IRESI - SketchMin algorithm

Simon Bihel, Florestan De Moor

November 25th, 2015

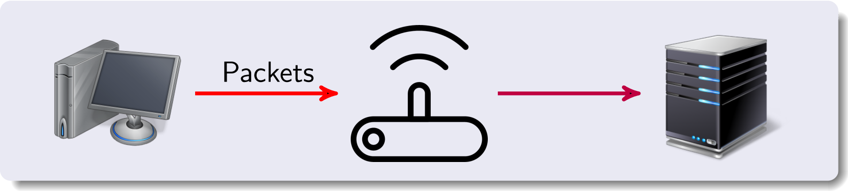
Outline

1. DDoS
   - Simple view
   - Multiple streams
2. Programming the algorithm
   - Extraction
   - Computing the codeviance
3. Experimental results
   - Real data traces
   - Randomly generated data

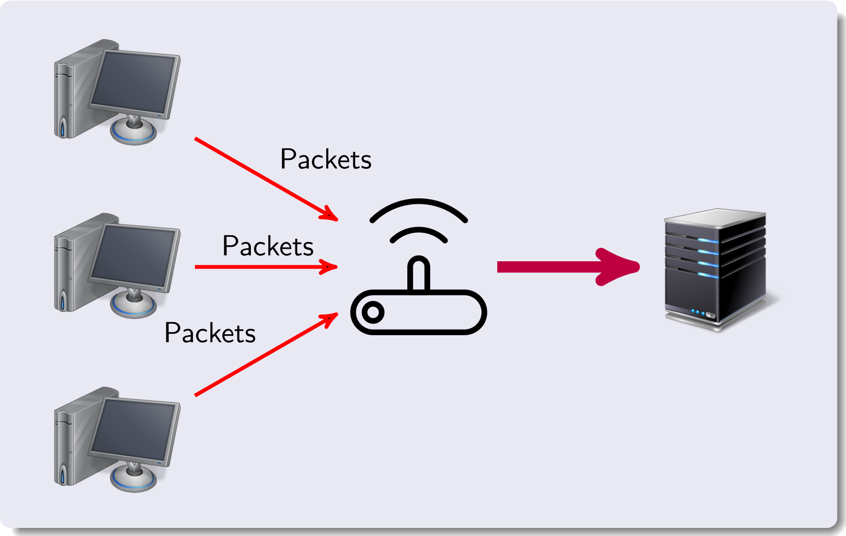
Outline

1. DDoS
   - Simple view
   - Multiple streams
2. Programming the algorithm
   - Extraction
   - Computing the codeviance
3. Experimental results
   - Real data traces
   - Randomly generated data

Simple attack

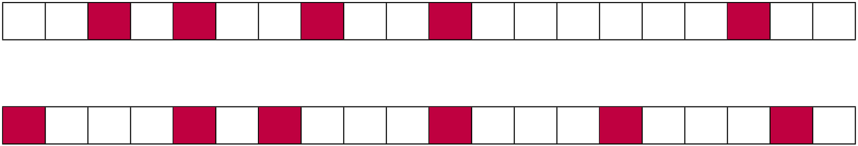


DDoS attack



Computing a correlation indicator

2 DATA STREAMS



Codeviance $\longrightarrow$ `if` High correlation `then` DDoS attack ongoing

Outline

1. DDoS
   - Simple view
   - Multiple streams
2. Programming the algorithm
   - Extraction
   - Computing the codeviance
3. Experimental results
   - Real data traces
   - Randomly generated data

Extraction on data traces

- Pick an entry on each line (=request), like the source or file requested ;
- convert them to integers using an injective function ;
- if multiple traces, extract all at once to keep correlation.

