# Packet Classification and Pattern Matching Algorithms for High Performance Network Security Gateway

Jun Li, Yaxuan Qi, Bo Xu, and Zongwei Zhou

*2007.10.11*

*Network Security Lab, RIIT*
*Tsinghua University*

# Outline

- Introduction
- Packet Classification Algorithms
- Pattern Matching Algorithms
- Integrated Framework
- Network Processor Implementation
- Summary

# Outline

- **Introduction**
- Packet Classification Algorithms
- Pattern Matching Algorithms
- Integrated Framework
- Network Processor Implementation
- Summary

# New Security Gateway: UTMs

- Network security has become one of the most critical issues
- Standalone security products are not effective
- Multiple security features need to be integrated
- Holistic protection results in Unified Threat Management (UTM)

# The Value of UTMs

- Cost-effectiveness
  - Reducing the number of appliances
  - lower deployment, management and support costs
- Easy-to-use
  - Simplifying the management of complex resources and platforms
- Application-level gateway
  - Blocking network threats before they enter the internal network

# Our Research for UTMs

- Packet Classification
  - Heart of packet filtering firewall
  - Base of stateful inspection firewall
- Pattern Matching
  - Core of deep inspection firewall
  - Key in intrusion detection/prevention, and anti-virus
- Integrated Framework
  - Flow identification
  - Order preservation
  - Defragment and Reassembly

# Outline

- Introduction
- **Packet Classification Algorithms**
- Pattern Matching Algorithms
- Integrated Framework
- Network Processor Implementation
- Summary

# Packet Classification: Example

| | Field 1 | Field 2 | … | Field *F* | Action |
|---|---|---|---|---|---|
| ***Rule 1*** | 192.163.190.69/21 | 166.163.80.11/32 | … | UDP | A1 |
| ***Rule 2*** | 192.168.3.0/24 | 166.163.0.0/16 | … | TCP | A2 |
| ***…*** | … | … | … | … | … |
| ***Rule N*** | 0.0.0.0/0 | 0.0.0.0/0 | … | ANY | An |

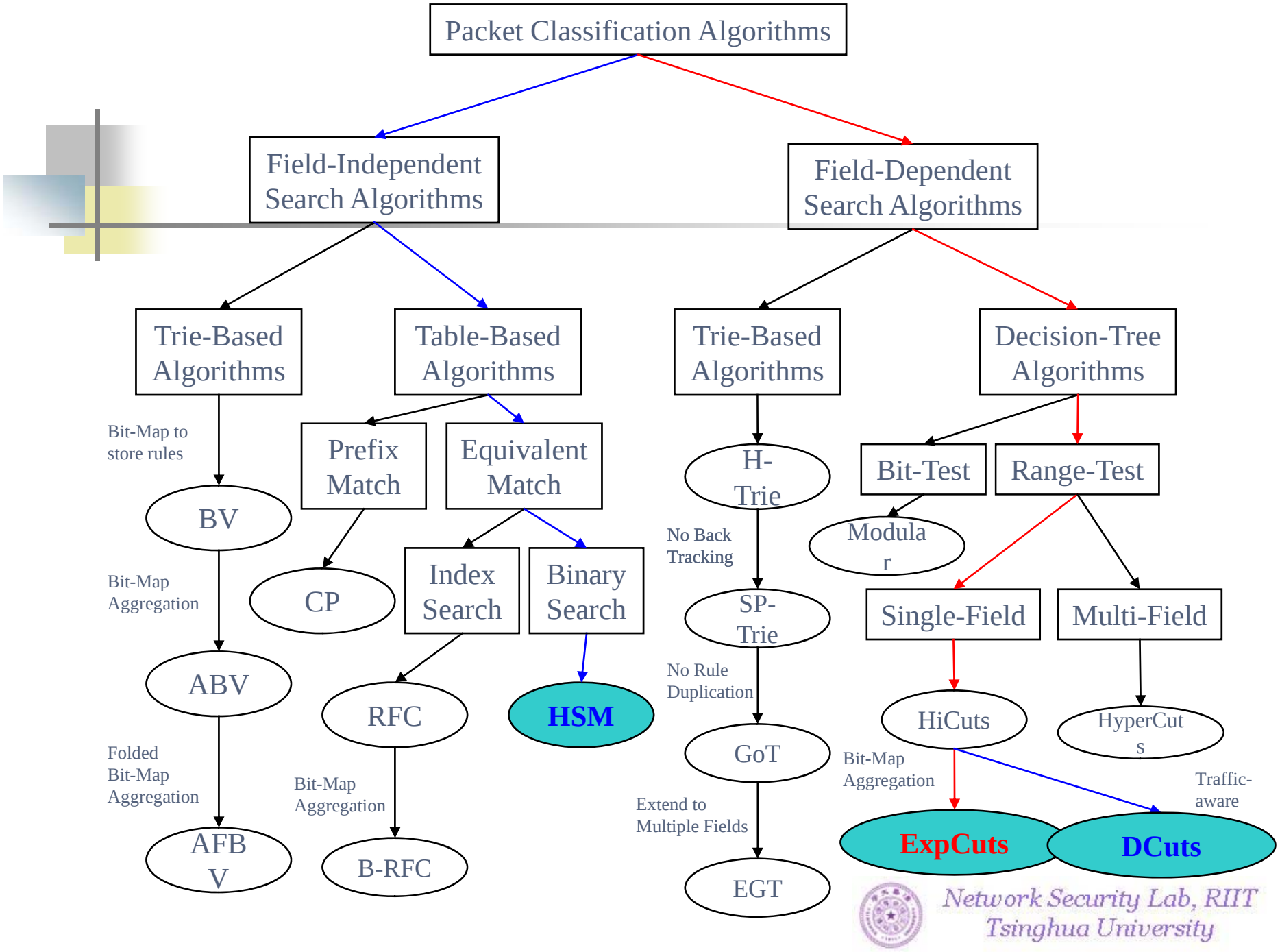
- Definition
  - Given N rules, find the action associated with the highest priority rule matching an incoming packet
- Example
  - A packet P(192.168.3.32, 166.163.171.71, ..., TCP) would have action A2 applied to it

# Packet Classification: Complexity

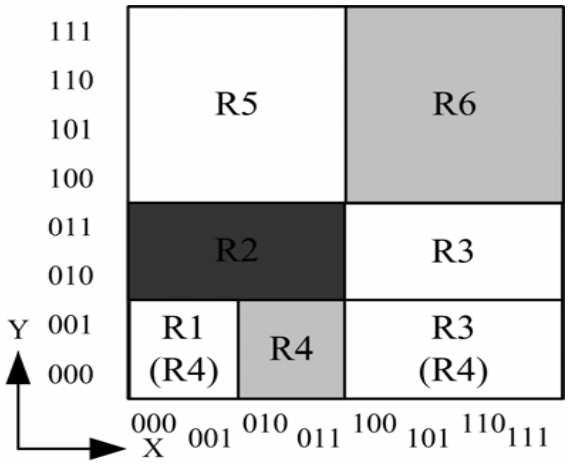
- Computational Geometry
  - Point Location among $N$ non-overlapping regions in $F$ dimensions
  - Takes either $O(\log N)$ time with $O(N^F)$ space or $O(N)$ space with $O(\log^{F-1} N)$ time
  - E.g. $N$=1000，$F$=4：1000G space, or 1000 accesses
- De-overlapping
  - $N$ overlapping regions need up to $(2N-1)^F$ non-overlapping region to represent
- Range-to-Prefix
  - $N$ rules in range $[0, 2^W-1]$ need up to $N(2W-1)$ prefixes

# Packet Classification: Observations

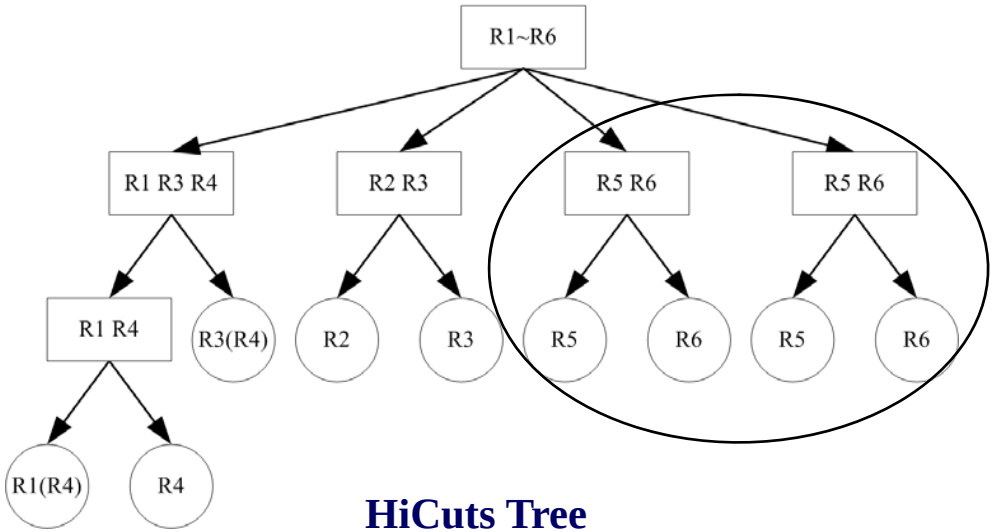
- It is not possible to arrive at a practical worst case solution
  - No application reaches the worst case bound
  - Real-life rule sets have some inherent data-structures
- No single algorithm performs well in all cases
  - Different applications require different packet classification schemes
  - Hybrid algorithms might be able to combine the advantages of several different approaches


Packet Classification Algorithms
Field-Independent Search Algorithms
Field-Dependent Search Algorithms
Trie-Based Algorithms
Table-Based Algorithms
Trie-Based Algorithms
Decision-Tree Algorithms
Bit-Map to store rules
BV
Bit-Map Aggregation
ABV
Folded Bit-Map Aggregation
AFBV
Prefix Match
CP
Equivalent Match
Index Search
RFC
Bit-Map Aggregation
B-RFC
Binary Search
HSM
H-Trie
No Back Tracking
SP-Trie
No Rule Duplication
GoT
Extend to Multiple Fields
EGT
Bit-Test
Modular
Range-Test
Single-Field
Multi-Field
HiCuts
HyperCuts
Bit-Map Aggregation
Traffic-aware
ExpCuts
DCuts
Network Security Lab, RIIT
Tsinghua University

