Exscind: A Faster Pattern Matching For Intrusion Detection Using Exclusion and Inclusion Filters

Monther Aldwairi and Duaa Alansari

Seventh International Conference on Next Generation Web Services Practices (NWeSP'11)

19-21, October 2011

Salamanca, Spain

Introduction to IDS

- IDS inspects ingress/egress traffic for attack signatures/behaviour
  1. Drops the packet
  2. Alerts the system security administrator
  3. Logs malicious activity
- Anomaly-based IDS
  - ➢ Employs machine learning to profile, then classify traffic into either normal or abnormal behavior
  - \+ Detects new attacks
  - \- High false positives
  - \- Longer detection time
- Signature-based IDS (Misuse)
  - ➢ Employs exact pattern matching to look for specific signatures within packets
  - \+ Faster than anomaly but not enough for inline deployment
  - \+ Low false positives
  - \- But unable to detect new attacks
  - \- Manual signatures generation
  - +- Accuracy depends on signatures accuracy

The Problem


```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80
(msg:"WEB-IIS CodeRed v2 root.exe access";
content:"/root.exe"; sid:1256)
```

- Snort rules or attack signatures

Year	2003 [2]	2011
Rules	3000	11,416
Signatures	**1542**	**9945**

- **30%-60%** of Snort processing time → pattern matching [1]
- Moore's law! Internet speeds double every eighteen months
- Internet traffic is doubling every six months [4]
- Better Pattern Matching → Speed, Memory, Configurability

Contributions

1. Add a modified BF in front of modified WM to skip as much clean traffic as possible
   - Program ONLY the prefix (first n-gram) and store SIDs
   - Produce a probable matches list
   - Provide an index for the first match position
2. Modify the WM
   - Search ONLY the probable matches list
   - Start from the first index of probable match
3. Support all ASCII codes
   - Extended, special characters and the Null character
4. Vary the prefix 4-to -10

Wu-Manber Algorithm

- Preprocessing Stage
  - Find the minimum pattern length (m)
  - Choose Block size *B* (2 or 3)
  - Builds three hash tables: SHIFT table, HASH table, and PREFIX table
  - For any Block *x*, does it appear in any pattern?

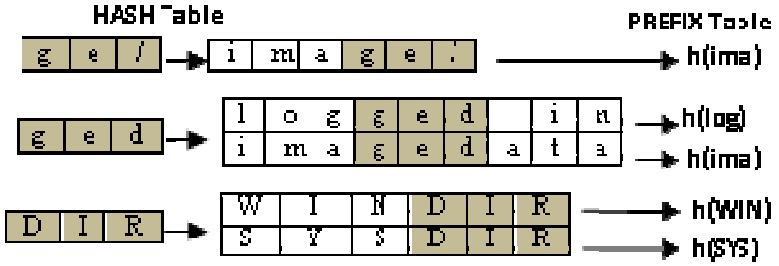
x ∈patterns?	**yes**	**No**
SHIFT [i]	*m-q*	*m-B+1*

    - *q* is the right most occurrence of *x* in a pattern
  - HASH and PREFIX tables contain linked lists of signatures with zero shift
- Search Stage
  - Slide a window, *w*, over the packet ∈ SHIFT Table
  - Hash last *B* chars of sliding window and access SHIFT table
    - *SHIFT[i]*>zero slide the window and repeat
    - If *SHIFT[i]*=zero, → search HASH and PREFIX tables to verify matches

Wu-Manber Example

Signature	SID
"image/"	2706
"logged in"	162
"imagedata"	12280
"WINDIR"	3010
"SYSDIR"	3011

Block	Shift	Block	Shift
mai	3	WIN	3
mag	2	IND	2
age	1	NDI	1
ge/	0	DIR	0
log	3	SYS	3
ogg	2	YSD	2
gge	1	SDI	1
ged	0	others	4



Step	z	t	i	m	a	g	e	/	l	k	S	Y	S	D	I	R	o	shift	output
1		z	t	i	m	a	g											2	
2				i	m	a	g	e	/									0	image/
3						m	a	g	e	/	l							4	
4									/	l	k	S	Y	S				3	
5												S	Y	S	D	I	R	0	SYSDIR
6													Y	S	D	I	R	4	

Related Work

- Mike Fisk et al. proposed a new hybrid system for pattern matching for IDS. This system provide a combination of Boyer-Moore algorithm and Aho-Corasick algorithm as a hybrid system
- Kostas Anagnostakis et al. proposed a new exclusion filter for IDS called *E²xB*. This stage marks the corresponding cell on the (256-cell) map for each character appearing in the text. Then, the decision that the text contains a character will be only if the corresponding cell of the map is marked with the num of the packet
- Sarang Dharmapurikar et al. proposed a new hardware-based technique for Bloom filters. They divided the signatures into groups according to their length and stored each group in a unique Bloom filter. Each Bloom filter used an analyzer for probing detected strings to determine whether it is a false positive or not
- Haoyu Song et al. proposed Extended Bloom Filter (EBF) to implement multi-pattern signature matching. They introduced a new technique to support long signature matching to minimize the maintained signature lengths. They embedded multi-port memories in FPGAs