Example

GET /Consumer.html	$\longrightarrow$	1
GET /Consumer.html	$\longrightarrow$	1
GET /Consumer.html	$\longrightarrow$	1
GET /News.html	$\longrightarrow$	2
GET /icons/circle_logo_small.gif	$\longrightarrow$	3
GET /logos/small_gopher.gif	$\longrightarrow$	4
GET /logos/us-flag.gif	$\longrightarrow$	5
GET /logos/small_gopher.gif	$\longrightarrow$	4
GET /logos/small_ftp.gif	$\longrightarrow$	6
GET /icons/book.gif	$\longrightarrow$	7

Hashing functions

The data trace has integers values which stand between 0 and u.
We have to define t universal hashing functions h_i :

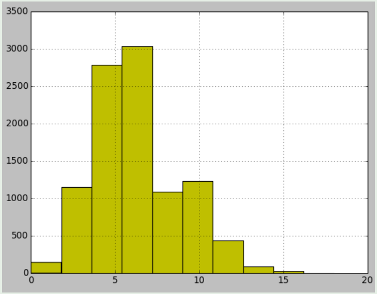
$$h_i(x) = ((a_i x + b_i) \mod u) \mod k \quad \forall i \in \{1 \dots t\}$$

where a_i, b_i are randomly generated in $\{1 \dots t-1\}$ for a_i, and $\{0 \dots t-1\}$ for b_i

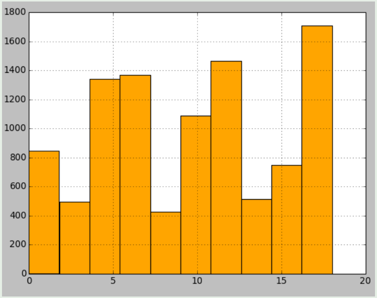
Hashing functions

Example

We generated a random data trace of size $10\ 000$, in $[0, u = 100]$ following a law of Poisson ($\lambda = u/2^4$). With $t = 7$, $k = 20$, $a = 6$, $b = 2$, here is the result :