# Related Work: HiCuts




**Rules**

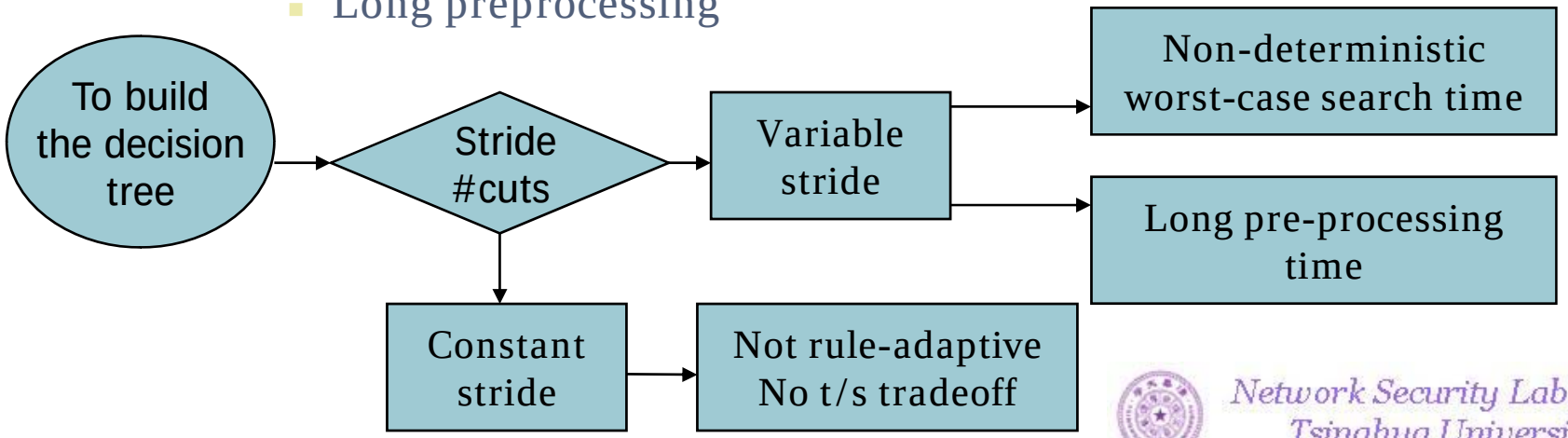



**HiCuts Tree**

- Field-dependent cuttings
- Variable number of cuttings
- Linear Search required

# Related Work: Summary

- Decision Tree Algorithms
  - Pros
    - Modest memory usage
    - Good average search speed
    - Ruleset adaptive
  - Cons
    - Non-deterministic worst-case search time
    - Excessive memory usage for large rulesets
    - Long preprocessing

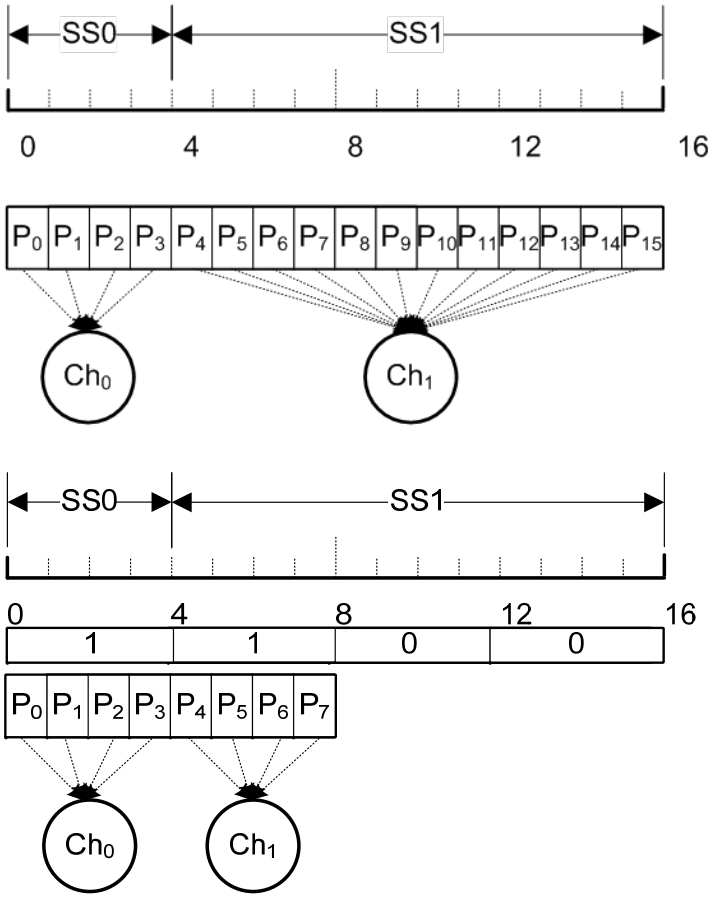


# ExpCuts: Novel Ideas

- Guarantee the Worst-case Search Time
  - Constant stride: Fixed number of cuttings
  - E.g. *for 5-tuple packet classification, let stride=8, then tree-depth*$_{worst}$*=(32+32+16+16+8)/8=13*
- Reduce the Pointer Array Size
  - Using bit-string to aggregate contiguous sub-spaces
- Further Space Compression
  - Aggregate non-contiguous sub-spaces
  - *Each point in the array points to a unique sub-space*

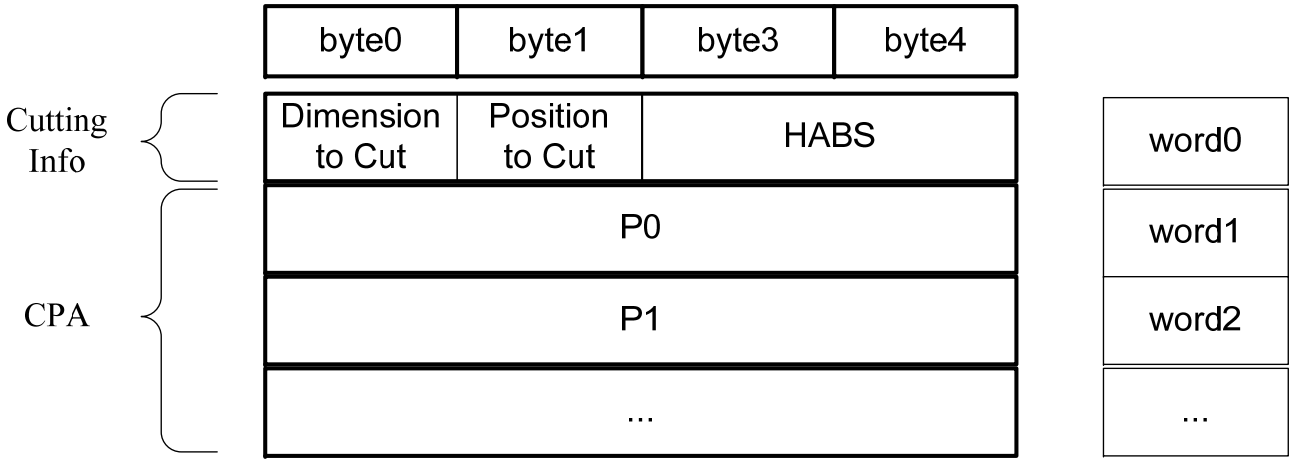
# ExpCuts : Optimization

- Compressed Pointer Array (CPA)
  - Observation: pointer arrays are sparse
  - Compress a sequence of consecutively identical pointers as one element in CPA
- Aggregation Bit String (ABS)
  - Use ABS to track the appearance of unique elements in the pointer array
  - Use population count instruction
- Hierarchical ABS (HABS)
  - Observation: ABS is still too large
  - Trade memory for speed using one bit for multiple pointers

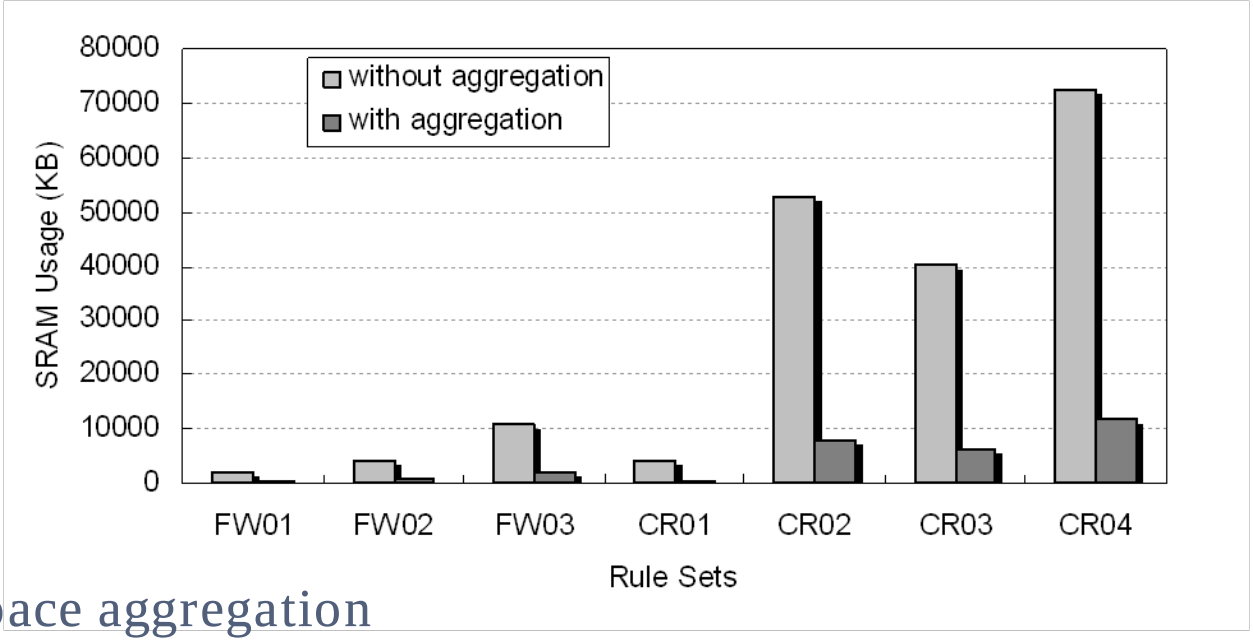


# Parameters and Data-structure

- Stride *w*: stride is set to 8
  - 256 cuttings per level, totally 13 levels
- Size of HABS: the size of HABS is set to be 16
  - HABS can be is stored together with the cutting information within a single 32-bit long-word
  - Each bit represent 256/16=16 consecutive pointers

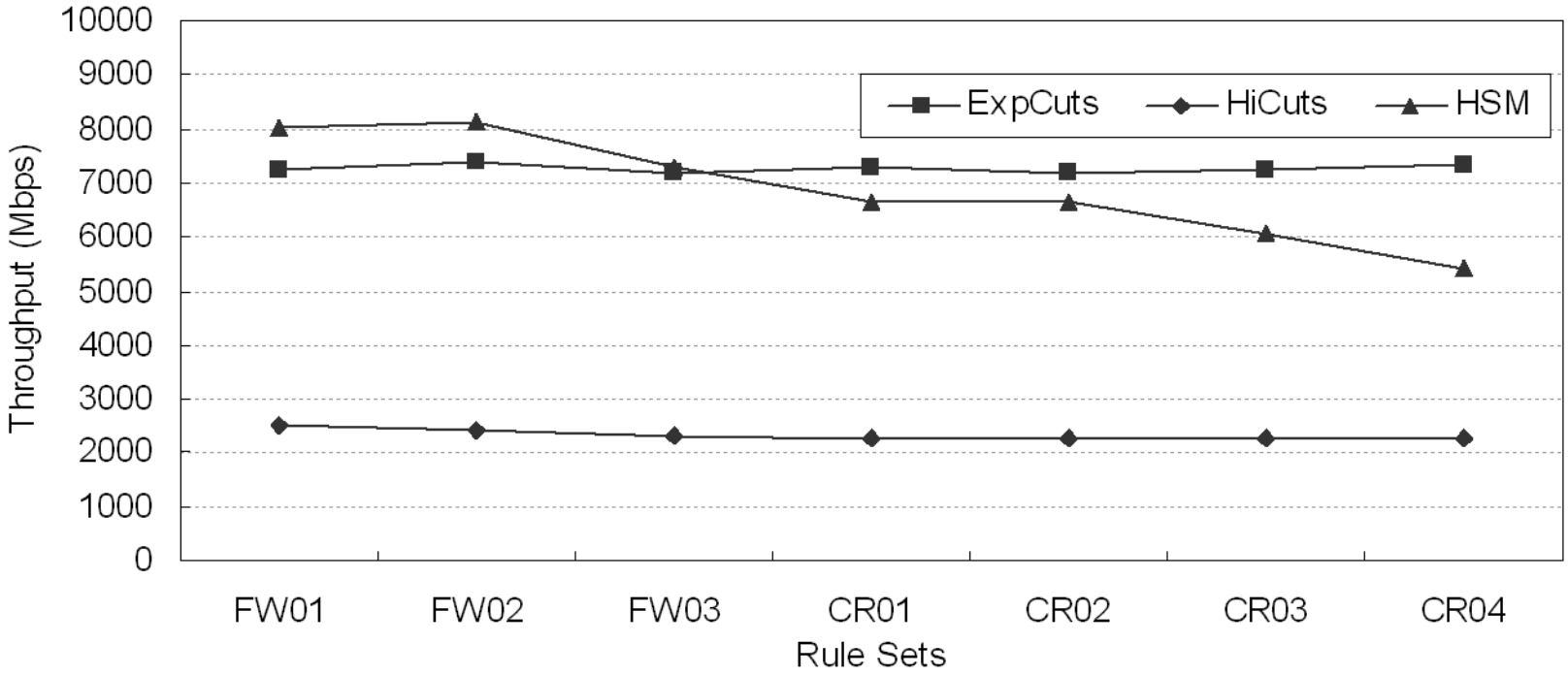


# ExpCuts Space Aggregation




- Space aggregation
  - Reduce up to 85% memory usage
- Without space aggregation
  - CR02~04 cannot be implement in the 8MB*3 SRAM chips
- With space aggregation
  - All rule sets can be implemented

