Inferring Protocol State Machine from Network Traces: A Probabilistic Approach

Yipeng Wang, ***Zhibin Zhang***, Danfeng(Daphne) Yao, Buyun Qu, Li Guo
Institute of Computing Technology, CAS
Virginia Tech, USA

Agenda

- Motivation
- Challenges
- Architecture of *Veritas*
- Packet Analysis
- State Message Inference
- State Machine Inference
- Experimental Evaluation
- Conclusions
- Future Works

Motivation

- Protocol specs is useful in many security applications
  - Traffic classification
  - IDS & DPI
  - Botnet detection
- Previous works are major in reverse engineering
  - Time consuming & Error prune
  - Codes are not always available
- Problem: Can we infer protocol specs from traffic automatically, if they are not encrypted?

Challenges

- How to discover protocol keywords (most frequent strings) from traffic traces?
  - Traffic classification needs keywords to label flows
- Naïve solution: sequence alignment algorithms (Needlemen-Wuch, DTW…)
  - Not scalable
  - Can not handle multiple keywords
- Our solution: K-S Filtering

Example

MAIL FROM: < alice@gmail.com>

MAIL FROM: < jason@hotmail.com >

MAIL FROM: < bob@live.cn>

Conclusion

MAIL FROM: <variable>

Example

MAIL FROM: < alice@gmail.com>

HELO jason

250 OK

Conclusion

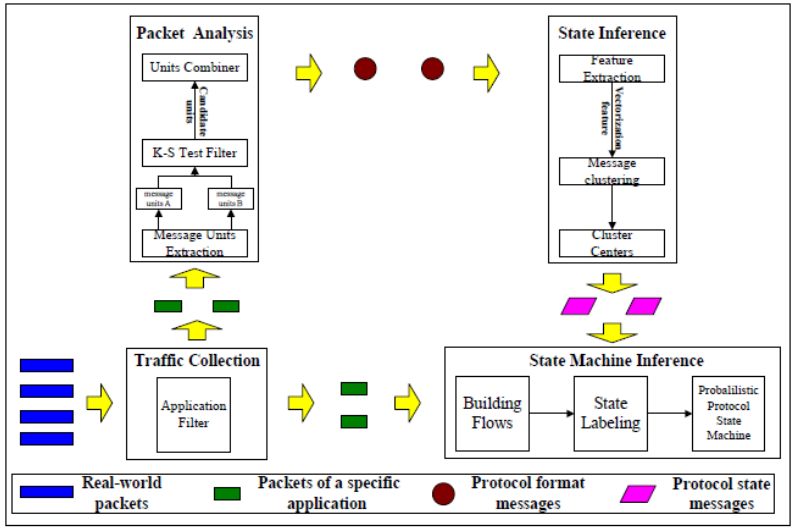
??

Challenges

- How to obtain protocol state machine from traffic traces?
  - State machine is the model of protocol grammar
    - Botnet behavior description
  - What is a state
  - How many states
  - State labeling within a flow
- Our solution: Clustering + P-PSM

Architecture of *Veritas*

- Assumption 1: Traffic is not encrypted
  - For performance concern many applications do not encrypted their traffic
- Assumption 2 (Single-Protocol Inference): The trace is only composed of flows from the protocol to be investigated
  - Single-protocol inference can be basic work for multiple case

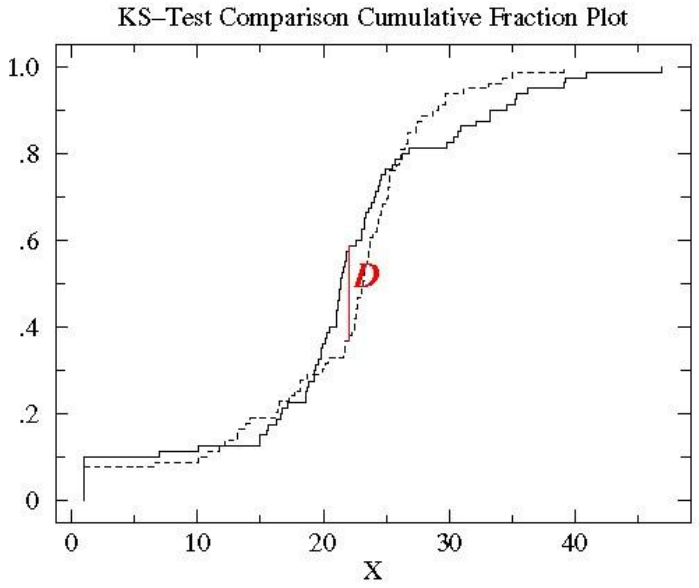


Packet Analysis

- Message Unit Extraction
  - Break flow data into subsequences
  - Count the frequency precisely
- Problems
  - Length of subsequences: l
  - Length of protocol-related sequences: n