Raw data trace



Hashed data trace

Using Python 3 language

`Matplotlib, Numpy`

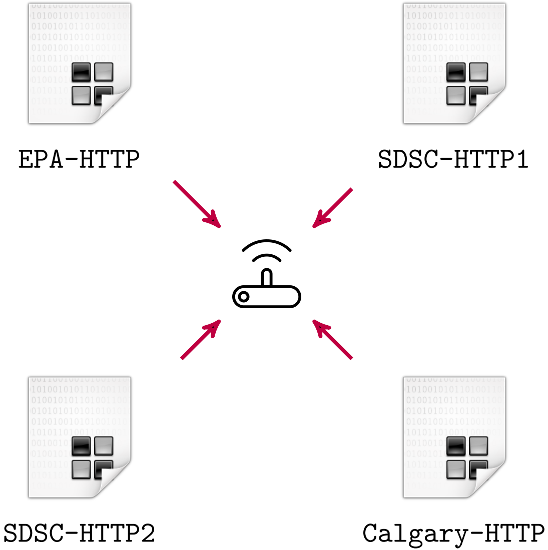
Reading files

Easy syntax

Huge standard library

Outline

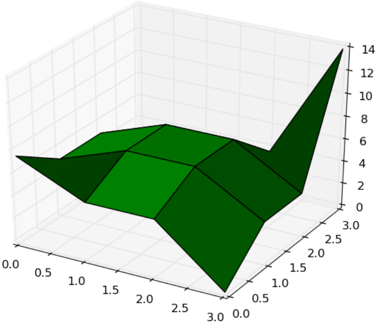
Real data traces



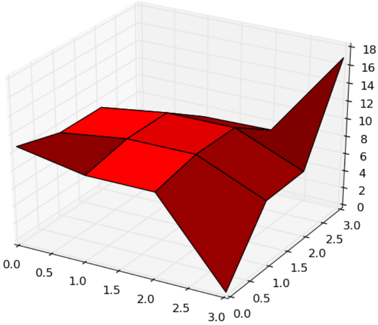
Real data traces

Codeviance matrix of real data traces

$$\varepsilon = 0.001 \quad | \quad \delta = 0.001$$



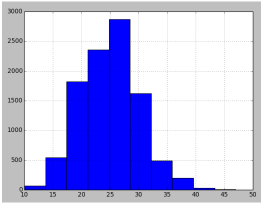
Exact codeviance



`SketchMin` algorithm

Generated data traces

size	interval $[0, u]$	ε	δ
10 000	$u = 100$	0.1	0.001

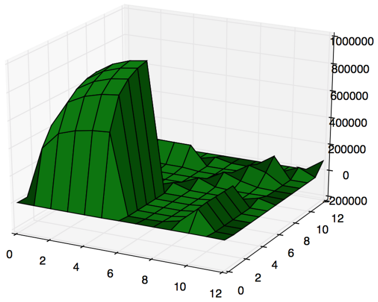


Histogram of trace 7

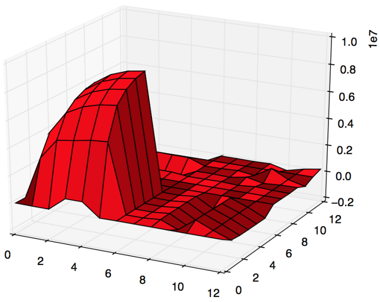
	Probabilistic laws	Parameters
Trace 0	Uniform	
Trace 1	Zipfian	$\alpha = 2$
Trace 2	Zipfian	$\alpha = 3$
Trace 3	Zipfian	$\alpha = 4$
Trace 4	Zipfian	$\alpha = 5$
Trace 5	Zipfian	$\alpha = 6$
Trace 6	Poisson	$\lambda = u/(2)$
Trace 7	Poisson	$\lambda = u/(2^2)$
Trace 8	Poisson	$\lambda = u/(2^3)$
Trace 9	Poisson	$\lambda = u/(2^4)$
Trace 10	Poisson	$\lambda = u/(2^5)$
Trace 11	Binomial	$p = 0.42$
Trace 12	Negative Binomial	$p = 0.42$

Generated data traces

Codeviance matrix of generated data traces



Exact codeviance



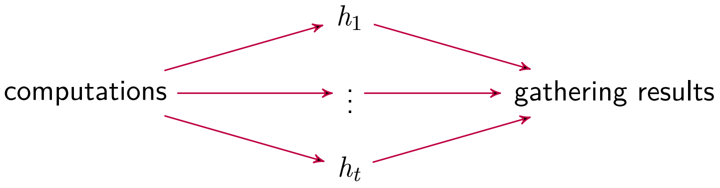
SketchMin algorithm

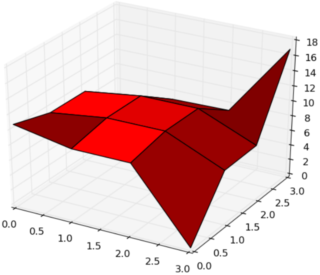
Conclusion

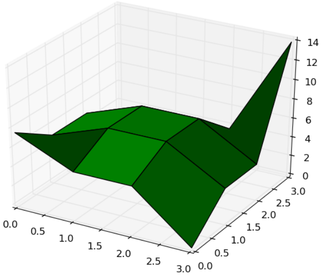
Algorithm results look the same as the exact entries, on a different scale. Can use the same method to detect attack on exact traces.

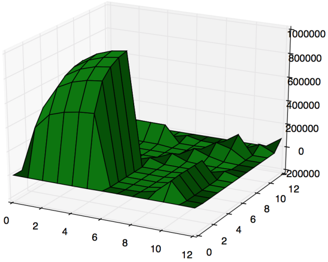
Improvements

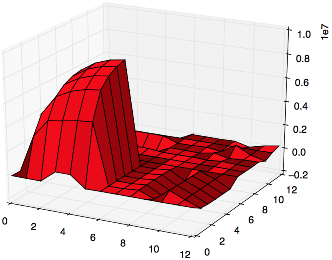
- Work on flows of data traces instead of only one complete trace ;
- put in concurrency the hashing functions computations ;
- find a way of detecting attacks without false-detection.




0 2 4 6 8 10 12 14 16 18
0.0 0.5 1.0 1.5 2.0 2.5 3.0
0.0 0.5 1.0 1.5 2.0 2.5 3.0


0 2 4 6 8 10 12 14
0.0 0.5 1.0 1.5 2.0 2.5 3.0
0.0 0.5 1.0 1.5 2.0 2.5 3.0


−200000 0 200000 400000 600000 800000 1000000
0 2 4 6 8 10 12
0 2 4 6 8 10 12


1e7
−0.2 0.0 0.2 0.4 0.6 0.8 1.0
0 2 4 6 8 10 12
0 2 4 6 8 10 12