# ExpCuts Throughput




- Vs. HiCuts (no linear search): 3 times faster than HiCuts
- Vs. HSM: Stable worst-case performance

# Outline

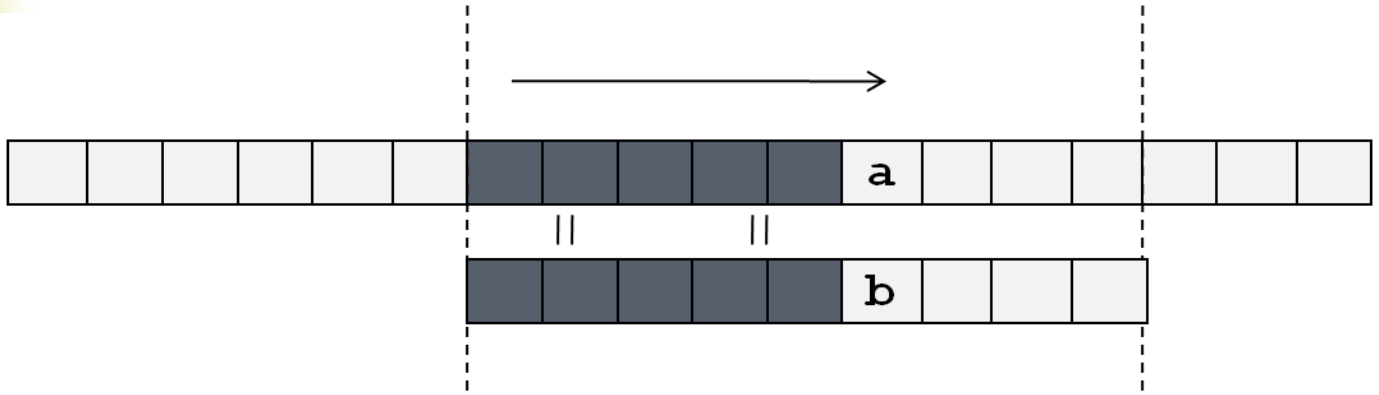
- Introduction
- Packet Classification Algorithms
- **Pattern Matching Algorithms**
- Integrated Framework
- Network Processor Implementation
- Summary

# Definition

- Given an alphabet set S, a pattern P of length m and a text T of length n, find if P is in T or the position (s) P matches a substring of T, where usually m<<n

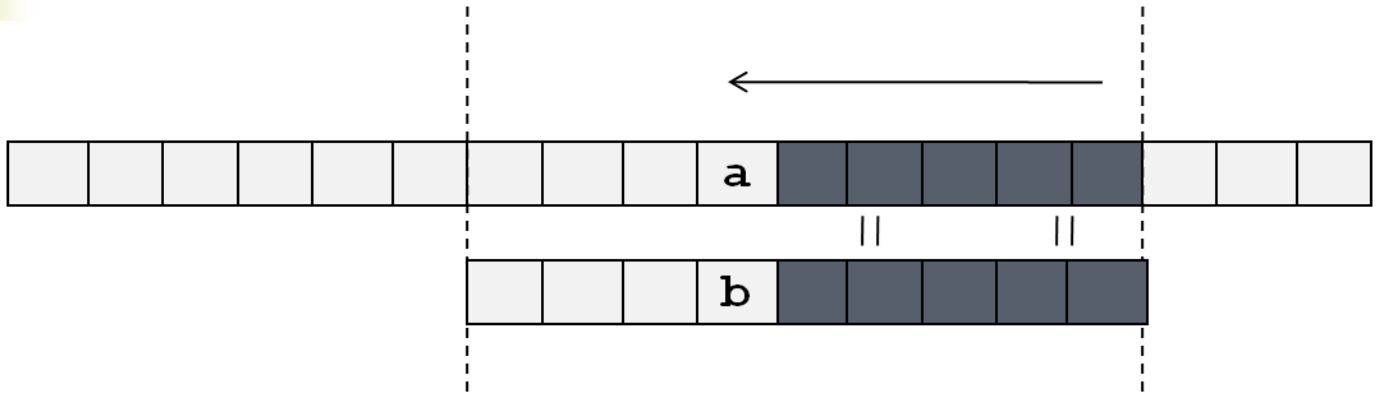
- Considering the pattern P
  - String
    - **exact string matching**
  - String with errors
    - **approximate string matching**
  - Regular expression
    - **regular expression matching**

# Prefix Based Algorithms



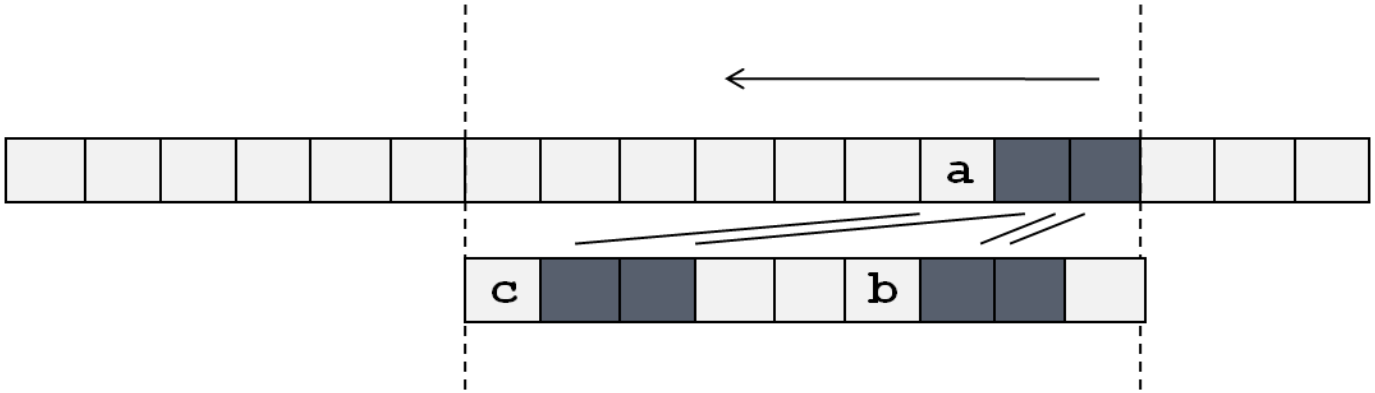

- Matching forward in the search window
- All the characters are read

# Suffix Based Algorithms



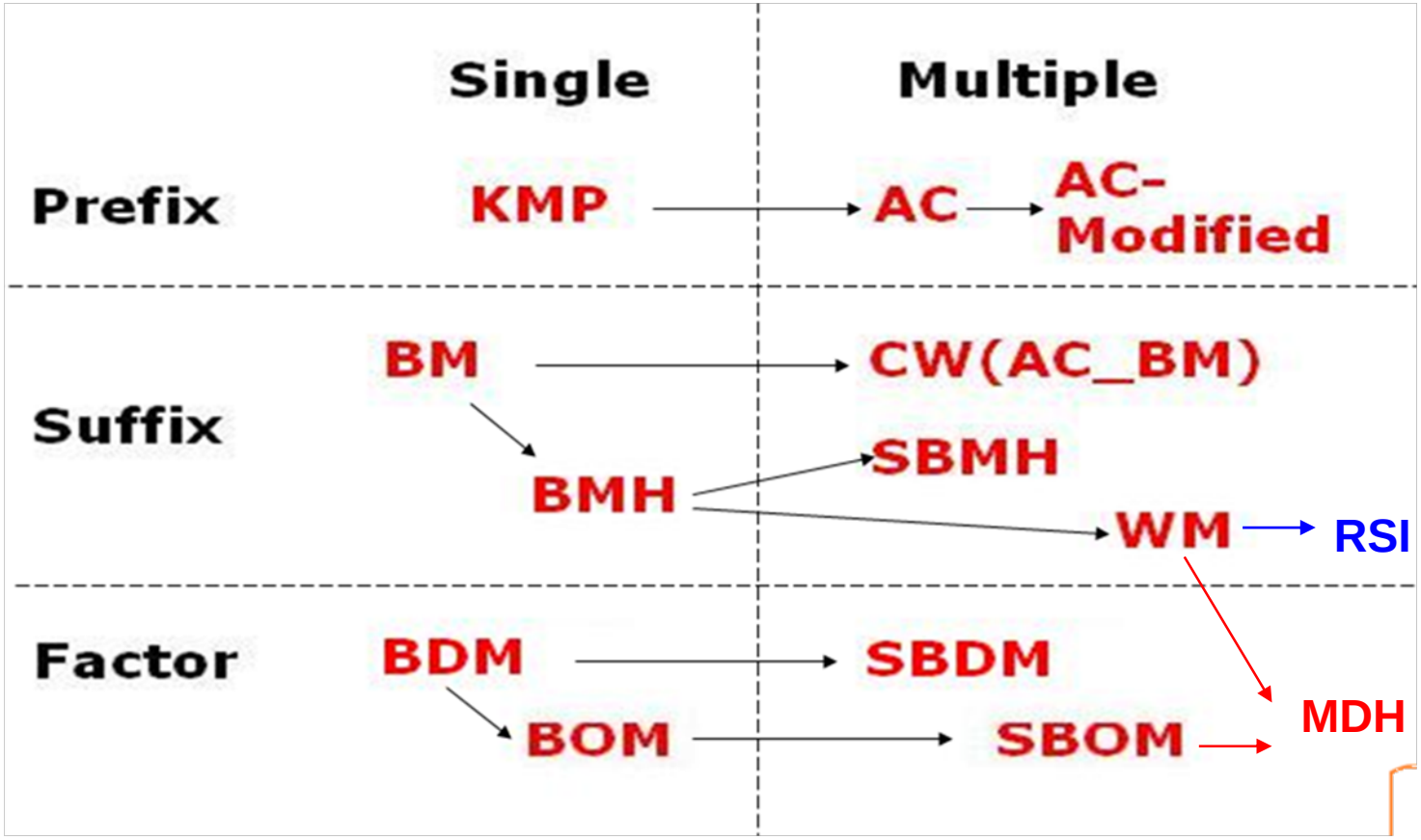

- Matching backward along the search window
- Not all the characters are read due to “shift” (“skip”, “leap”), which leads to sublinear average-case algorithms