K-S Test Filtering

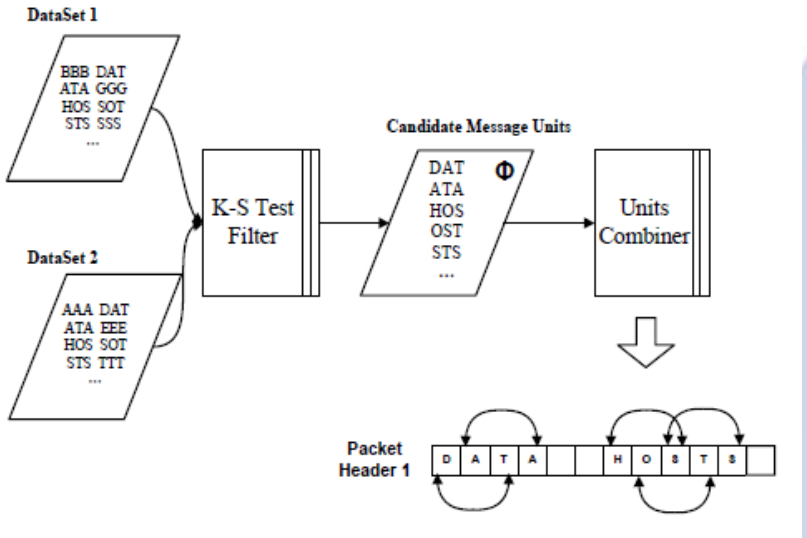
- What is K-S test
  - Try to determine if two datasets differ significantly
  - Making no assumption about the distribution of data
- How to do K-S test
  - Building CDF for A & B
  - Building K-S statistic & test



$$D_{n,n'} = \sup_x |F_n(x) - F_{n'}(x)|$$

Protocol Format Messages Inference

- Choosing candidate units with K-S filter
  - Select nontrivial units from noises
- Combine message units
  - Link nontrivial units
  - Get protocol format messages



Protocol State Message Inference

	(A	D	E	H	L	O	T)
a = HELO	(0	0	1	1	1	1	0)
b = EHLO	(0	0	1	1	1	1	0)

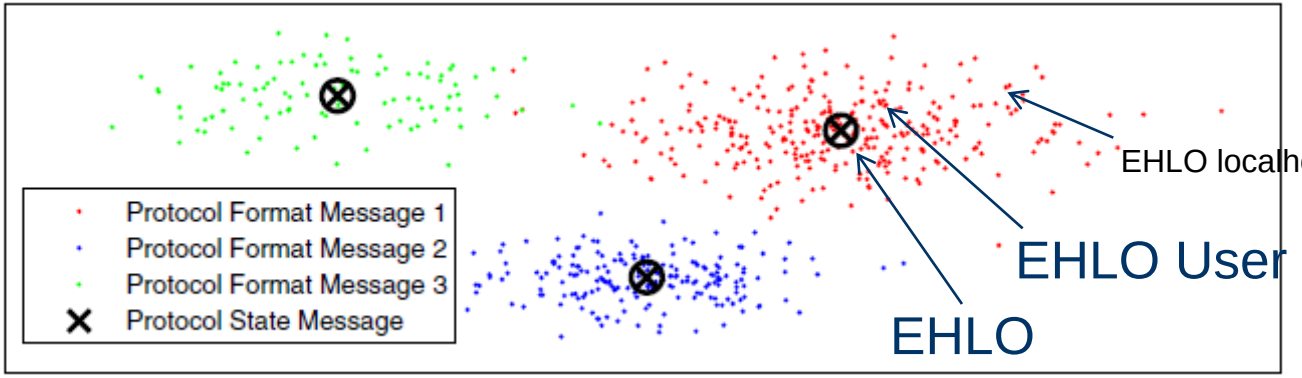
D(a,b) = 1 – J(a, b) = 1 – 4/4 = 0

	(A	D	E	H	L	O	T)
a = HELO	(0	0	1	1	1	1	0)
c = DATA	(2	1	0	0	0	0	1)