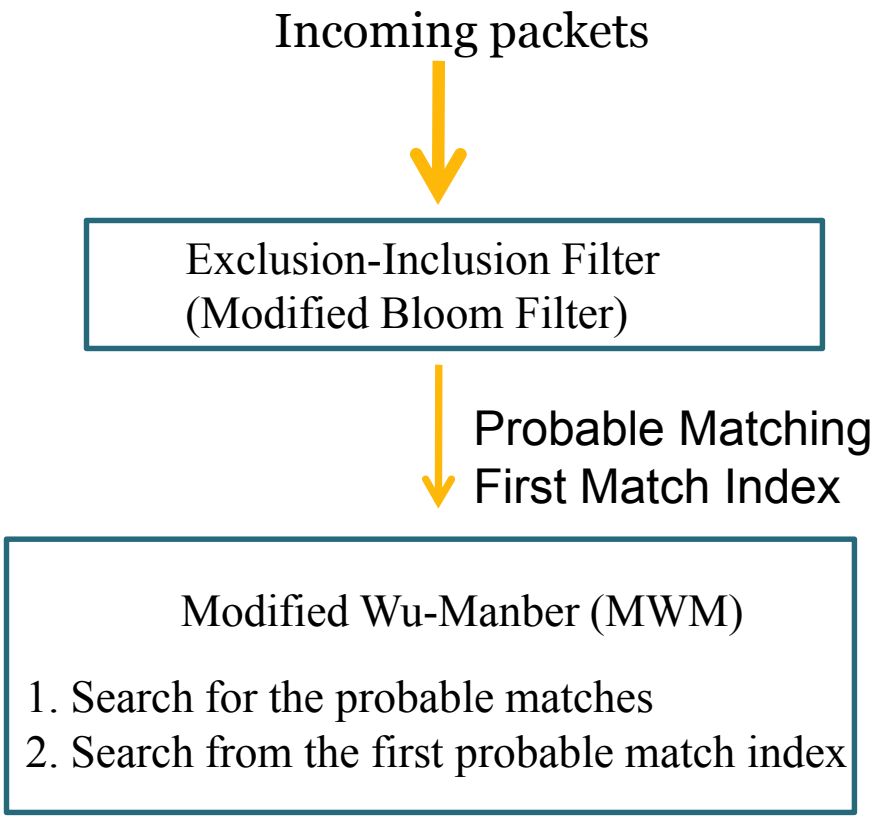
Outline

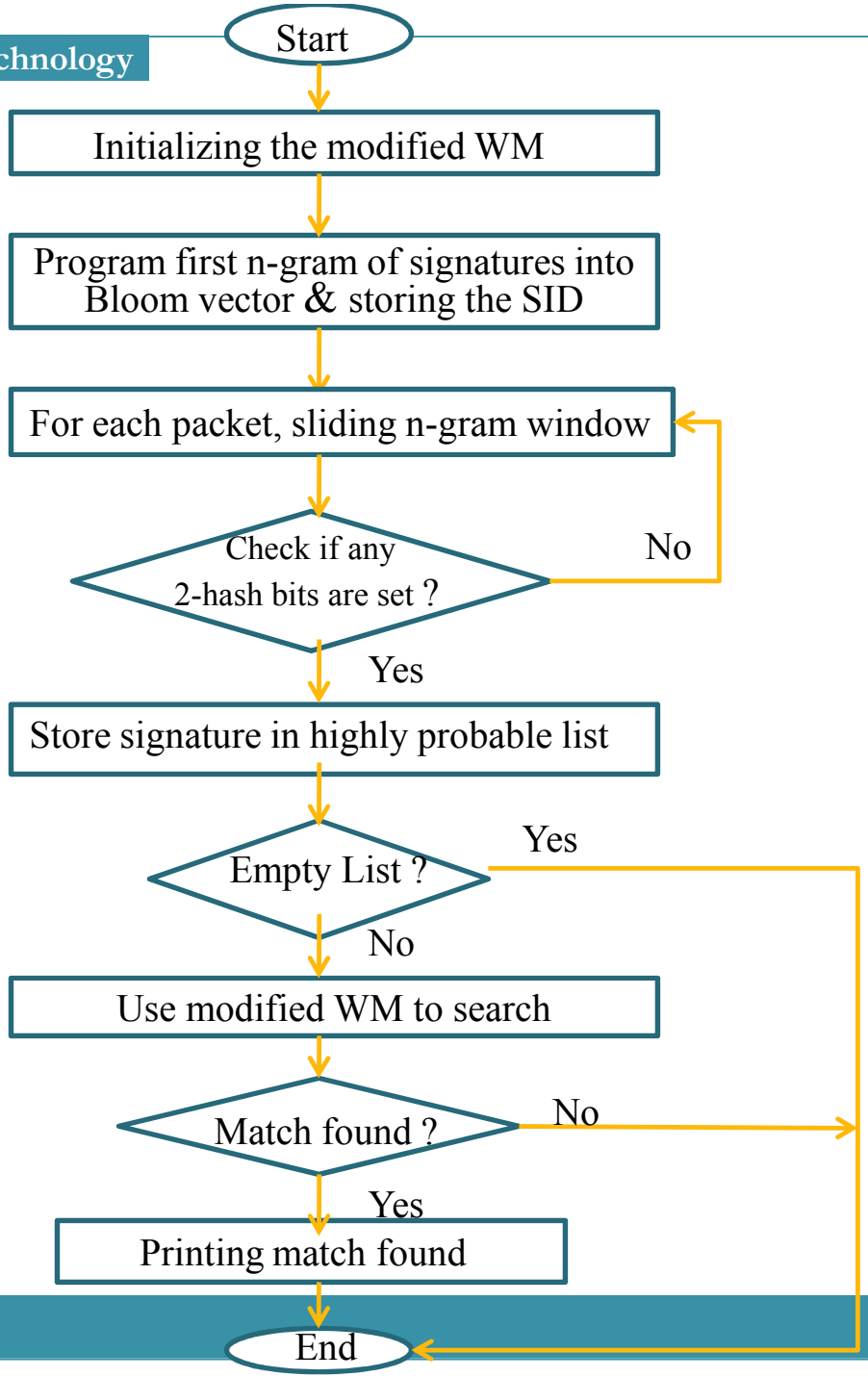
- Introduction to Intrusion Detection
- The Problem
- Related Work
- Exscind: The Big Picture
- Bloom Filters
- Exscind Example
- Experimental Evaluation
- Conclusions

Exscind the Big picture

- We propose Exscind: meaning to exclude from the union
  - Exscind is an exclusion-inclusion intrusion detection
  - It can exclude as many clean packets as possible
  - Modified Exclusion-Inclusion Bloom filter using only 4-gram
    - Produces a probable matches list
    - Provides an index for the first match position
  - Modifies the WM
    - Search ONLY the probable matches list
    - Start from the first index of probable match
    - Support all ASCII codes
    - Extended, special characters and the Null character

Exscind the Big picture




Start
Initializing the modified WM
Program first n-gram of signatures into Bloom vector & storing the SID
For each packet, sliding n-gram window
Check if any 2-hash bits are set ?
No
Yes
Store signature in highly probable list
Empty List ?
Yes
No
Use modified WM to search
Match found ?
No
Yes
Printing match found
End

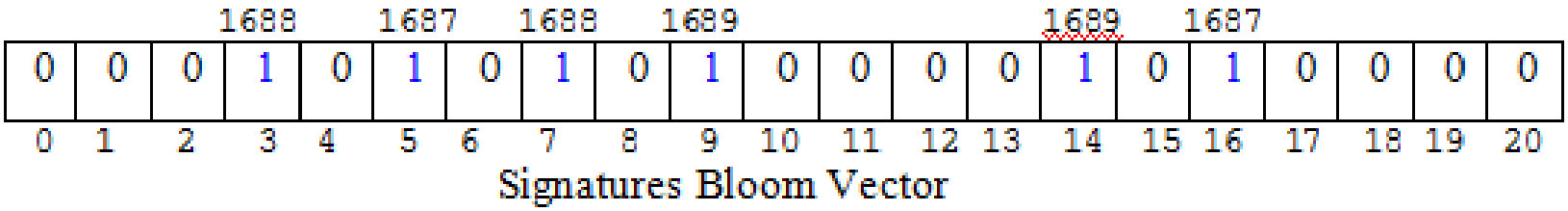
Bloom Filters

- Exclusion filter through approximate membership queries on a set of strings
  - Bloom Vector: array vector of *m* bits
  - Programming: use *k* hash functions on signatures and set the corresponding bits
  - Querying: Compute the same *k* hash functions on the packet, then check corresponding hash bits?
    - If at least one bit is not set, 100% certainty the signature does not belong
    - If all *k* bits are set, the signature might belong to the set with a probability
  - Assume the signature "cmd.exe" hashes to 2 and 8
  - Set bits 2 and 8 in a 10-bit Bloom filter

Exscind Example

1. Initialize modified WM with all Snort signatures
2. Program the Bloom vector with the 4-gram prefix of all Snort signatures
3. Store the SID of each 4-gram corresponding to set bits

Serial	SID	Signature	Hash Bits
3135	1687	dba_tables	5, 16
3136	1688	user_tablespace	3, 7
3137	1689	sys.all_users	9, 14



Signatures Bloom Vector

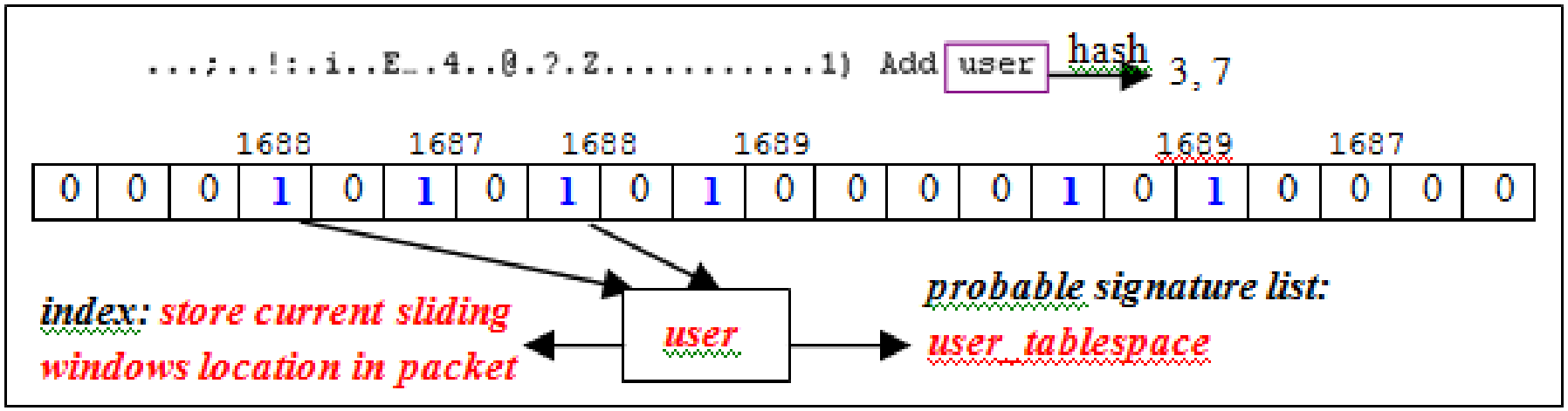
Exscind Example

4. Slide a 4-gram window throughout the packet
5. Computes the same two hash functions

Sliding Window Moving Throughout the Packet	Hash Bits
...;..!:.i..E...4..@.?.Z.....1) Add user	0, 8
..;..!:.i..E...4..@.?.Z.....1) Add user	4, 12
..;...!:.i..E...4..@.?.Z.....1) Add user	10, 8
...;...!:.i..E...4..@.?.Z.....1) Add user	14, 17
.	.
.	.
...;..!:.i..E...4..@.?.Z.....1) Add user	3, 7

Exscind Example

4. Check the hash bits of each window against the signatures Bloom vector.
5. If they are set, this means that there is a probable match
6. The system retrieves the stored SIDs and adds the signatures to the probable signatures list.



