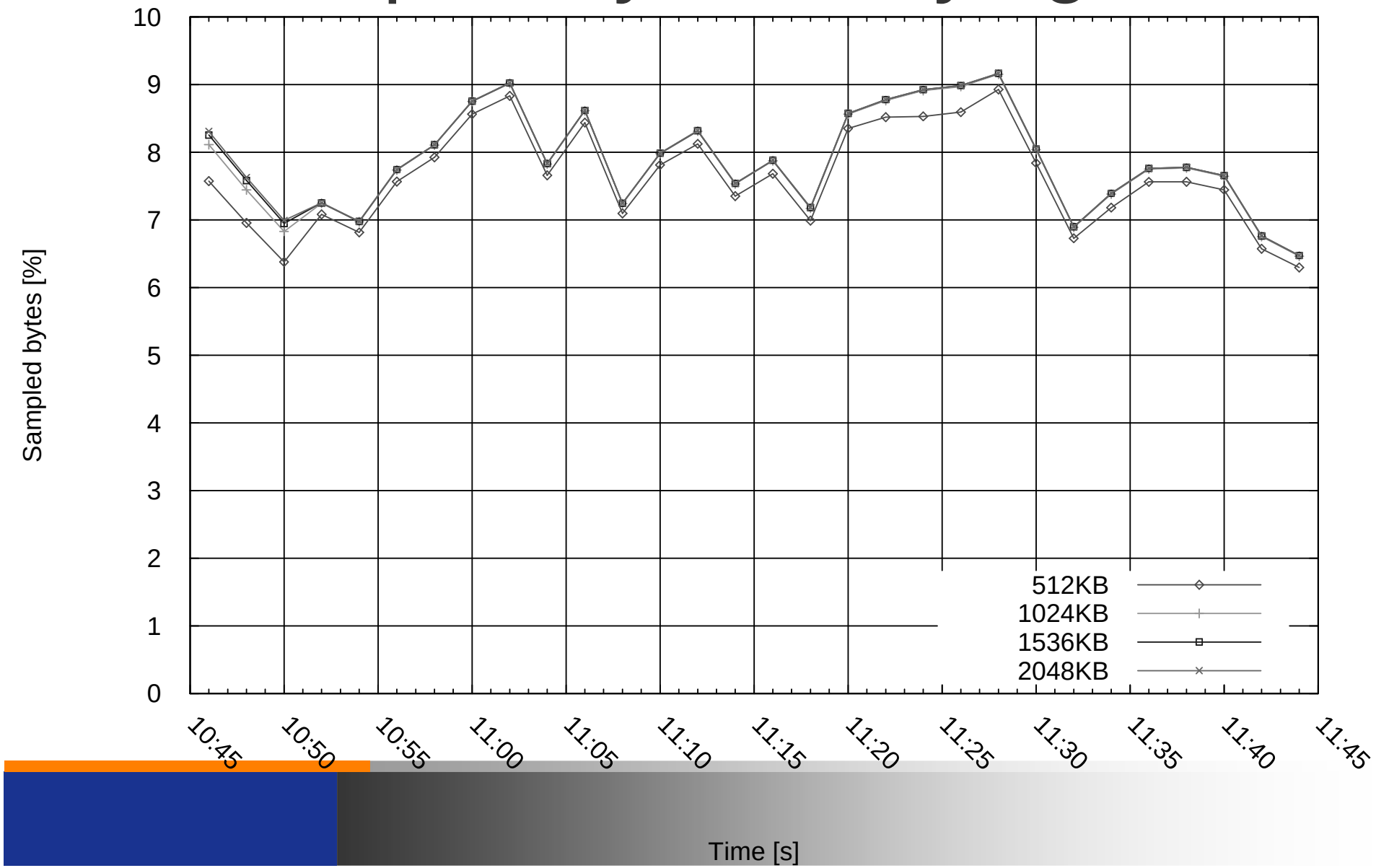
Results: sampled bytes



Sampled packets, varying M



Sampled bytes, varying M



Effects of false positives

M	Affected flows	Unsampled packets	Unsampled bytes
512KB	5529 (0.0021%)	517173 (0.0221%)	199167936 (0.0323%)
1024KB	4863 (0.0018%)	58007 (0.0024%)	22907748 (0.0037%)
1576KB	4274 (0.0016%)	15830 (0.0006%)	6535918 (0.0010%)
2048KB	4237 (0.0016%)	6086 (0.0003%)	2475934 (0.0004%)

Sampling plus flow classifier

- We used a pattern matching flow classifier derived from open source tool l7-filter
- Test data contains 2,642,841 flows
- With sampling (N = 10, W = 120, M = 512 KB) 2,636,549 flows are reported
- 6,292 (0.0024%) unreported flows
- 6,021 (0.0023%) flows had classifier error
- **Total error** is 12,323 **(0.0047%) flows**

Future Work

- FPGA based hardware implementation
- Flow sampling works
- Focus shifted to improving classification (identification) of applications