D(a,c) = 1- J(a, c) = 1 - 0/7 = 1

- What is a state?
  - Intuition: Messages with same format share the same interpretations
  - State: Message with distinguishable format
  - Example:
    EHLO EHLO User EHLO localhost.localdomain
- How to distinguish a state from others
  - Jaccard index based distance

Protocol State Message Inference



- Using clustering to distinguish states
  - Using Medoid algorithm to cluster
  - How many states (How many clusters): k
    - Using Dunn index to measure clustering quality and get the best k
  - Labeling each packet with a state according to its cluster center

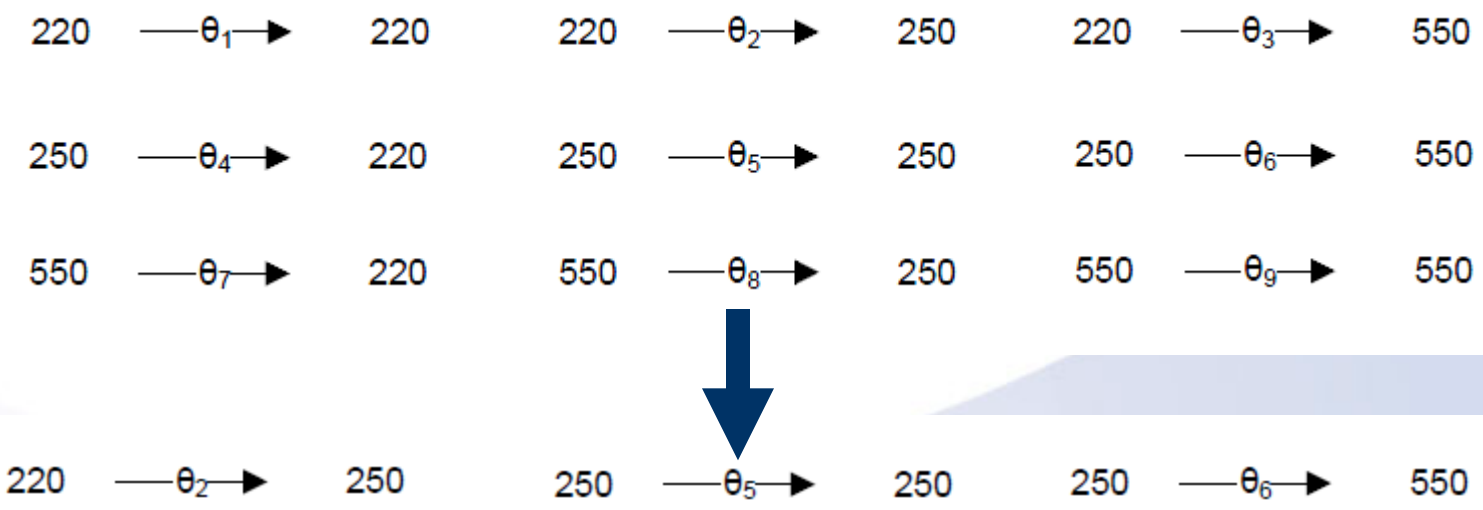
State Machine Inference

- The true state machine
  - Defined in documents
  - Implemented in applications
- The probabilistic state machine
  - Inferring from traffic traces
  - Data dependent
  - Time series mining based approach

State Machine Inference

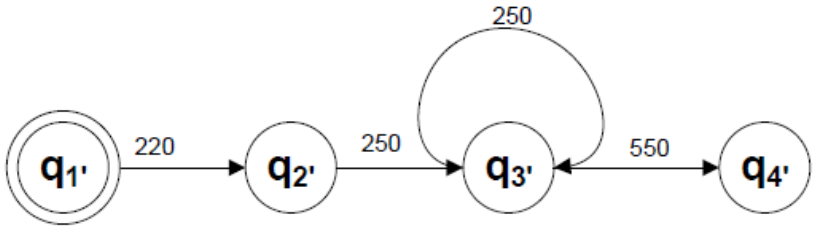
- Step 1: State labeling within a flow

- Step 2: Calculating frequency of each state pair and filtering them with a frequency filter