# Factor Based Algorithms



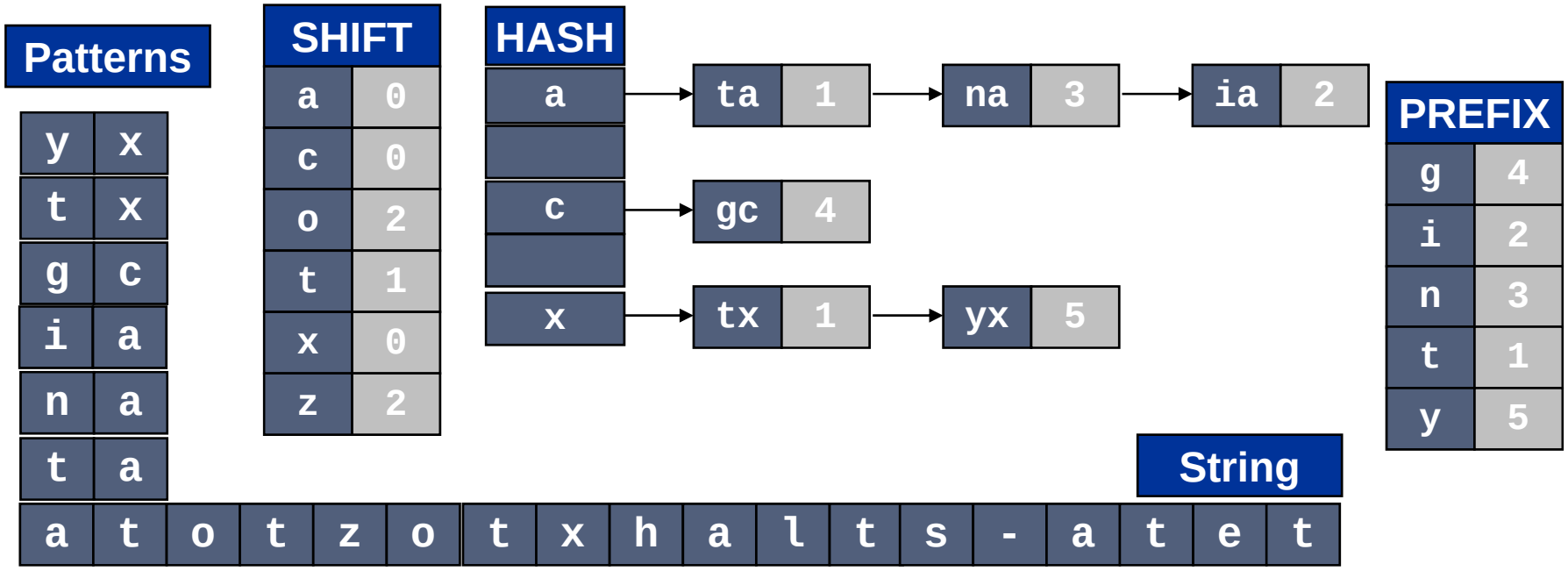

- Matching backwards along the search window
- Not all the characters are read, but requires to recognize the set of factors (sub pattern) of the pattern (s)

# Categorization



# Wu-Manber Algorithm

- A hash table **SHIFT** to store the shift values of character blocks and link the patterns has the same last character block
- A hash table **PREFIX** to discriminate patterns link with the **HASH** entry
- SHIFT and HASH share the same hash function

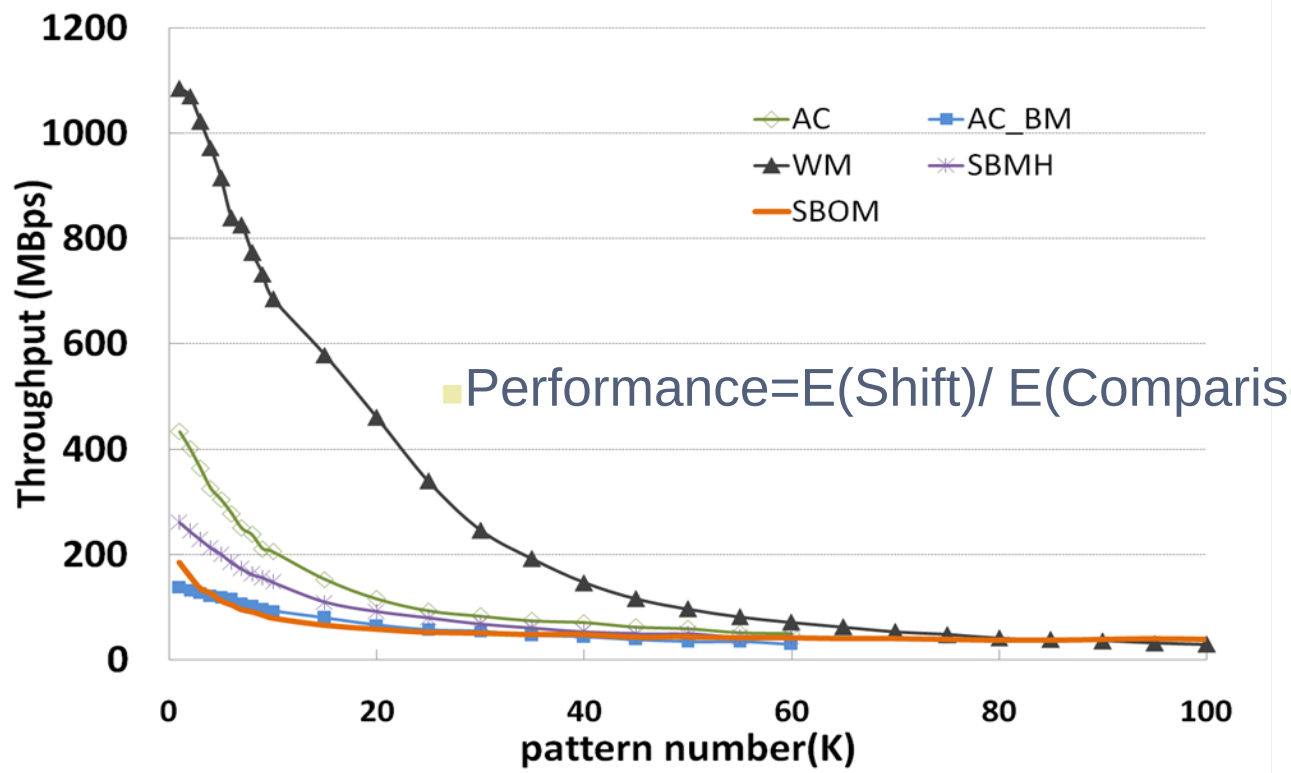


# Wu-Manber Algorithm

- Pros: Excellent average time performance
  - Hash function
  - Avoid unnecessary character comparison
- Cons:
  - Bad worse case performance
    - Ex: {baa, caa, daa} against a string of “a”
  - Shift distance is limited by length of shortest pattern

# Pattern Matching Challenges

- Large-scale pattern sets: e.g. Clam AntiVirus
- Increasing network edge bandwidth: 10Gbps UTM



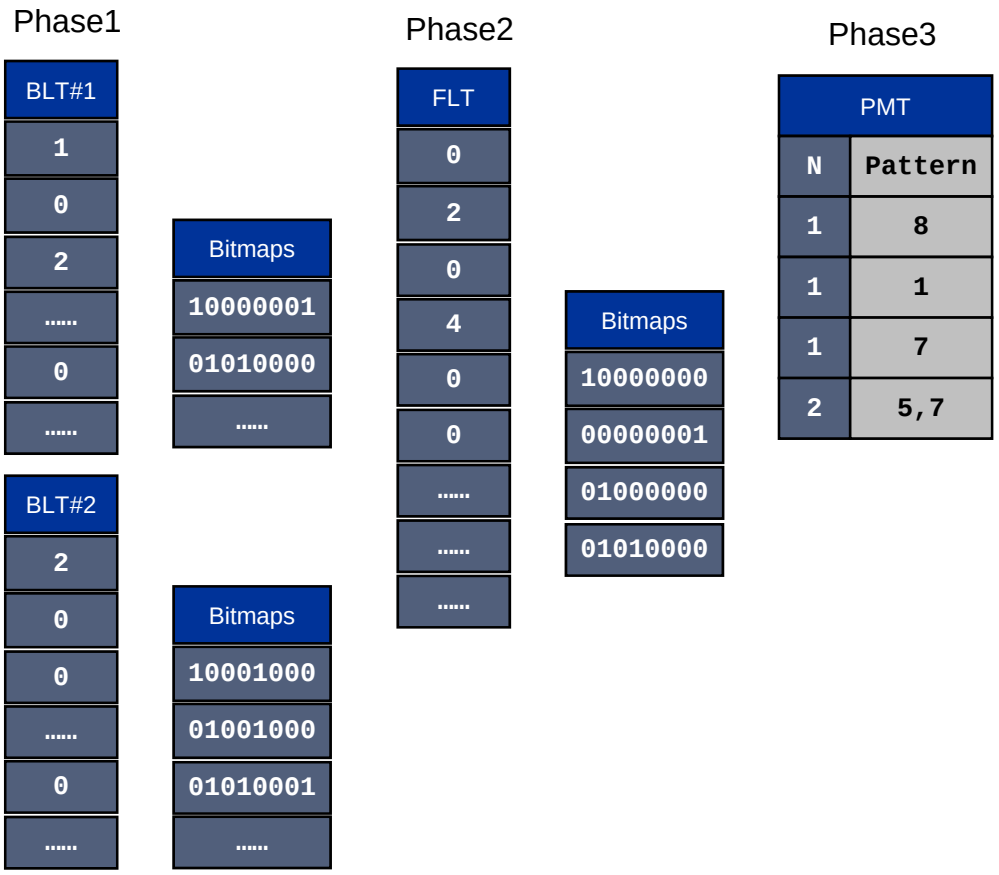
# Motivation and Observations

- Shifts are needed when designing high speed multi-pattern string matching algorithms for large scale pattern sets
- Table based algorithms are faster as direct table lookup is faster than automaton and trie traversing
- WM can be improved for large pattern sets as two/three character heuristic is not strong enough to generate shifts and diminish hash collisions

# RSI: Recursive Shit Indexing

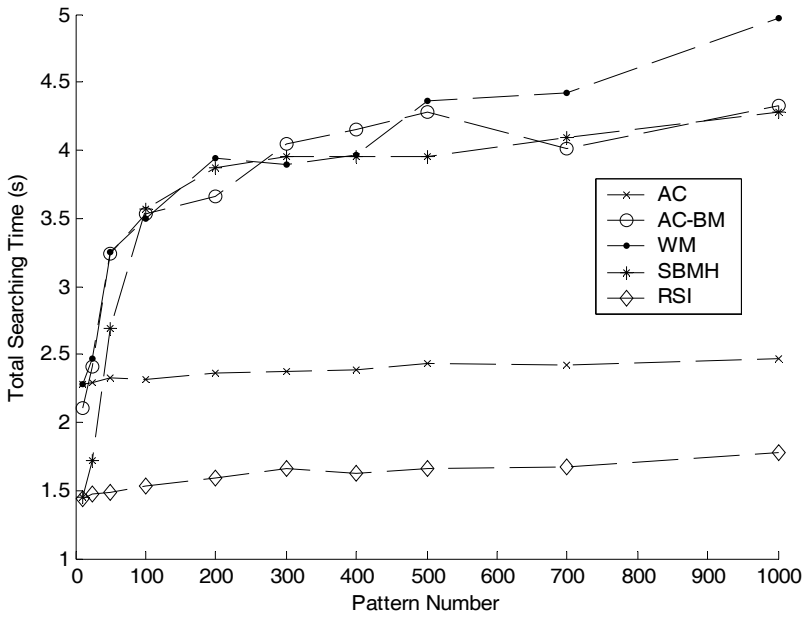
- More heuristic to generate long leaps when there is no match
  - Block Leap Tables (BLT)
  - Further Leap Table (FLT)
- Keep track of the potential matching patterns to avoid naïve comparisons with all the patterns
  - Potential Match Table (PMT)
- Consider both time and space efficiency