Exscind Example

8. Exscind checks the probable signatures list
   a. If empty → packet is clean → 100% certainty it has no attack signatures →skip pattern matching

   b. If not empty → packet is suspicious → modified WM → starting from first position of probable match →search only the probable signatures

   c. If any match occurs within the packet, the system considers malicious packet and records the number of matches occurrences along with the index of each occurrence

Exscind Savings and Overhead

- Exscind reduces execution time
  - Exclusion-Inclusion skips pattern matching for clean packets
  - MWM searches suspicious packets ONLY for small probable signatures matching list
  - MWM search starts from the index for the first probable match
- Exscind over head on memory and time
  - Time for BF programming added to preprocessing
  - Time for BF querying
  - Memory space for Bloom vector and the of signatures SIDs

Outline

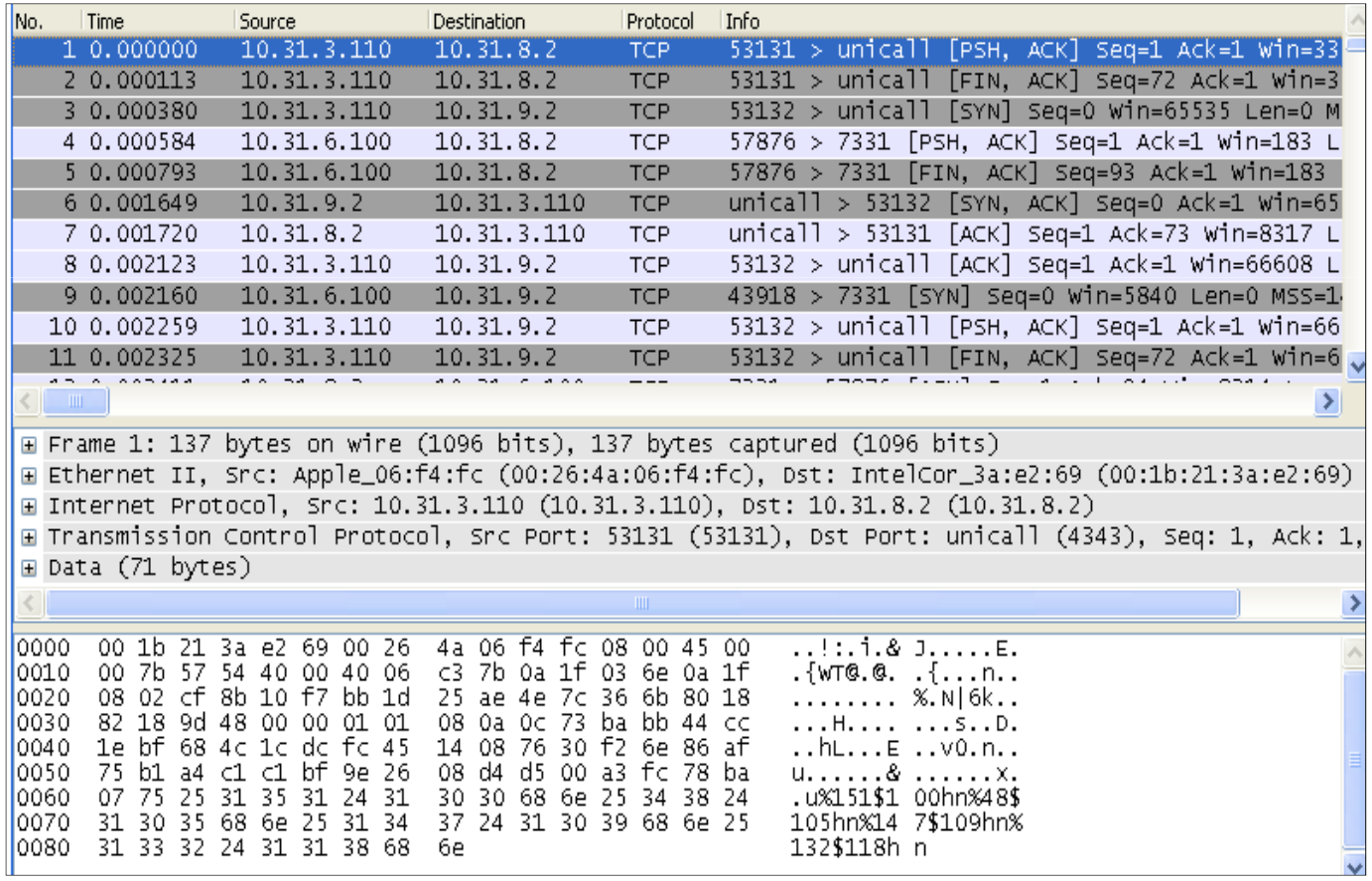
- Introduction to Intrusion Detection
- The Problem
- Related Work
- Exscind: The Big Picture
- Bloom Filters
- Exscind Example
- Experimental Evaluation
- Conclusions

Experimental Setup and Metrics

- Metrics
  1. Average execution time
  2. Average memory usage
  3. WMSC (Wu-Manber Skip Counter) means the number of clean packets skipped without pattern matching
- Experimental Environment
  - Workstation: 3 GHz Intel Core ™ 2, 4 GB of RAM
  - execution time ➔ ExecutionStopWatch class
  - memory usage ➔ Garbage Collection class
  - 20 experiments average numbers reported