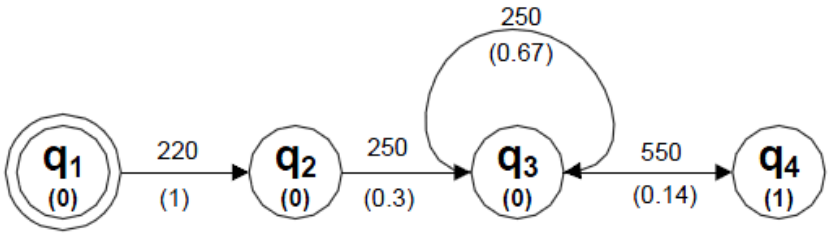


State Machine Inference

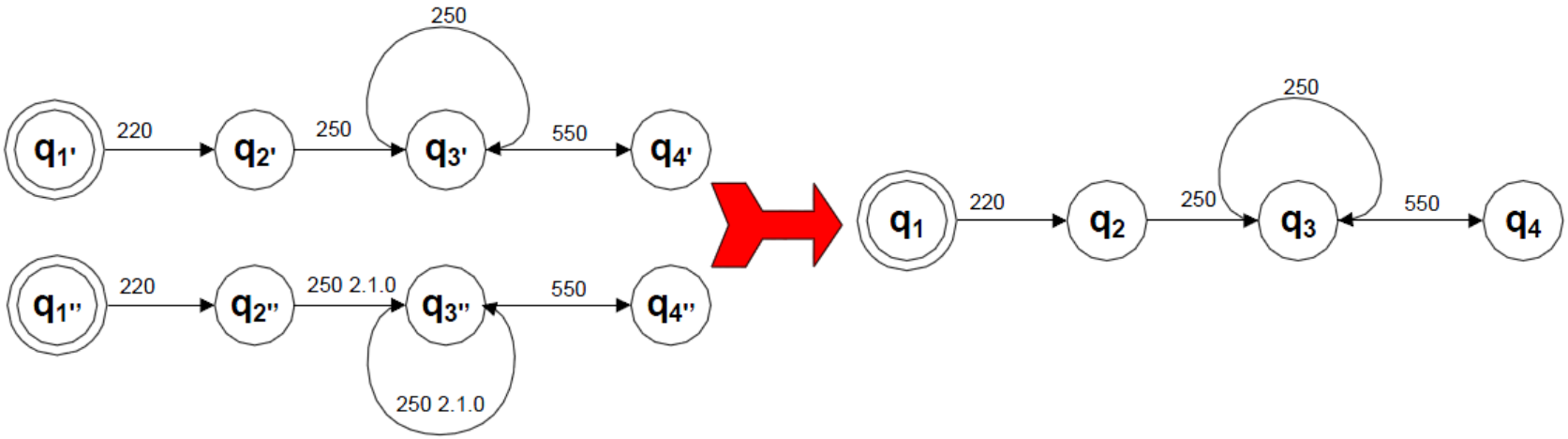
- Step 3: Depicting the linkage of each state pair with a directed labeled graph



- Step 4: Building the probabilistic protocol state machine (P-PSM)