# RSI: Complete Table Structure

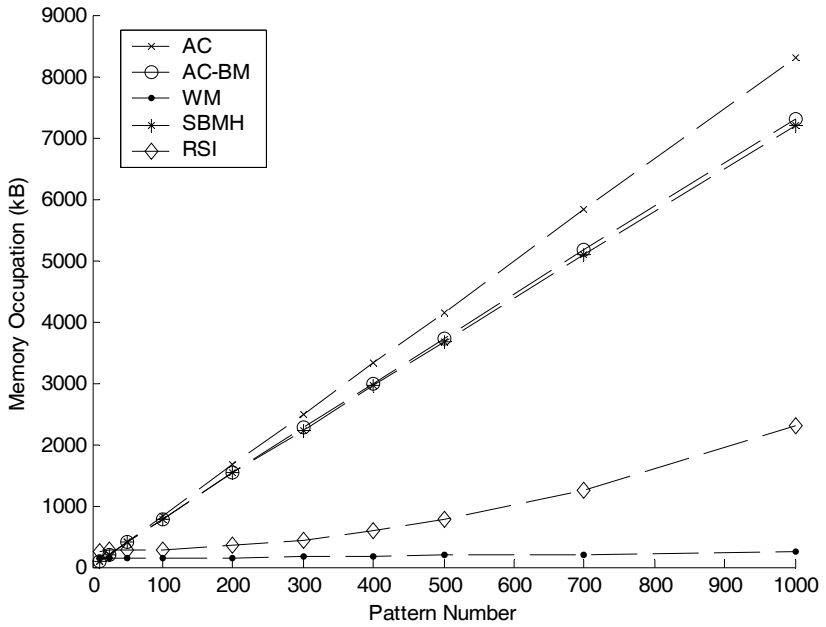



- Create BLT#1 according to the first block of all the patterns
- Create BLT#2 according to the second block of all the patterns
- Corresponding to zero values of BLT#1 and BLT#2, create FLT according to the combined block of 4 characters
- Corresponding to zero values of FLT, create PMT to record the potential match patterns

# RSI: Performance




- Time with fixed pattern length 8 and varying pattern number from 10~1000




- Space with fixed pattern length 8 and varying pattern number from 10~1000

# MDH: Multi-phase Dynamic Hash

High Speed Multi-phase Dynamic Hash String Matching Algorithm for Large-scale Pattern set

- Two important improvement on WM
  - Multi-phase Hash
  - Dynamic-cut Heuristics
- High throughput and low memory requirement under large-scale pattern set

# MDH: Multi-phase Hash

- Use big block size (E.g. 4)
- SHIFT table
  - Compressed hash function h1
  - Reduce table size from 2^32 to 2^a (a<32, e.g. a=20)
- PMT table
  - Compressed hash function h2
  - Handle with all the character blocks has zero shift value
  - Table size is 2^b (b<a<32, e.g. b=17)

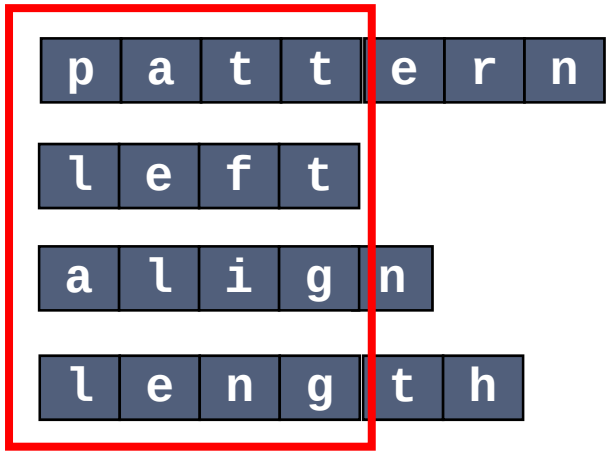
# MDH: Multi-phase Hash

- Reduce zero value entry in SHIFT table
- Cut down memory requirement
  - WM: 2^32+2^32  MDH: 2^a+2^b

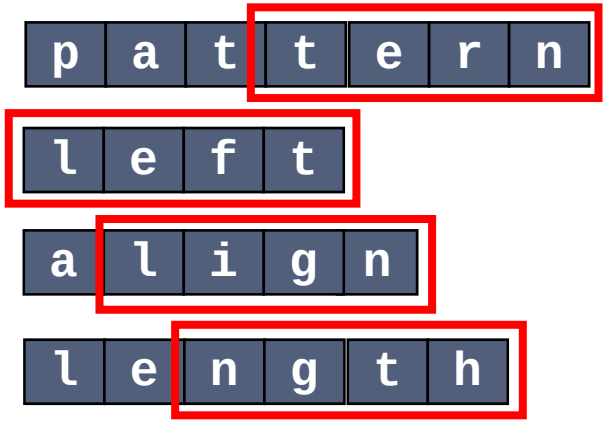
| Pattern set size | WM(*B*=2) | | WM(*B*=3) | | MDH | |
|---|---|---|---|---|---|---|
| | ZR (%) | MEM (MB) | ZR (%) | MEM (MB) | ZR (%) | MEM (MB) |
| 10k | 14.2 | 0.95 | 0.059 | 80.64 | 0.85 | 2.42 |
| 25k | 31.7 | 1.91 | 0.149 | 81.59 | 1.91 | 2.98 |
| 50k | 53.3 | 3.5 | 0.297 | 83.19 | 3.46 | 3.93 |
| 75k | 68.0 | 5.09 | 0.446 | 84.78 | 4.32 | 4.87 |
| 100k | 78.3 | 6.69 | 0.594 | 86.38 | 6.25 | 5.81 |

# MDH: Dynamic-cut Heuristics

WM




MDH




optimum *m* window

# MDH: Performance

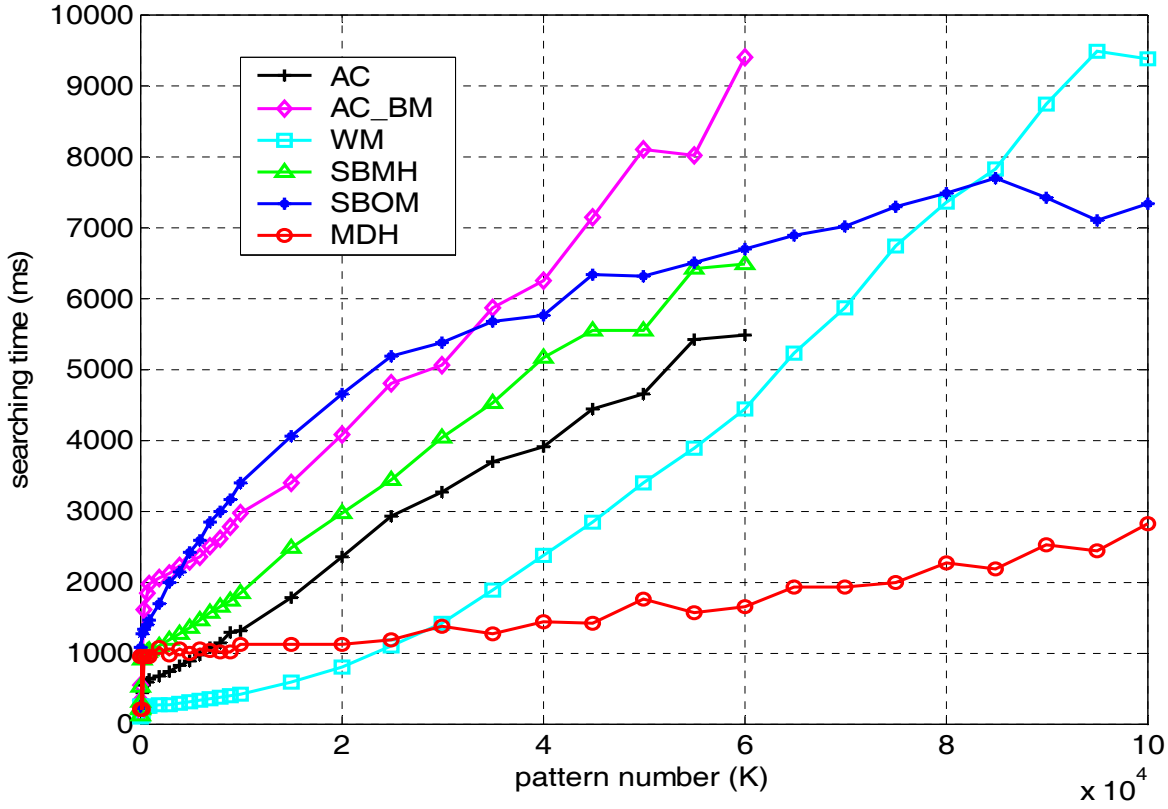
- Lower ZR and APM
- Reasonable preprocessing overhead
- Improved searching throughput

- Real-life pattern set from ClamAV

| Algorithm | 20k | | 40k | | 60k | | 77k | |
|---|---|---|---|---|---|---|---|---|
| | Thr (Mbps) | Mem (MB) | Thr (Mbps) | Mem (MB) | Thr (Mbps) | Mem (MB) | Thr (Mbps) | Mem (MB) |
| MDH | 250.56 | 3.82 | 203.28 | 5.2 | 174.24 | 8.08 | 150.16 | 10.41 |
| WM | 329.52 | 3.33 | 126 | 5.2 | 66.88 | 8.53 | 43.36 | 11.27 |
| SBOM | 69.68 | 81.87 | 56.16 | 162.5 | 43.76 | 244.7 | 36.48 | 316.84 |