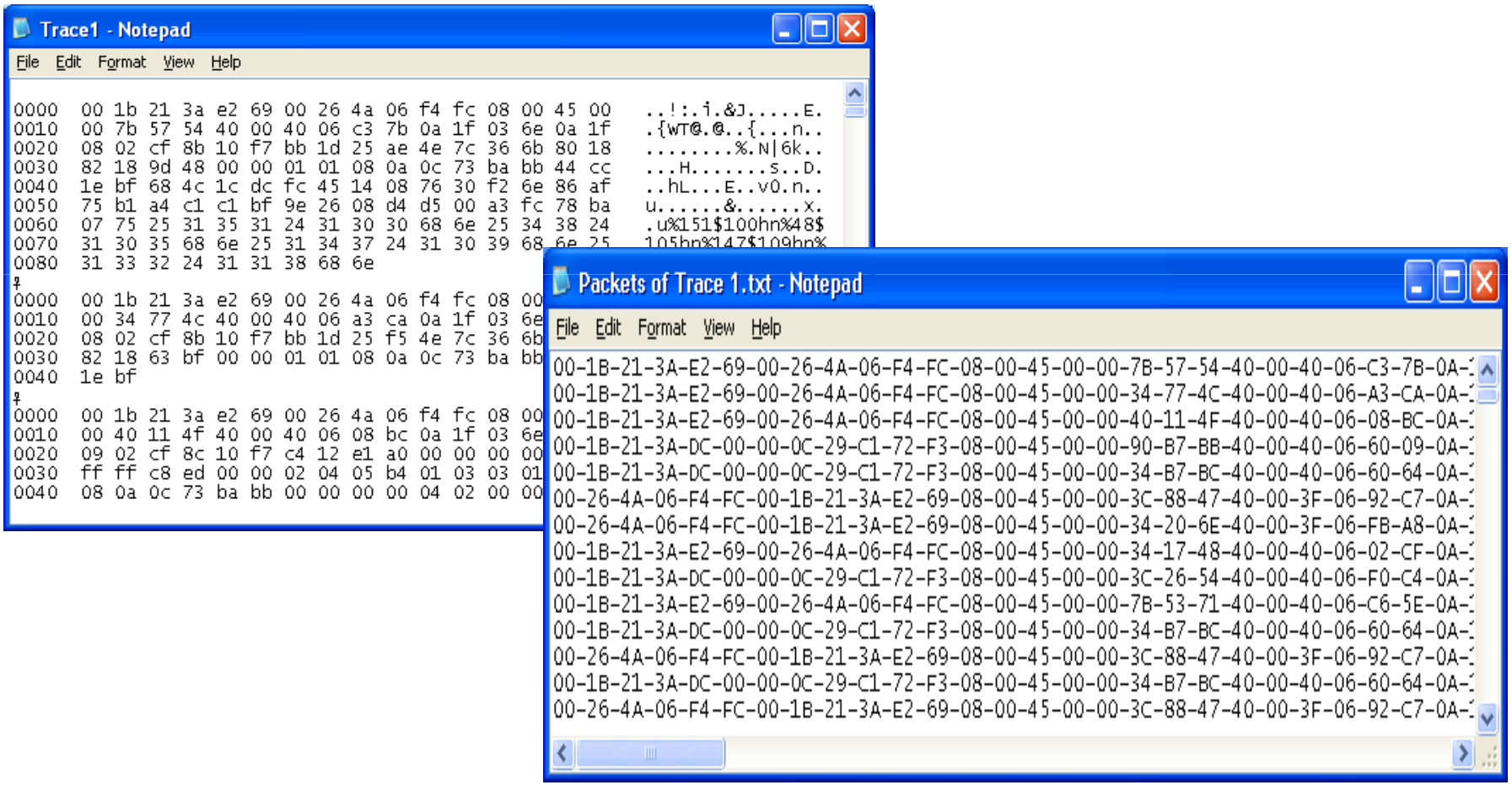
Export Packet Contents

- Wireshark Network Protocol Analyzer v1.4.4



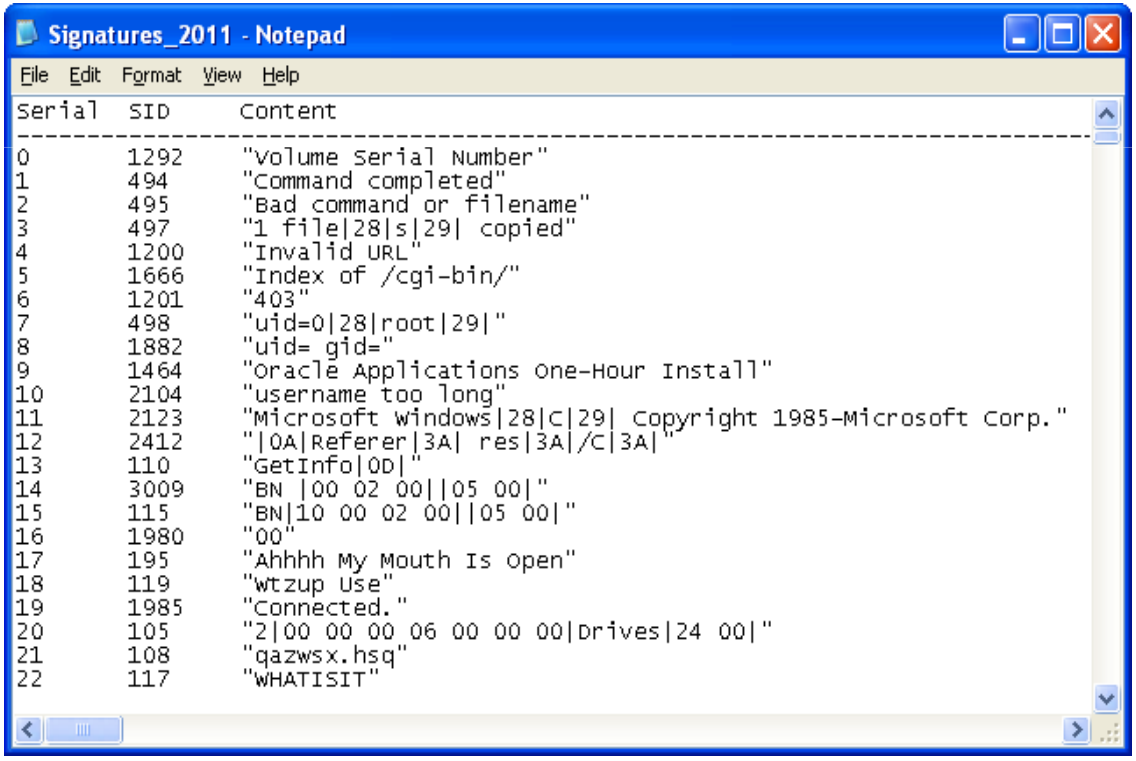
Packets Extraction

- C# utility to parse payload



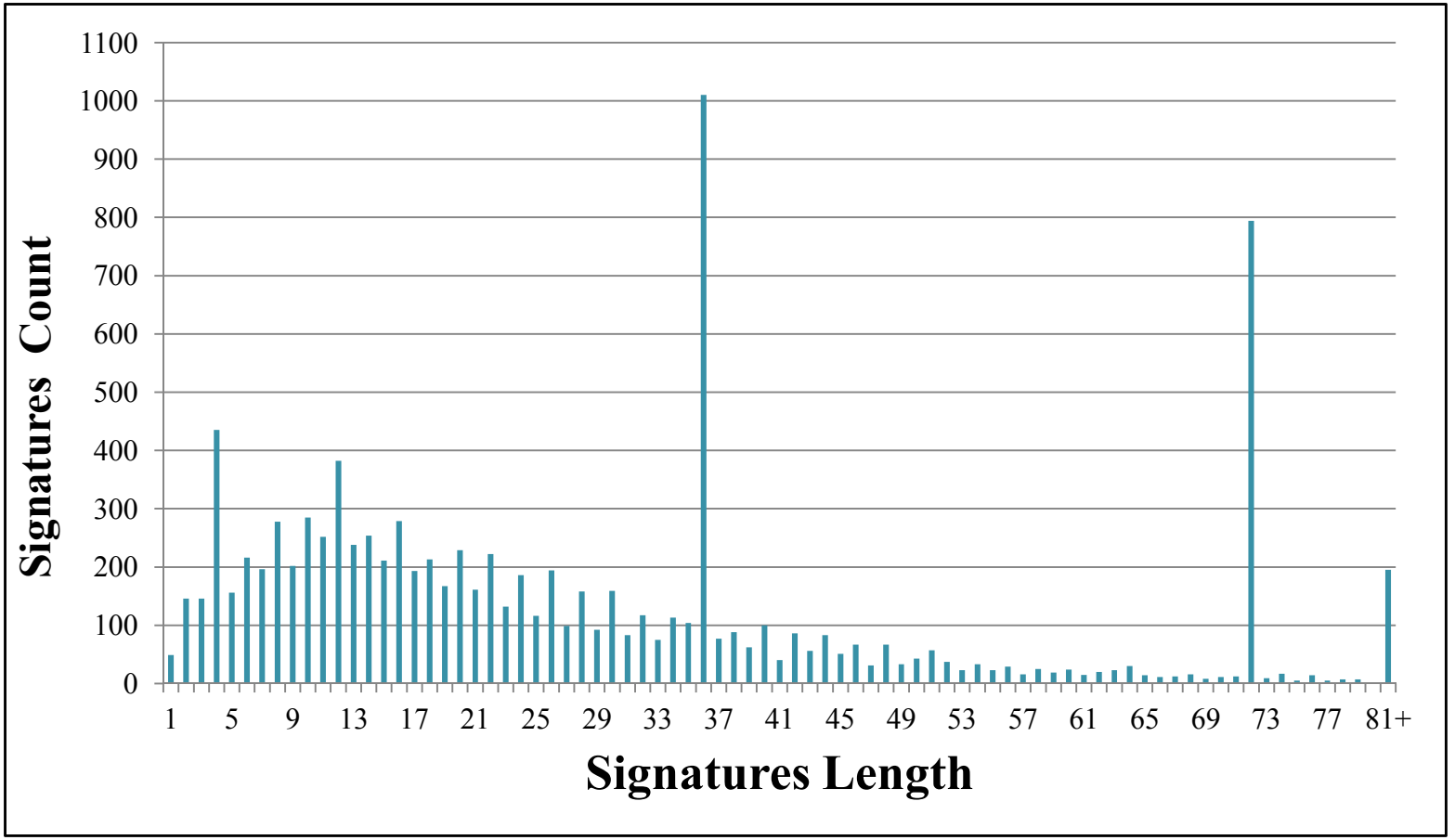
Signatures Extraction

- Snort 2.9.0.4 rules released in March 2011
- 56 rules files → C# to extract SIDs, content and uricontent → concatenate same rule signatures with zero space