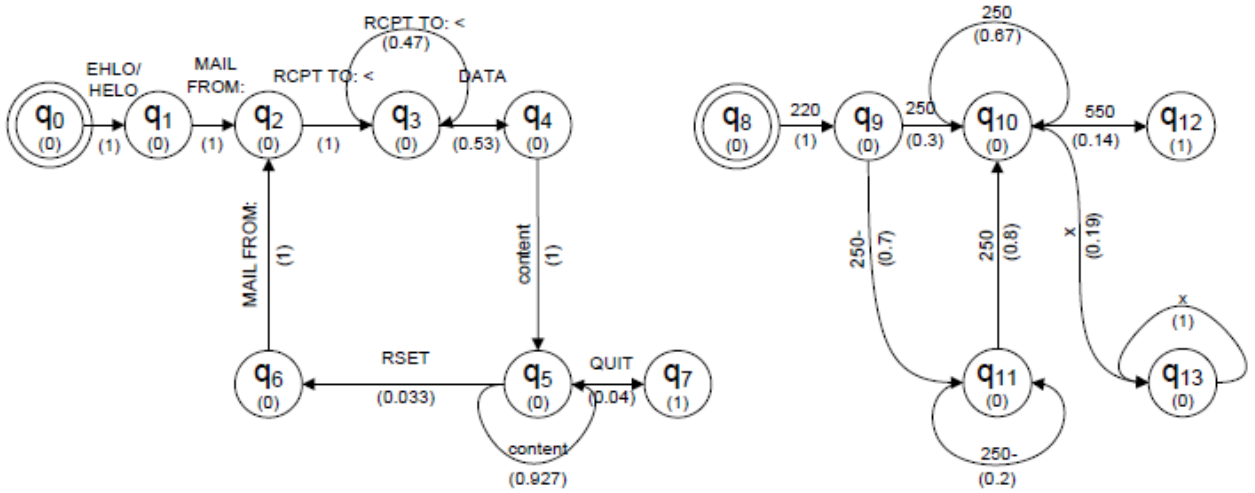


State Machine Inference



- Step 5: Merging states with same input and output

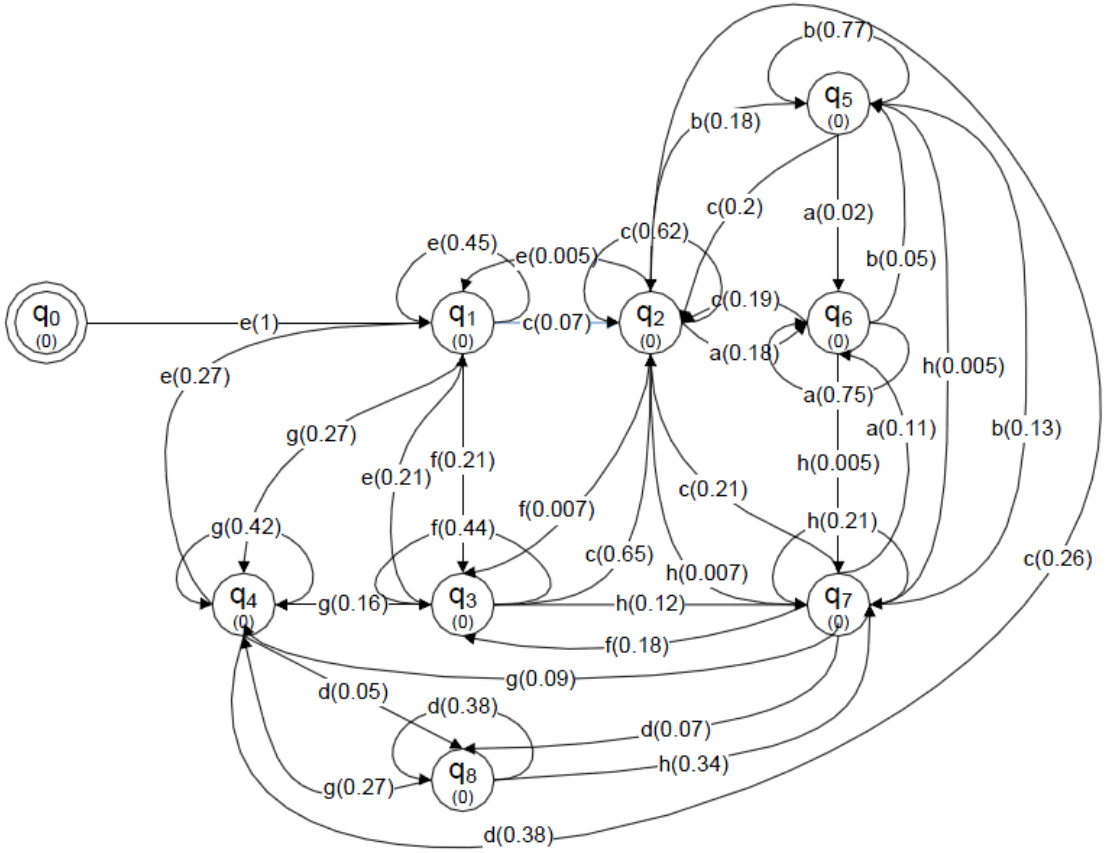
Experimental Evaluation



- Text Protocol
  - Using ASCII printable characters as protocol format
  - SMTP
    - l: 3
    - n: 12
    - k: 12