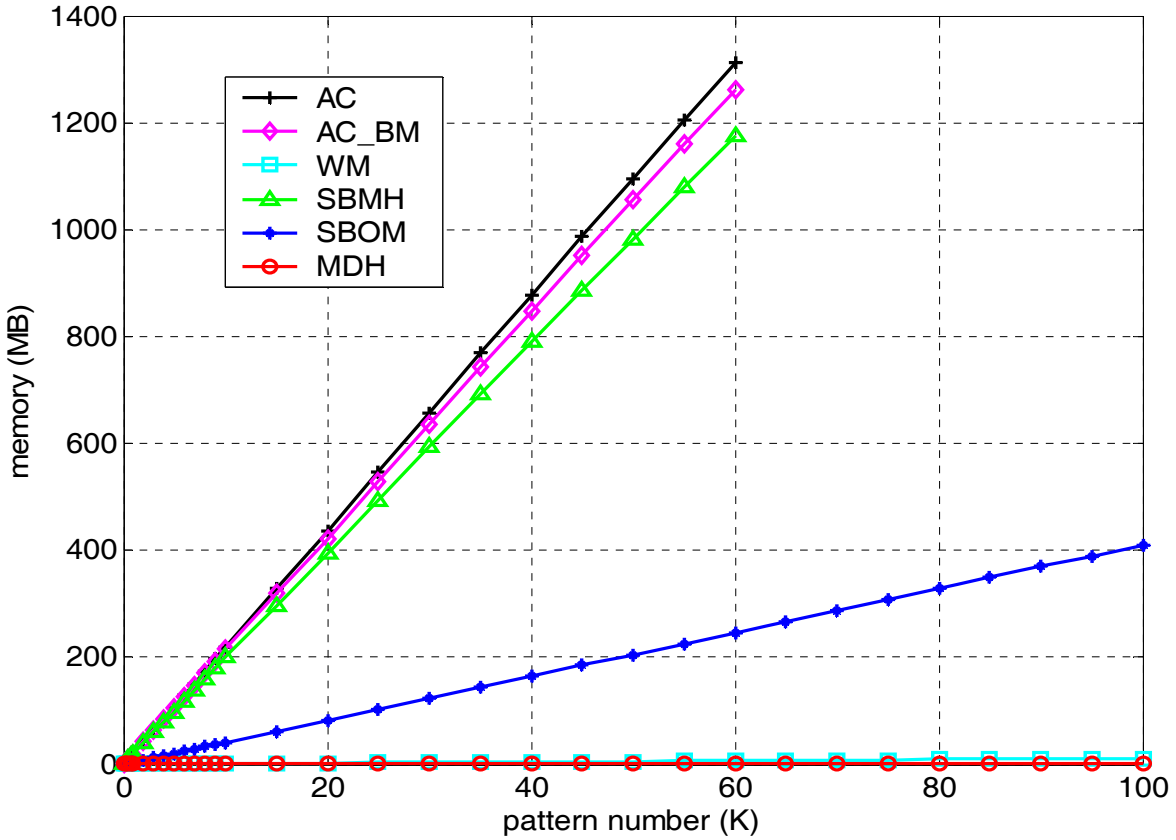
# MDH: Performance

- Searching time



# MDH: Performance

- Memory requirement



# Outline

- Introduction
- Packet Classification Algorithms
- Pattern Matching Algorithms
- **Integrated Framework**
- Network Processor Implementation
- Summary

# Problems

- Redundant protocol processing
  - Security applications are commonly deployed in different modules
  - Each packet header is loaded multiple times from the main memory and then processed by different modules
- Unnecessary deep inspection
  - Deep inspection is very time-consuming and often the bottleneck of a UTM device
  - Only malicious or dubious traffic needs to be processed using deep inspection

# Contribution

- **Algorithm:** Integrated protocol processing (IPP)
  - Unnecessary deep inspection can be significantly reduced by protocol analysis
  - Protocol processing can be effectively integrated in a single module
- **Implementation:** Network processor
  - NPs are optimized for network processing
  - IPP algorithm is implemented and evaluated on the Intel IXP2850 NP

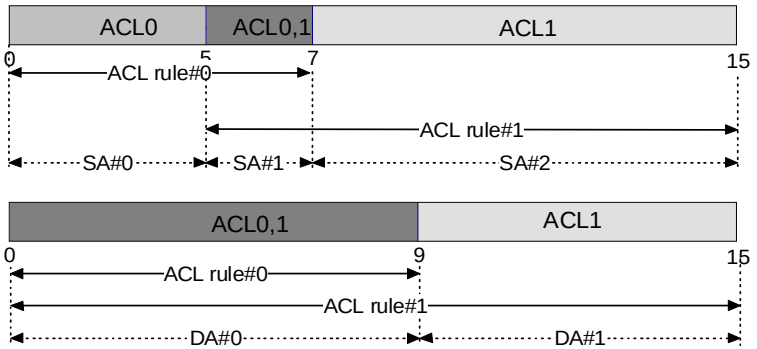
# Protocol Processing

- Generally, protocol processing
  - Refer to all network security applications responsible for the manipulations of networking protocols
  - Involve packet classification, session setup/teardown, and statistics gathering...
- In our research, protocol processing
  - Focus on multidimensional packet classification operation
  - Because it is the key operation to the system-level optimization

# HSM

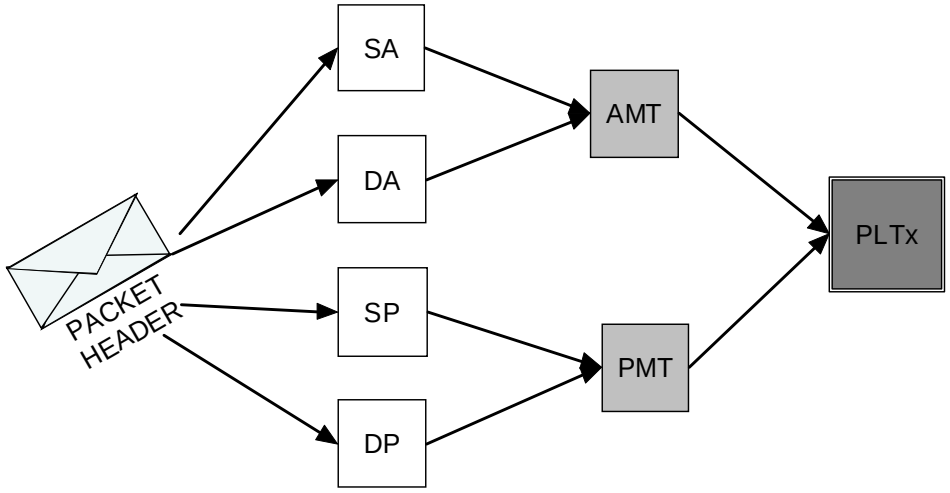
- HSM performs multi-phase searches
  - In the first phase, the original search space are **segmented**
  - In subsequent phases, spaces are recursively **aggregated**
  - In the final phase, the table lookup yields the action

Segmentation



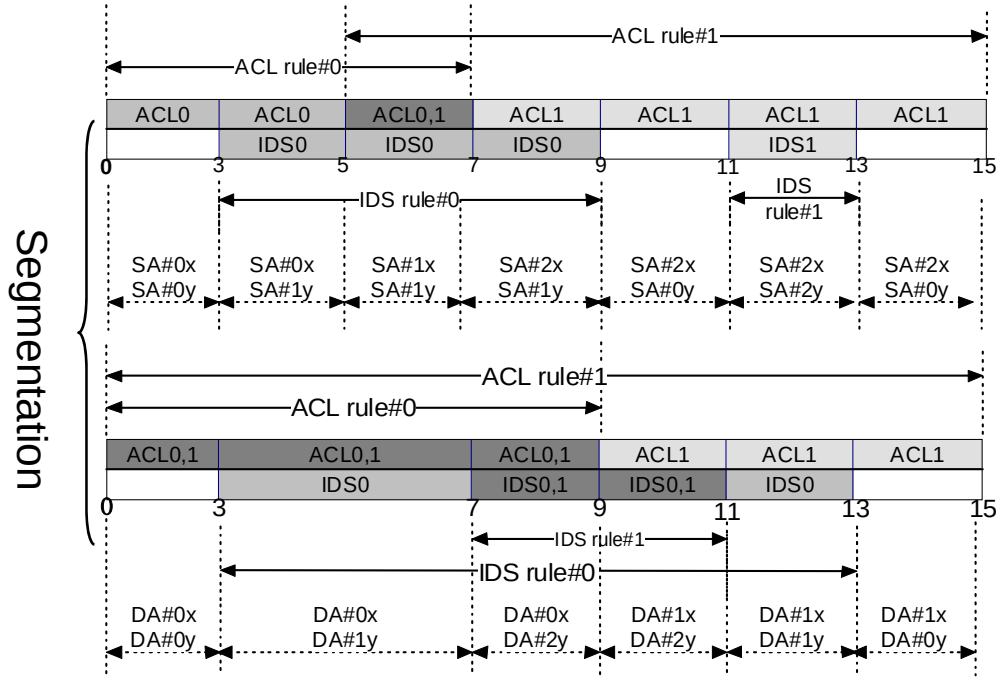

Aggregation

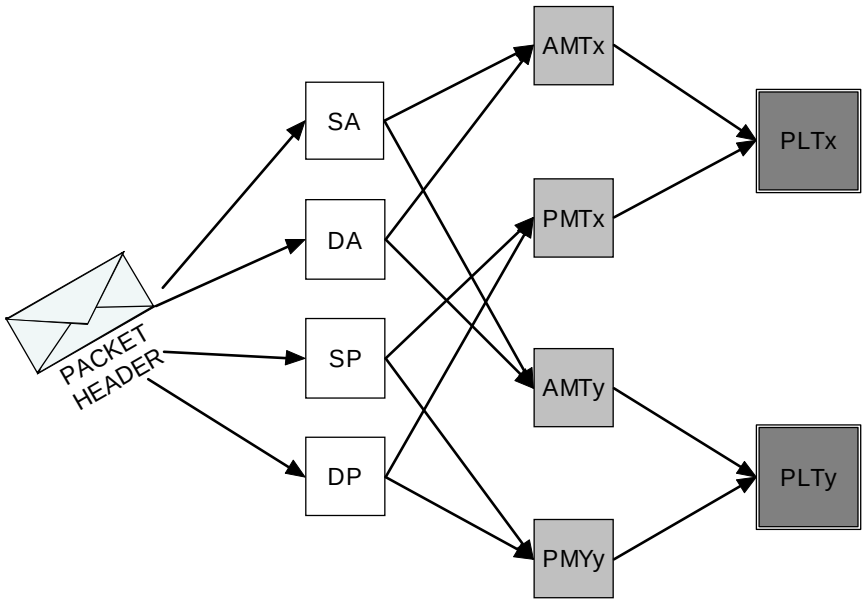
| AMT | SA#0 | SA#1 | SA#2 |
|---|---|---|---|
| DA#0 | 1 (ACL0) | 2(ACL1) | 3(ACL1) |
| DA#1 | 0(N/A) | 3(ACL1) | 3(ACL1) |



# IPP

- IPP handles two independent rule sets by
  - Integrated space segmentation
  - Independent space aggregation



# Temporal Performance

Memory Access (Unit: 32bit-word)

| RULESET | #RULE | FW+IDS | IPP |
|---|---|---|---|
| **ACL01** | 68 | 30 | 20 |
| **ACL02** | 136 | 31 | 22 |
| **ACL03** | 340 | 34 | 24 |
| **ACL04** | 500 | 34 | 24 |
| **ACL05** | 1,000 | 36 | 26 |
| **ACL06** | 1,530 | 36 | 26 |
| **ACL07** | 1,945 | 36 | 26 |

# Spatial Performance

Memory Usage (Unit: Byte)

| RULESET | #RULE | FW+IDS | IPP |
|---|---|---|---|
| **ACL01** | 68 | 563,350 | 563,484 |
| **ACL02** | 136 | 584,680 | 584,944 |
| **ACL03** | 340 | 672,922 | 673,492 |
| **ACL04** | 500 | 609,880 | 610,664 |
| **ACL05** | 1,000 | 1,002,096 | 1,003,690 |
| **ACL06** | 1,530 | 898,422 | 899,902 |
| **ACL07** | 1,945 | 937,998 | 939,586 |