Signatures Analysis

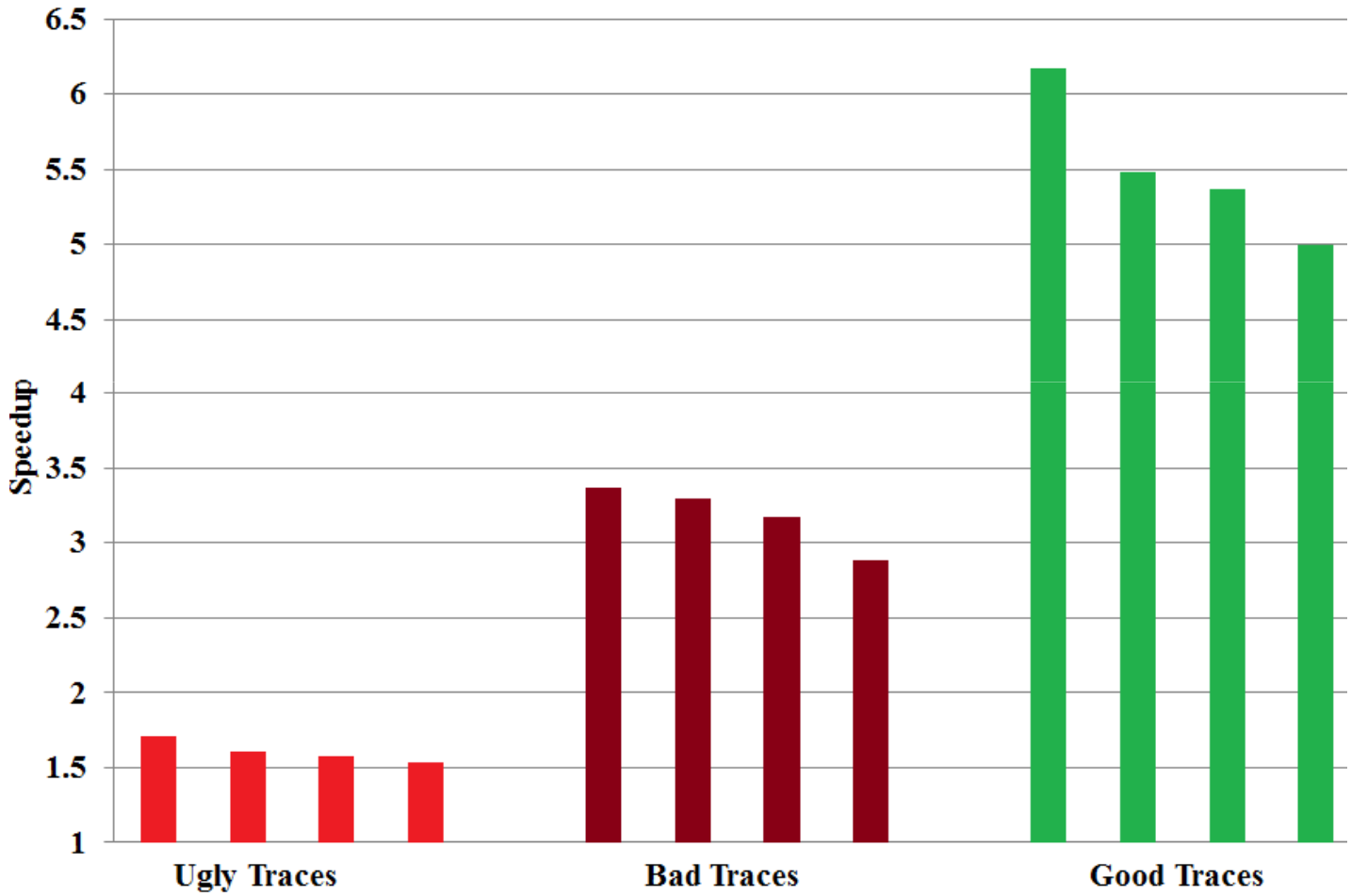
- 9,945 signatures



Malicious Traffic Analysis

Ugly Trace	Total Packets	Malicious Packets	Malicious Packets (%)	WM Time	Exscind Time	Speed up
8	671116	258556	38.5	25.74	15.09	1.71
57	209188	86129	41.2	15.29	9.53	1.60
51	299713	129763	43.3	15.02	9.50	1.58
58	268000	119627	44.6	11.37	7.42	1.53
Bad Trace	**Total Packets**	**Malicious Packets**	**Malicious Packets (%)**	**WM Time**	**Exscind Time**	**Speed up**
1	688158	109311	15.9	75.44	23.00	3.28
22	659365	110363	16.7	76.22	22.90	3.33
0	771382	139866	18.1	66.18	20.87	3.17
2	642091	130901	20.4	55.04	19.11	2.88
Good Trace	**Total Packets**	**Malicious Packets**	**Malicious Packets (%)**	**WM Time**	**Exscind Time**	**Speed up**
VA	3606	54	1	105	17	6.18
GD	9354	175	2	345	63	5.48
LC	29474	1031	3	698	130	5.37
HM	2817	126	4	90	18	5.00

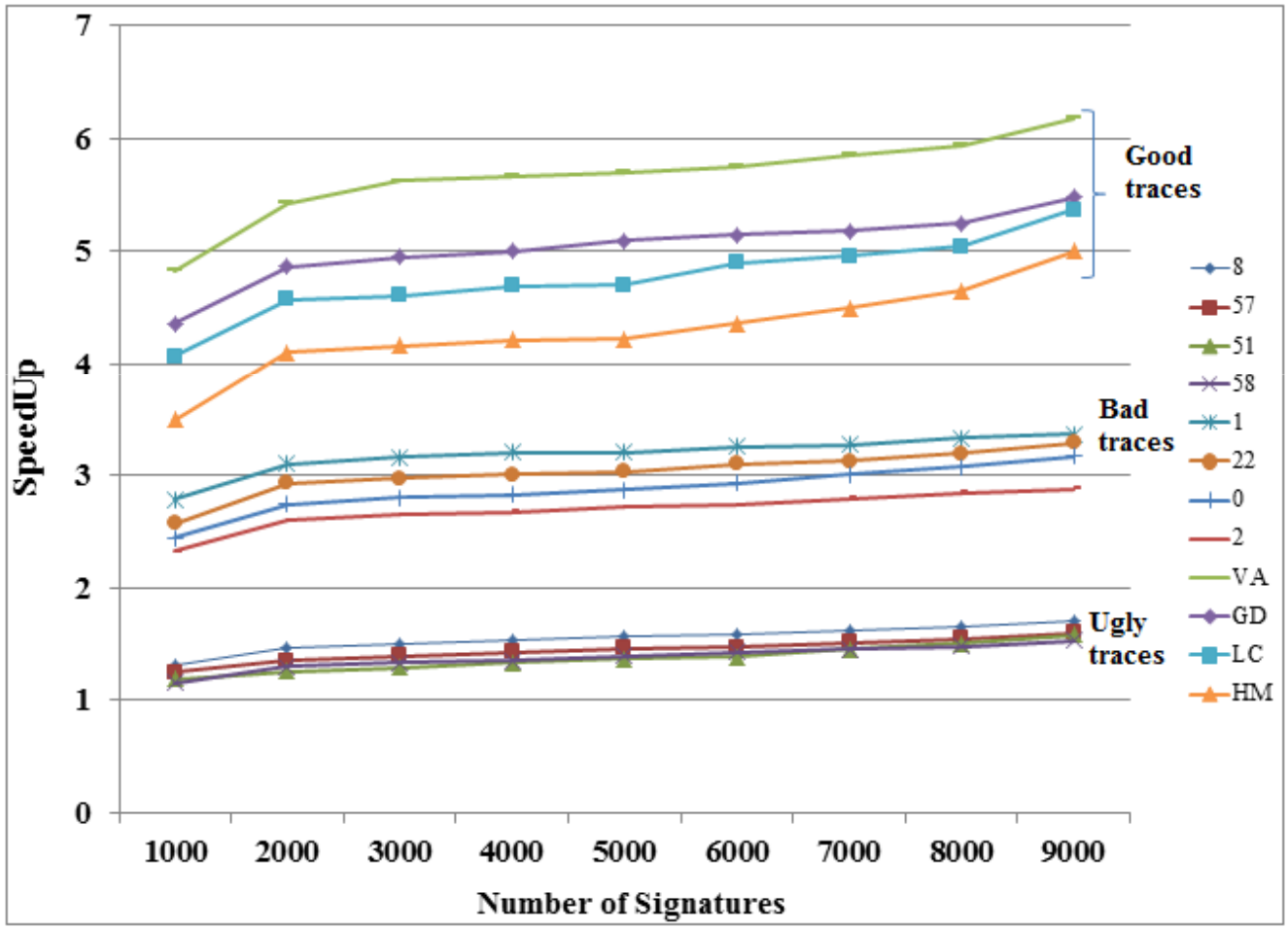
Speedup


Speedup
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
Ugly Traces
Bad Traces
Good Traces