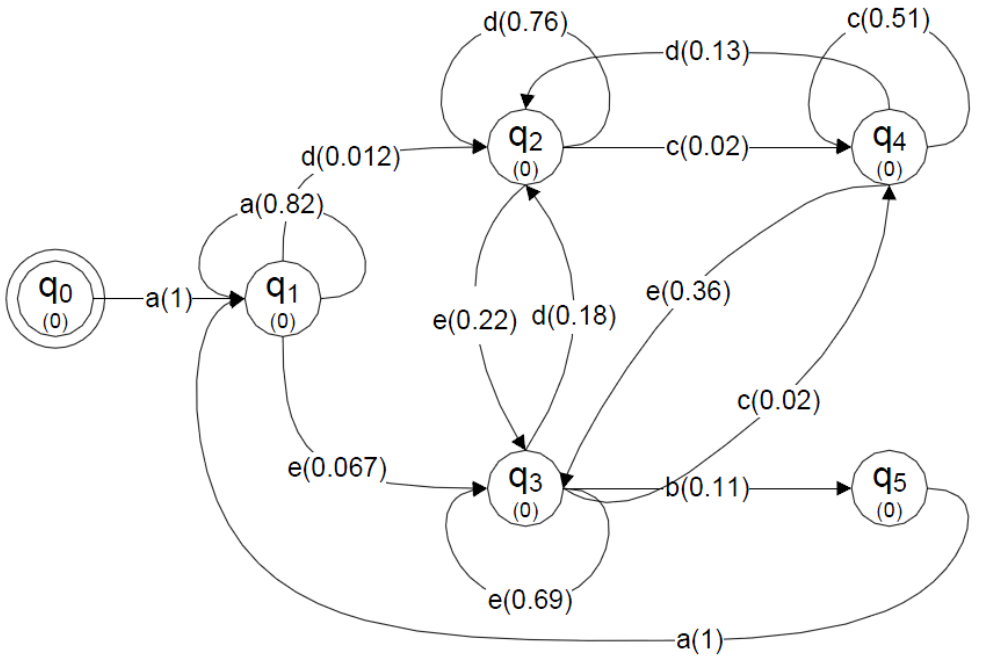
Experimental Evaluation

- Binary Protocol
  - PPLive
    - l: 3
    - n: 12
    - k: 12



Experimental Evaluation

- Binary Protocol
  - XUNLEI
    - l: 3
    - n: 12
    - k: 10->5



Experimental Evaluation

- Metric: Completeness (Coverage)
  - Using new flows to pass the inferred machine
  - SMTP flows: 100K 86%
  - PPLive UDP packets: 20K 100%
  - Xunlei UDP packets: 50K 99%

Conclusions

- *Veritas*: A system that can infer protocol state machine solely from traffic
- K-S filtering based protocol format extraction approach
- Clustering based protocol state labeling approach within a flow
- P-PSM: A probabilistic approach to infer protocol state machine

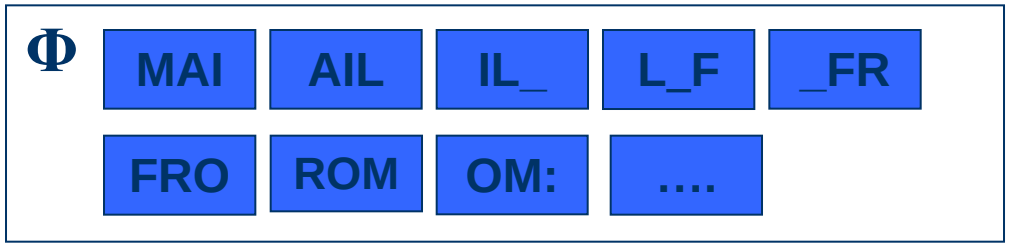
Future works

- Limitations
  - Data dependent
  - Several parameters
- Future works
  - Language model based protocol specs inference
  - Semantic inference
  - Multi-protocol inference

Gracias!