# Hardware Performance

Throughput (Unit: Gigabits/Second)

| RULESET | #RULE | FW+IDS | IPP |
|---|---|---|---|
| **ACL01** | 68 | 3.72 | 4.64 |
| **ACL02** | 136 | 3.55 | 4.48 |
| **ACL03** | 340 | 3.37 | 4.46 |
| **ACL04** | 500 | 3.28 | 4.37 |
| **ACL05** | 1,000 | 3.10 | 4.03 |
| **ACL06** | 1,530 | 3.19 | 4.04 |
| **ACL07** | 1,945 | 3.16 | 3.97 |

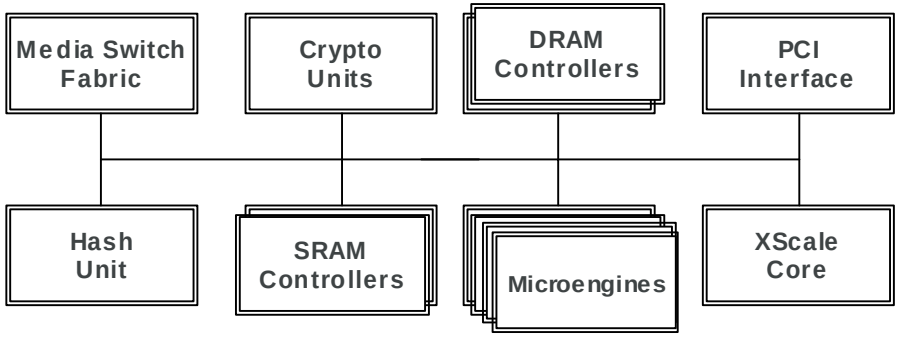
# Outline

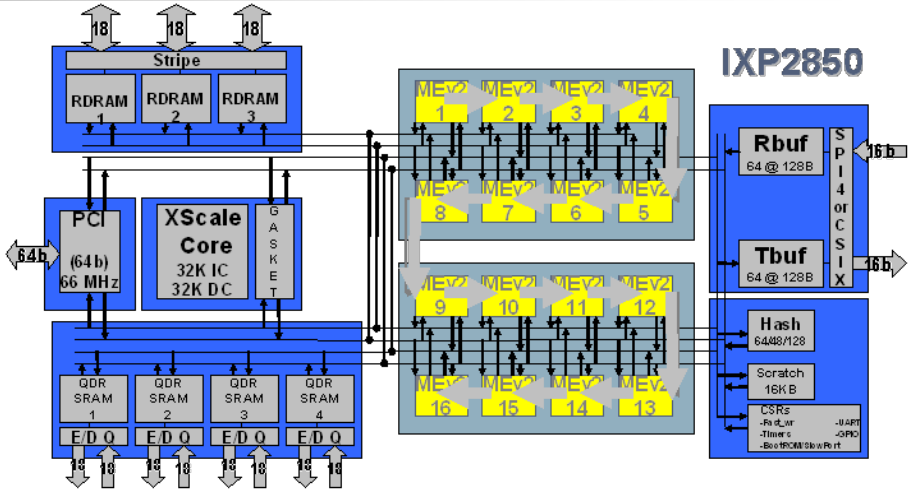
- Introduction
- Packet Classification Algorithms
- Pattern Matching Algorithms
- Integrated Framework
- **Network Processor Implementation**
- Summary

# Hardware: Architecture Limitation

- TCAM
  - Board area
  - Power
  - Range matching
- ASIC/FPGA
  - R&D cost
  - Update
- General Purpose CPU
  - Lack of integrated networking processing power
- Network Processor (NP)
  - Highly integrated processing units
  - Date plane & control plane
  - Handle rarely associative network traffics

# Intel IXP2850 NP



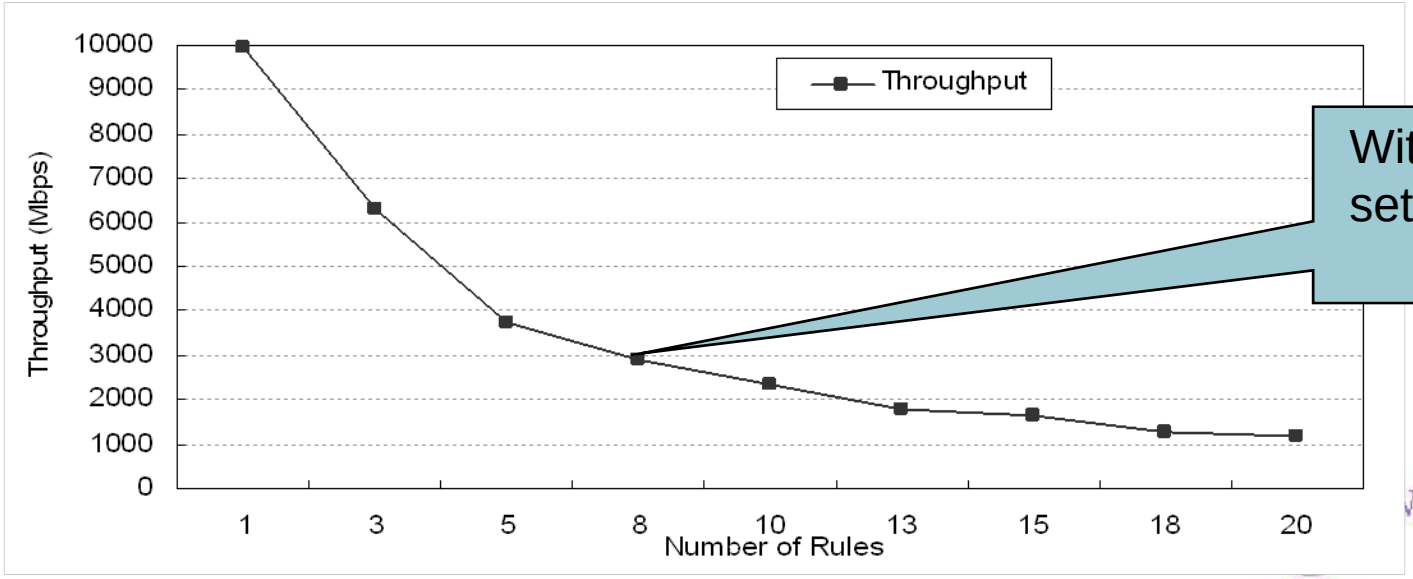

- Intel XScale core
  - 1 general purpose 32-bit RISC processor
- Multithreaded microengines:
  - 16 MEs working in parallel at 1.4 GHz clock frequency
- Memory hierarchy
  - 4 channels of QDR SRAM running at 233 MHz
  - 3 channels of RDRAM running at 127.3 MHz
- Build-in media interfaces
  - 2 configurable 32-bit media switch interfaces

# Programming Challenges

- Achieving a deterministic bound on packet processing operation
  - Due to the line rate constraint, the number of clock cycles to process the packet on each NP cannot exceed an upper bound
  - Use the right kind of data structures, and limiting the total number of memory accesses
- Hiding memory access latency through multi-threading
  - Memory access latency is typically much higher than the amount of processing budget
  - Utilize the multiple hardware threads to hide memory latency effectively
- Preserve packet order in spite of parallel processing
  - Preserve packet order is extremely critical for applications like security gateways and traffic management
  - Packet ordering can be guaranteed using tags and/or strict thread ordering

# Why Not Existing Algorithms?

- For example, HiCuts algorithm for packet classification
  - Admittedly
    - HiCuts has good time/space tradeoffs and works well for real-life rule sets
  - However
    - HiCuts has non-deterministic worst-case search time
      - Because the number of cuttings varies at different tree nodes, the decision-tree may have inexplicit worst-case depth
    - HiCuts also requires linear search
      - Experimental results show that linear search is very time-consuming



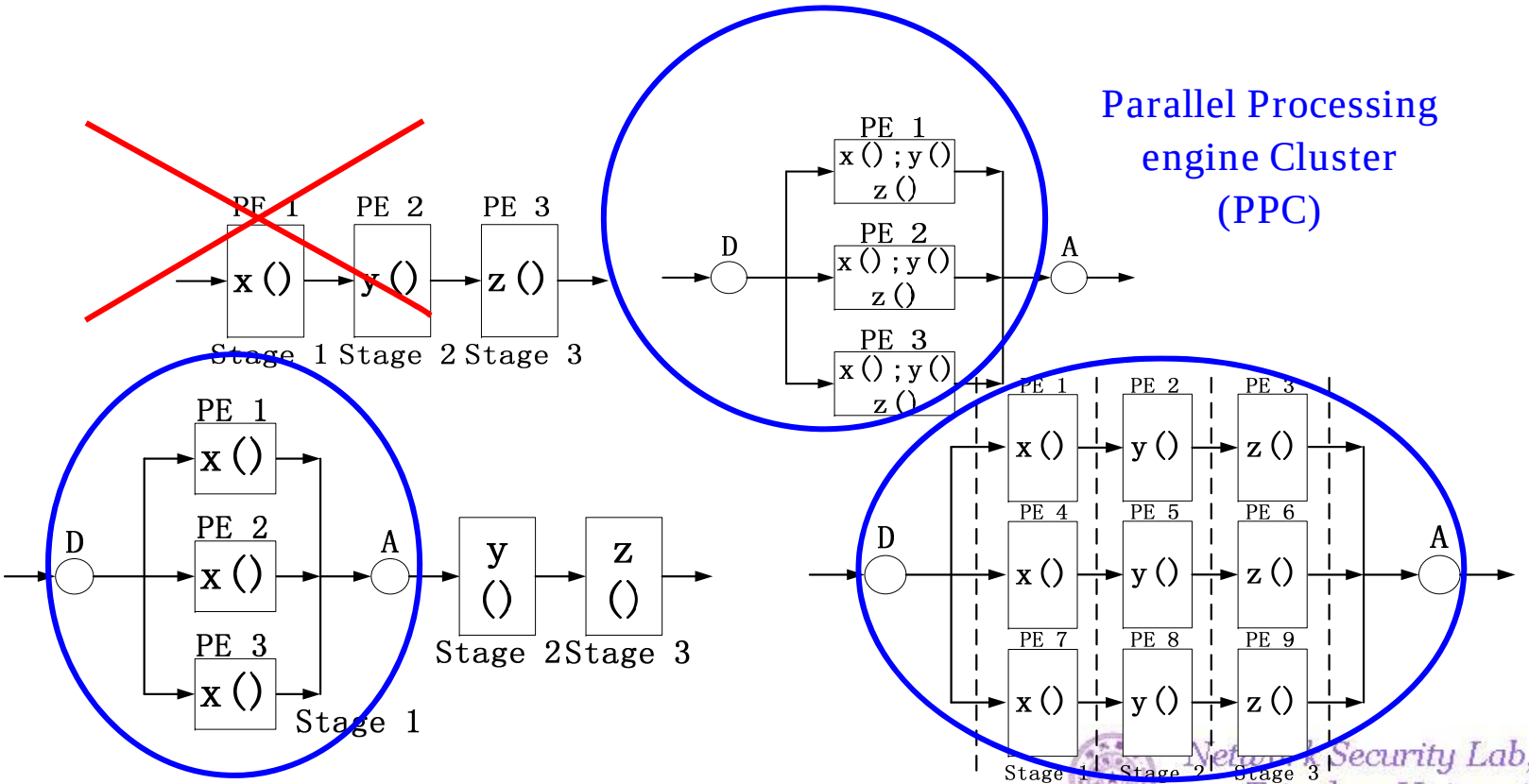

With HiCuts default setting, only 3Gbps throughput

# So We Design New Algorithms

- For example, ExpCuts
  - Fix the number of cuttings at internal-nodes
    - If the number of cuttings is fixed to $2^w$, the worst-case bound of tree depth is then $O(W/w)$
  - Eliminate linear search at leaf-nodes
    - Linear search can be eliminated if we "keep cutting" until every sub-space is **full-covered** by a certain set of rules
  - Performance estimation
    - The common 5-tuple packet classification problem (32-bit source/destination IP addresses, 16-bit source/destination port numbers and 8-bit protocol field)
    - If $w$ is fixed to be 8, then a (32+32+16+16+8)/8=13 worst-case search time is guaranteed, and no linear search is required