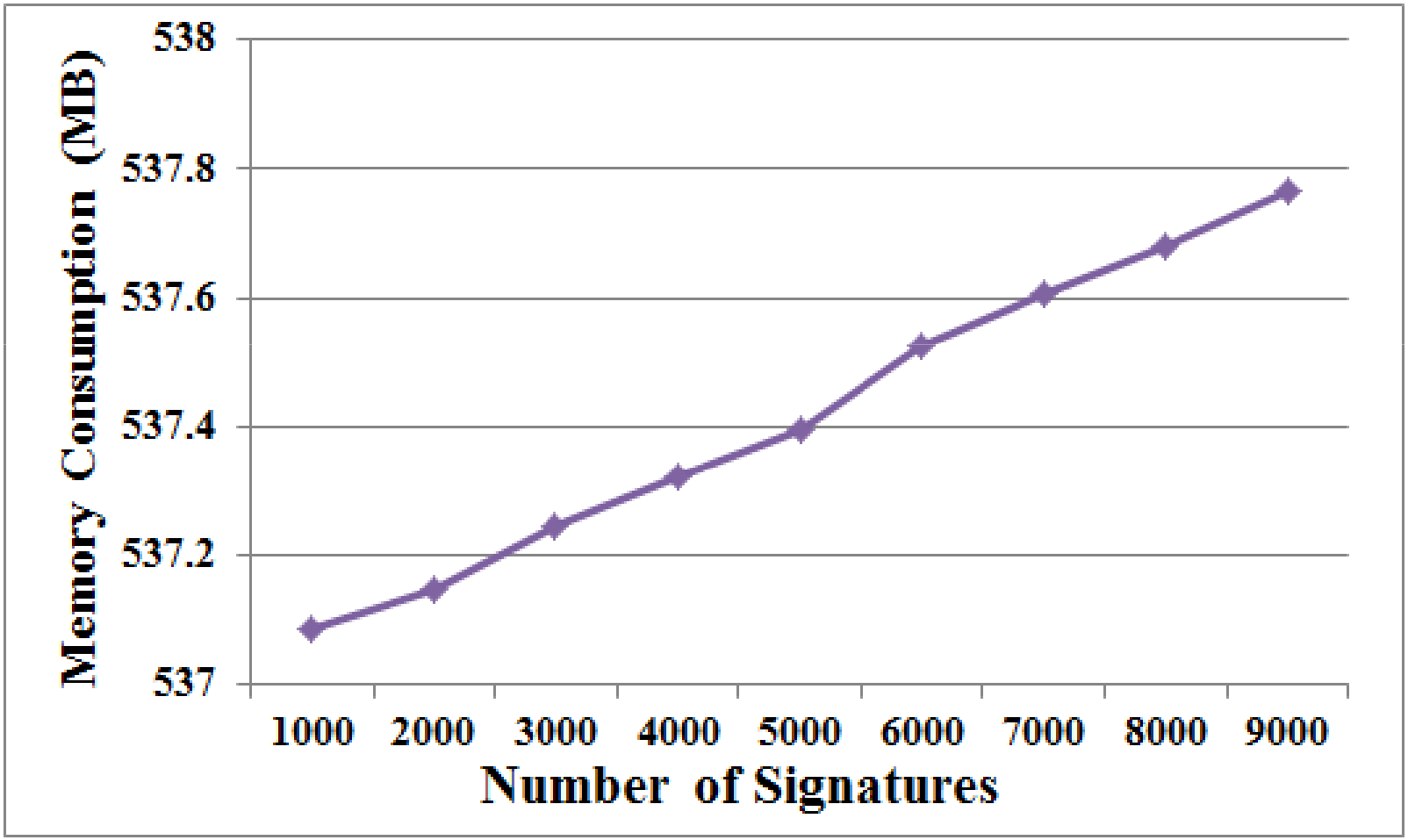
Memory in KBs

WM Memory	Our Memory	Added Overhead	Percentage overhead (%)
537,178	537,766.1	588.1	1.095

Speedup Scaling



Memory Scaling



Conclusions

1. Average speedup of 3.4 times
2. Normal traffic speedup > 6 times
3. Overhead: 0.11% increase in memory usage
4. Speedup is almost constant and undiminished with increasing number of signatures
5. Memory usage increases linearly