Combine Message Units

Input:



Units Combiner

Output:

FIND!!

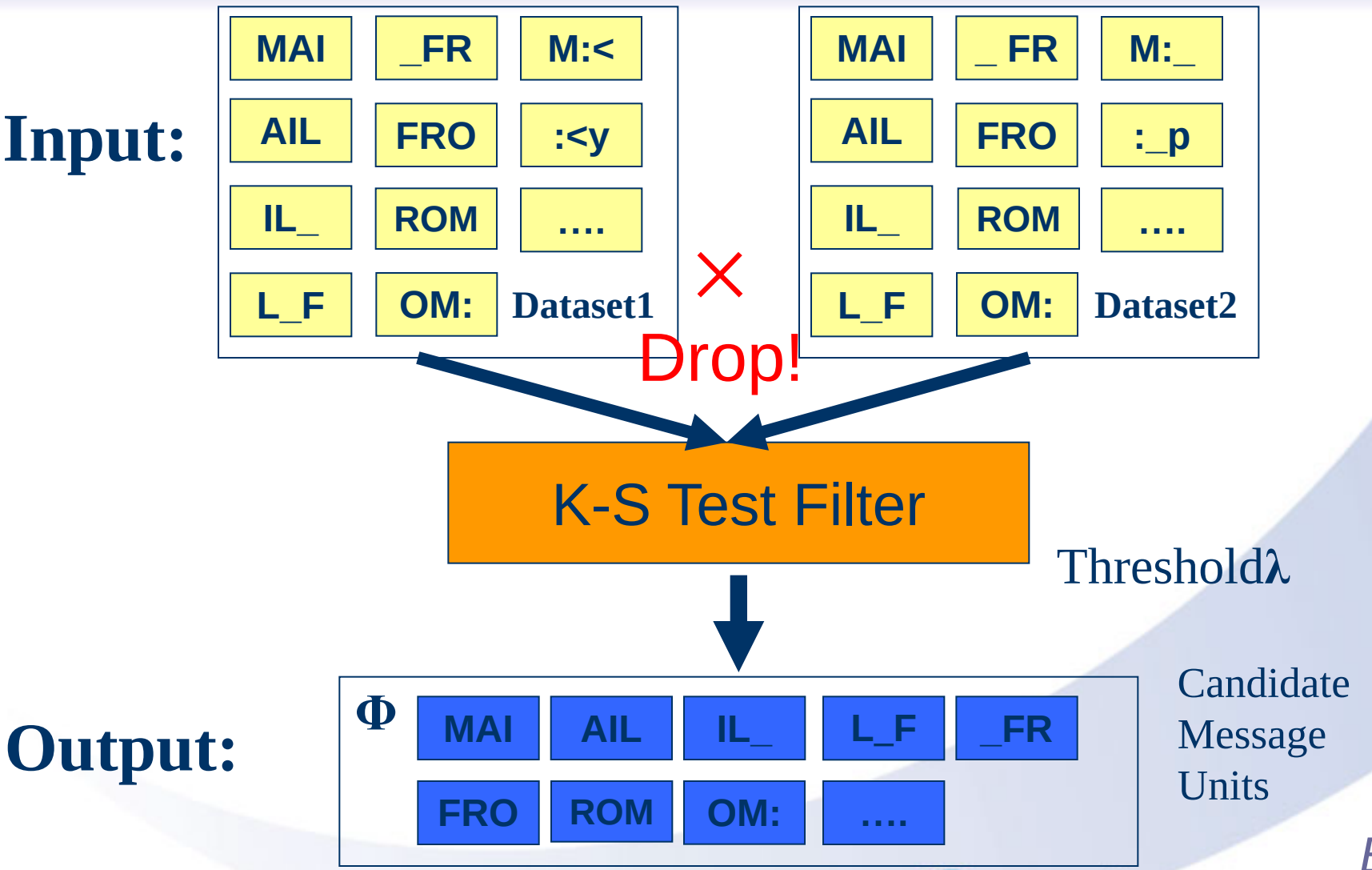
DROP!!