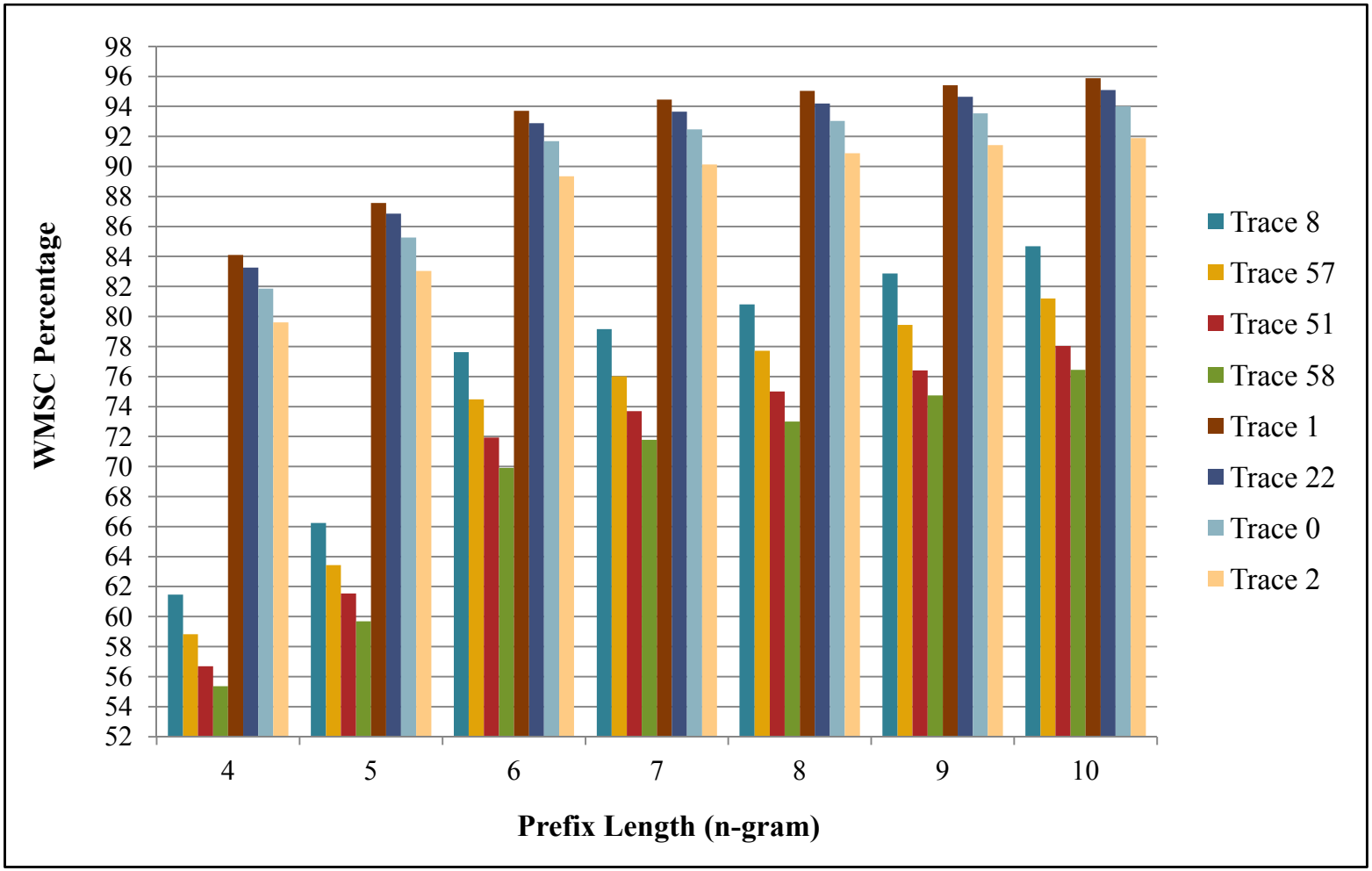
Qs

Signatures Collisions

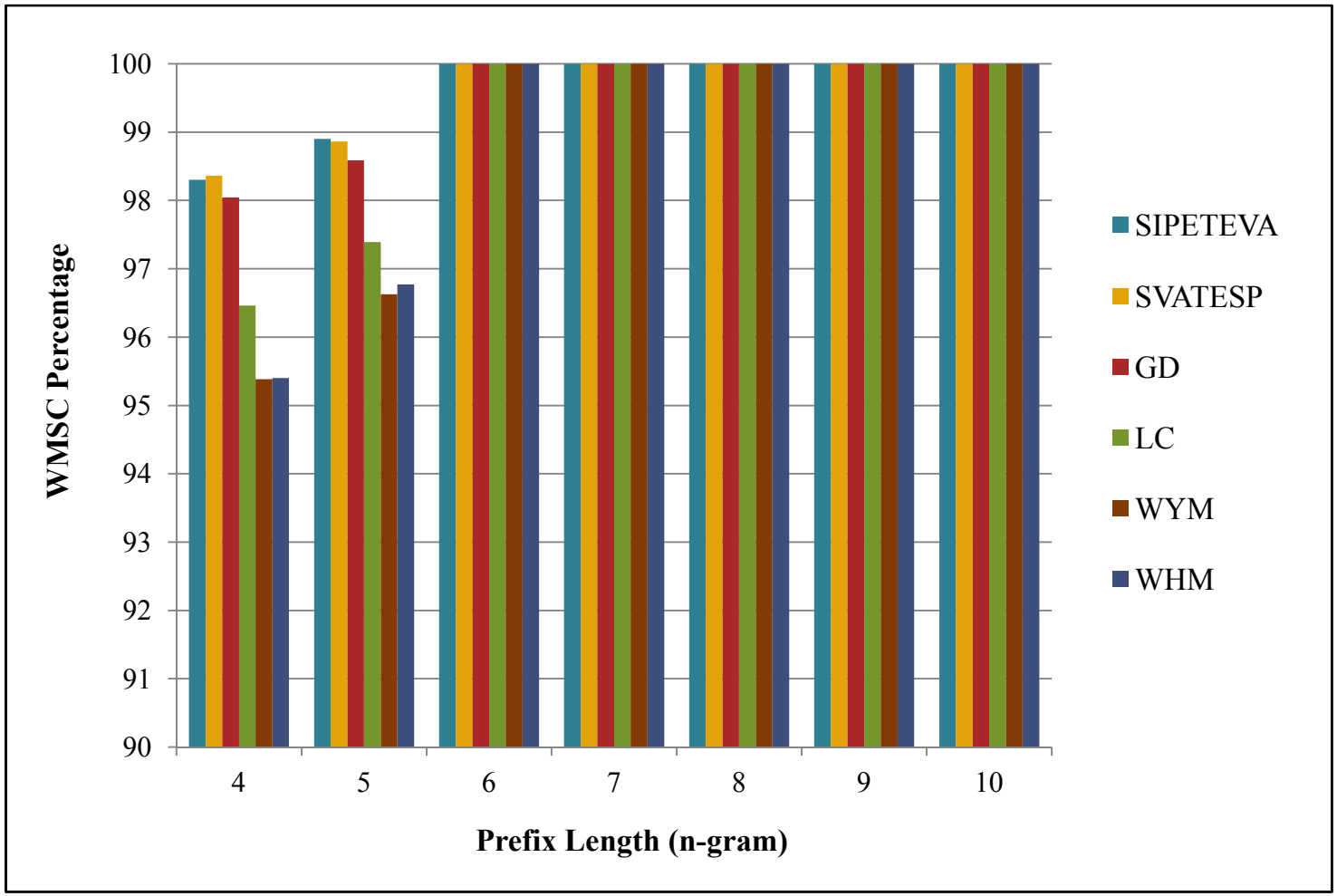
- Collision are the number of signatures with same hash bits!
- Bloom vector length 700,000 bit

Min Signature Length (n-gram)	Num of Signatures	n-grams of Signatures	Bloom Collisions	Bloom Collisions Percentage
4	9605	5017	1	0.000199322
5	9170	5662	1	0.000176616
6	9014	5934	1	0.00016852
7	8798	6080	0	0
8	8602	6193	0	0
9	8324	6208	0	0
10	8122	6277	0	0

WMSC Worst Case Performance

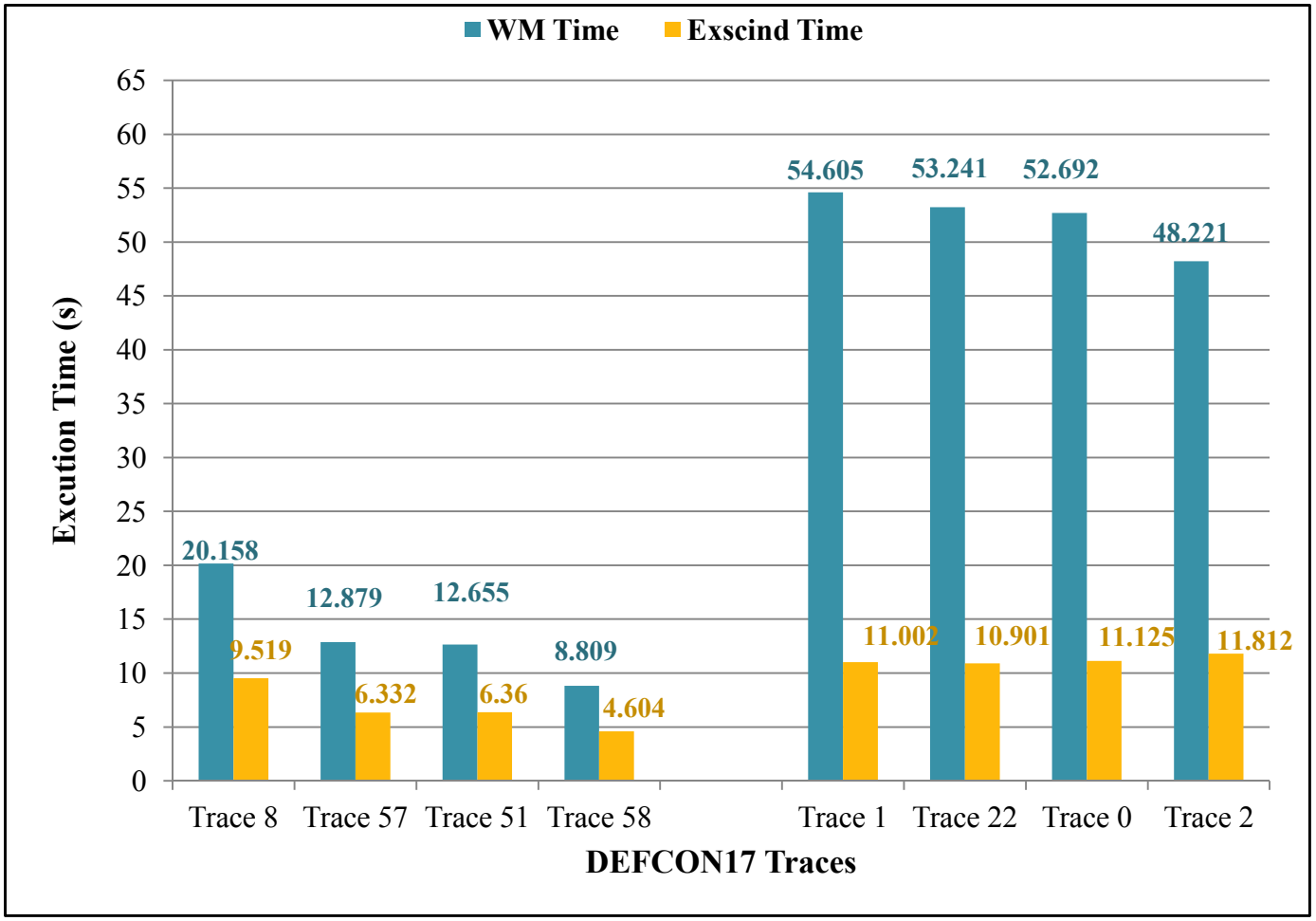


WMSC Best Case Performance



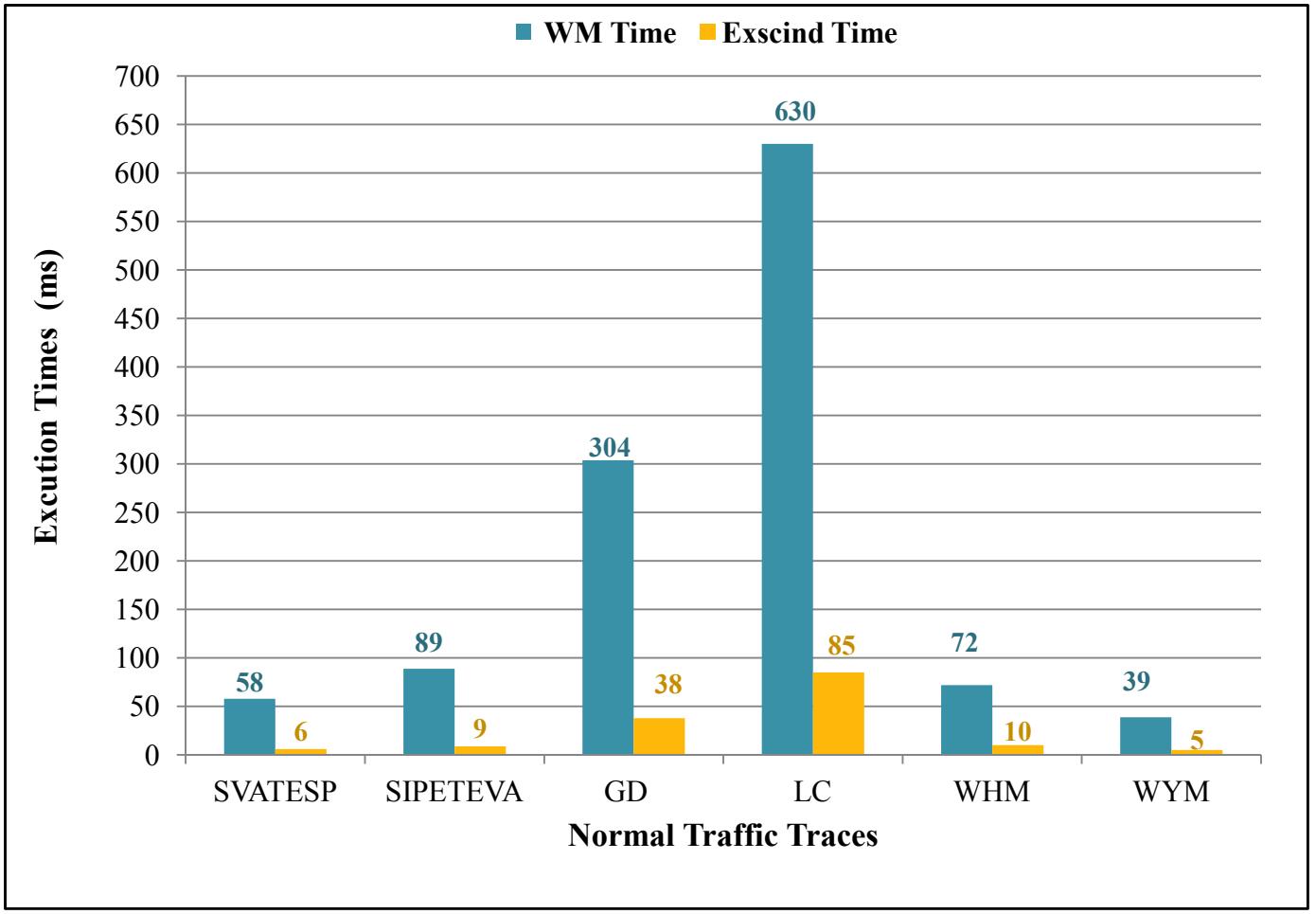
Worst Case Execution Time Comparison

- For n= 6



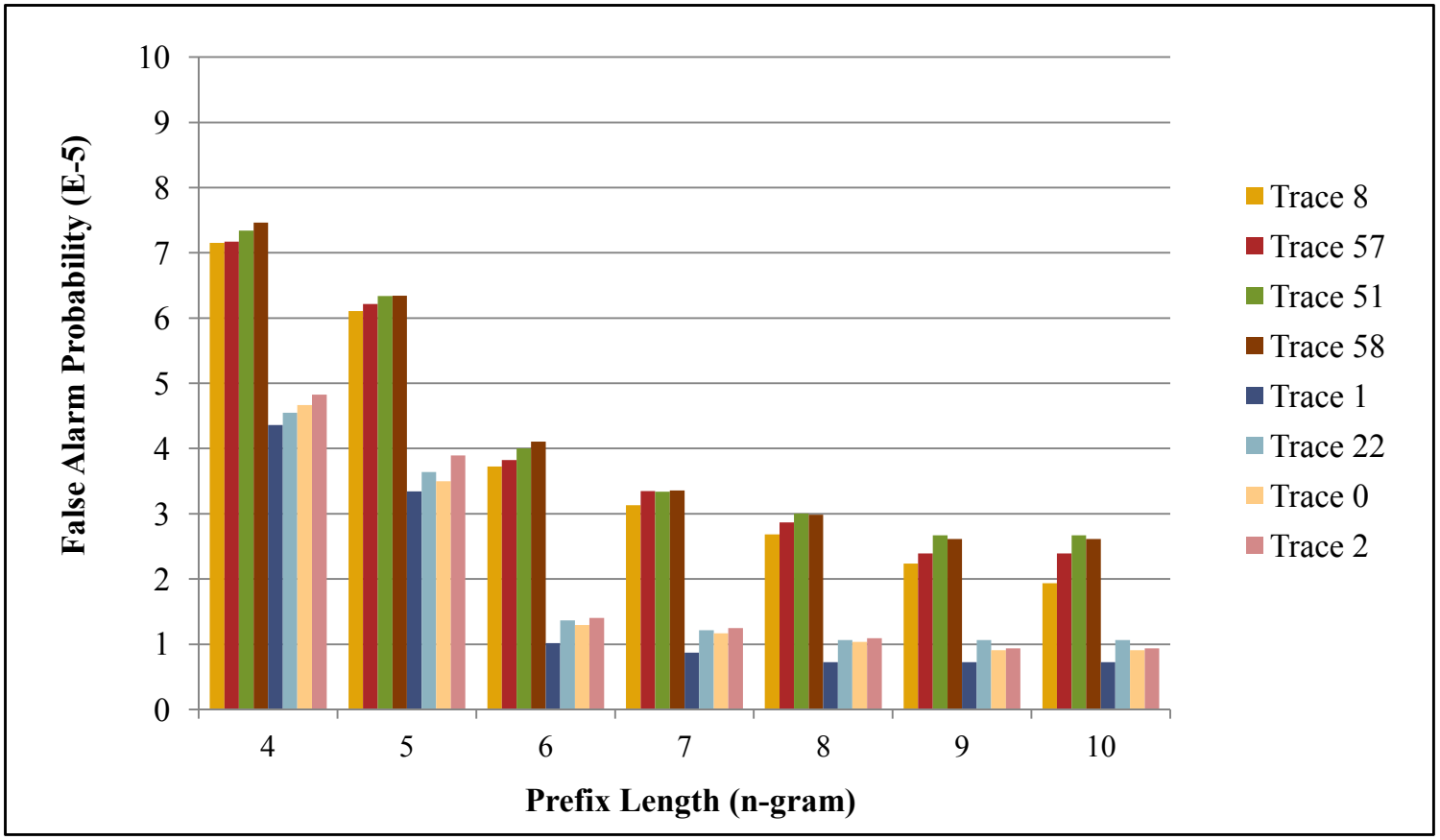
Best Case Execution Time Comparison

- For n= 6



Worst Case False Alarms Probability

Additional pattern matching performed due to the fact the BF only inspects n-grams



Hash Functions

1. Rolling hash: it is used for n-gram sliding windows move throughout a text. There are two popular rolling functions: Cyclic Polynomial and Rabin-Karp. We use Cyclic Polynomial because it is faster than Rabin-Karp

2. SAX hash: it is also called Shift-add-XOR, because it combines two functions; addition and XOR with a shift operation. It works as follows for each character in the string:

   SAX hash = (hash << 5) + (hash >> 2) + character