M	A	I	L	␣	F	R	O	M	:	<	y

M	A	I	L	␣	F	R	O	M	:

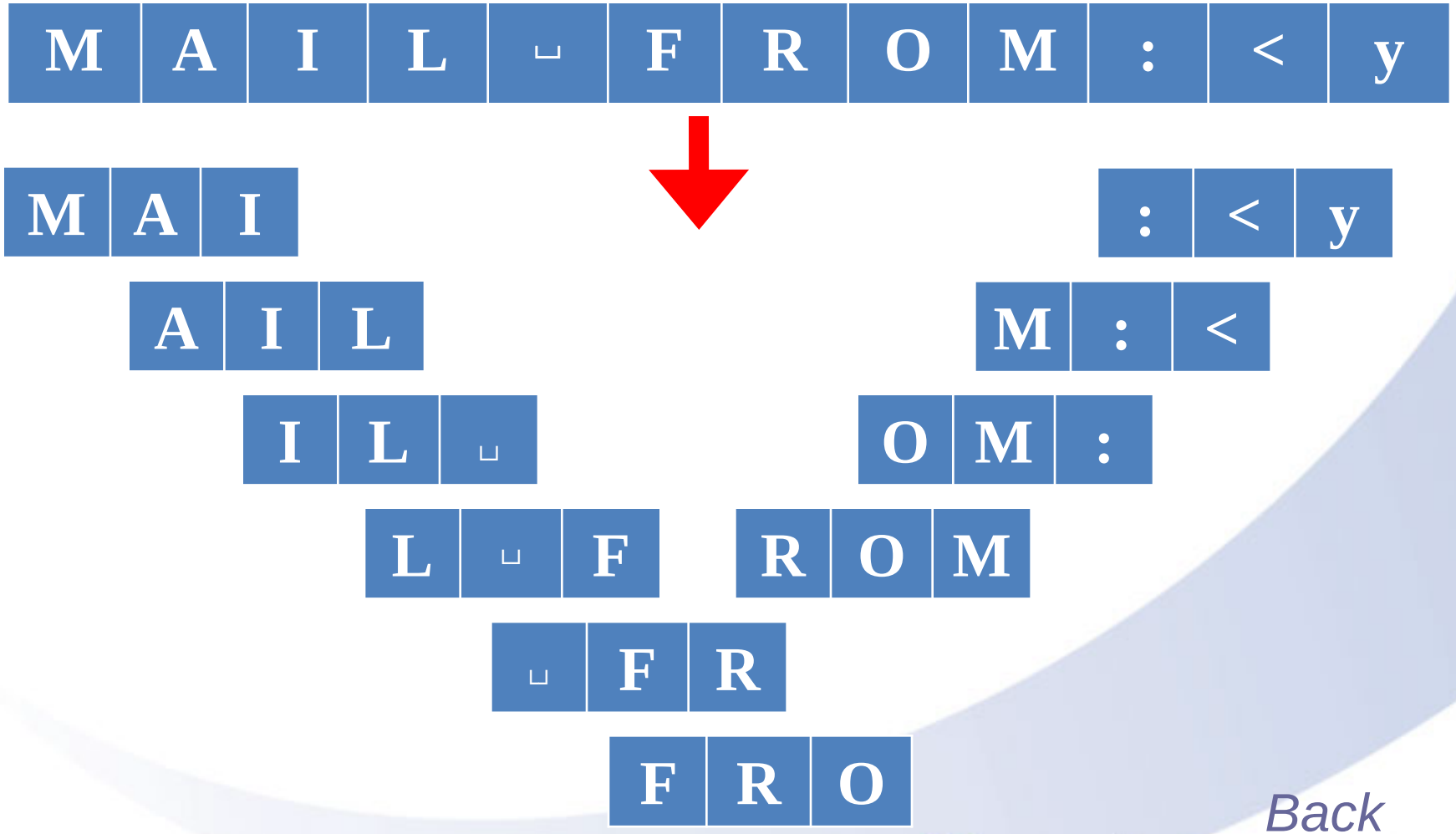
Back

Choosing candidate with K-S filter



Back

Break data into subsequences



Back