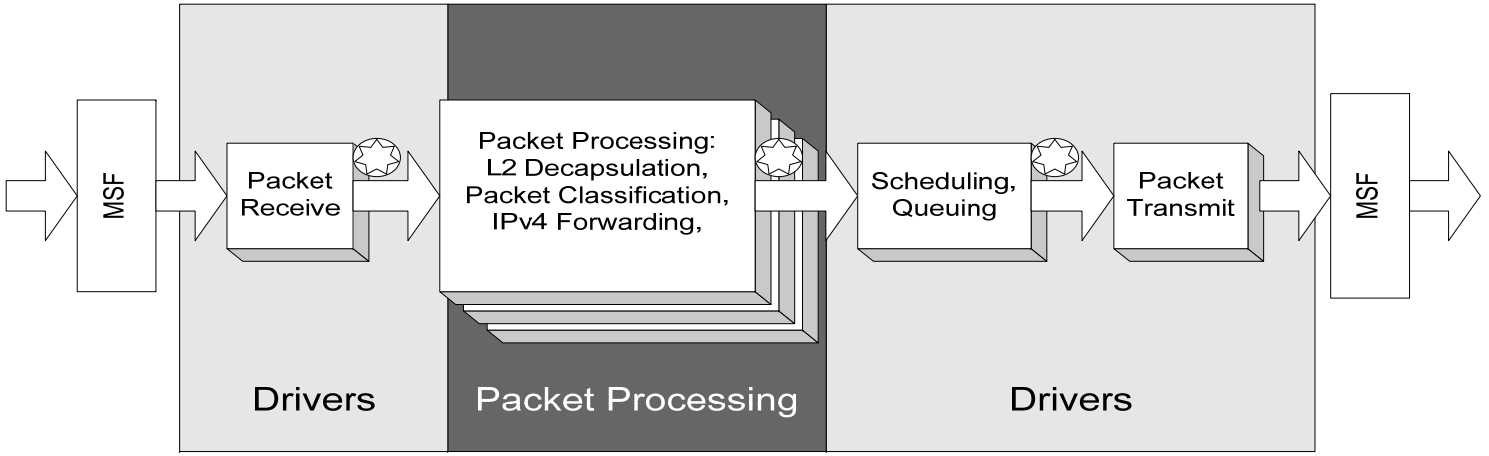
# Hardware Mapping (General)

- Pure pipeline model does not work well; Parallelism is a must in next-generation network processor design



# Hardware Mapping (Specific)

- Context-pipelining
  - Rx, Processing, Scheduling, Tx
  - Advantages: separate driver/processing codes
- Multi-processing
  - Packet processing
  - Advantages: scalability and per-packet data cache




| Task | Receive | Processing | Scheduling Queue | Transmit |
|---|---|---|---|---|
| Num. of MEs | 2 | 1~9 | 3 | 2 |

# Parallel Processing (General)

- How can a network processor maximize the utilization of its Paralleled Processing engines Cluster (PPC)?
  - Load-balancing: the processing task should be uniformly distributed across the processing engines
    - Our Solution: Flow-based Load-Balancing via Static/Dynamic Hashing (SQF-C)
  - Intra-flow packet ordering: the packets in the same flow should leave in their arrival order
    - Our Solution: Per-flow ordering without per-flow information
  - Memory contention: efficient memory subsystem should be developed to catch up with line rate processing
    - Our Solution: Distributed Memory Hierarchy

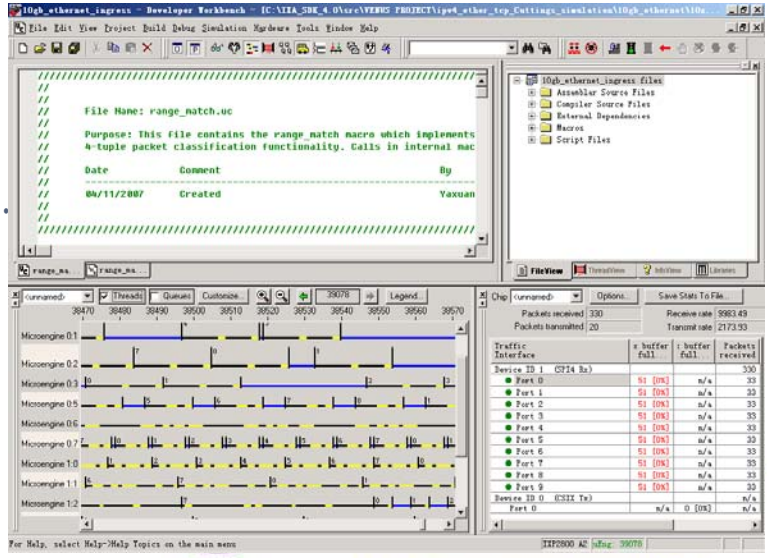
# Parallel Processing (Specific)

- Multi-channel memory allocation
  - Distribute different level of the decision-tree on different SRAM channels according to the bandwidth headroom of each channel
- Flow-level Packet Ordering
  - External Packet Ordering (EPO) by ordered-thread-execution
  - Internal Packet Ordering (IPO) by SRAM QArray
- Load Balancing
  - CRC hardware supported hashing
  - Flow-level fragment load balancing
- Instruction Selection
  - POP_COUNT can count the number of '1's in a 32-bit bit-string within only 3 system cycles
  - 10 times faster than other RISC implementations

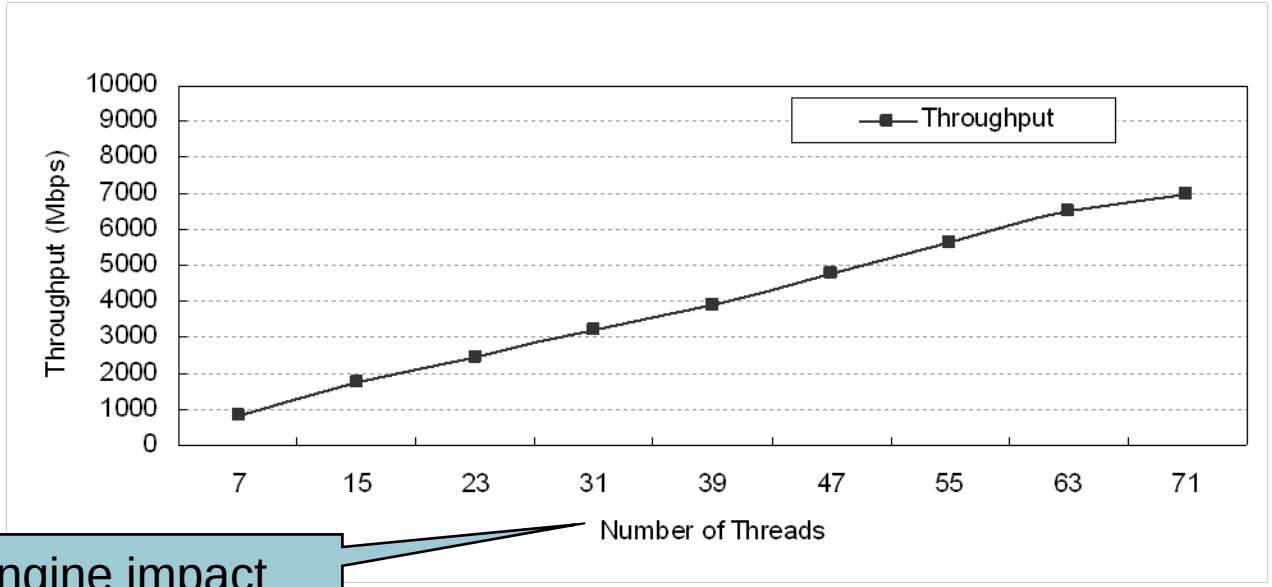
# Data-set and Development-Kits

- Rule Set Selection
  - Synthetic rule sets
    - Used by existing work
    - Algorithm-dependent performance
  - **Real-life rule sets**
    - More complex
    - Objective performance
- Development-Kits
  - Microengine C and IXP C
    - Compiler-dependent
    - Poor performance
  - **Microcode assembly**
    - High performance: MUTEX, Debug…
- Evaluation
  - **Software evaluation**
    - Cycle-accurate workbench
  - **Hardware evaluation**
    - Smartbit 600

| Set | #Rules | Length of Prefix |
|---|---|---|
| FW1 | 68 | University gateway firewall rules |
| FW2 | 136 | |
| FW3 | 340 | |
| CR1 | 500 | Large ISP core router ACLs |
| CR2 | 1,000 | |
| CR3 | 1,530 | |
| CR4 | 1,945 | |



# ExpCuts Performance




Multi-thread Microengine impact

Multi-channel SRAM impact

| #SRAM Channel | Throughput |
|---|---|
| 1 | 4963Mbps |
| 2 | 5357Mbps |
| 3 | 6483Mbps |
| 4 | 7261Mbps |

# Outline

- Introduction
- Packet Classification Algorithms
- Pattern Matching Algorithms
- Integrated Framework
- Network Processor Implementation
- **Summary**

# Summary (I)

- Packet Classification
  - “Fast enough”?
    - Worst-case bounded: Fixed stride decision tree
  - “Use not too much memory”?
    - Contiguous space aggregation: Bit-string compression
    - Non-contiguous space aggregation
- Y. Qi and J Li, “Towards Effective Packet Classification”, CNIS，2006.
- B. Xu, D. Jiang, and J. Li, “HSM: A Fast Packet Classification Algorithm,” AINA, 2005.
- Y. Qi and J. Li, “Dynamic Cuttings: Packet Classification with Network Traffic Statistics,” TIW, 2004.

# Summary (II)

- Pattern Matching
  - Automata-based algorithms
    - Set-wise AC algorithm: exploit real-life police structures
    - Trade speed for space: NFA with Bitmap compression
  - Hash-based algorithms
    - More effective/intelligent shift: RSI, MDH
    - Avoid very short patterns: hybrid algorithms with cache
- Z. Zhou,Y. Xue, J. Liu, W. Zhang, and J. Li, “MDH: A High Speed Multi-Phase Dynamic Hash String Matching Algorithm for Large-Scale Pattern Set,” ICICS, 2007.
- B. Xu, X. Zhou and J. Li, “Recursive Shift Indexing: A Fast Multi-Pattern String Matching Algorithm,” ACNS, 2006.

# Summary (III)

- Integrated Framework
  - High-performance UTMs should be optimized at system-level rather than simply stringed together a number of security applications
- Algorithmic Solution
  - The IPP algorithm avoids Redundant Packet Classification and Unnecessary Deep Inspection
- Hardware Evaluation
  - NP evaluation shows that our scheme outperforms existing algorithms with about 30% increase of throughput
- Y. Qi, B. Xu, F. He, B. Yang, J. Yu and J. Li, “Towards High-performance Flow-level Packet Processing on Multi-core Network Processors,” ANCS, 2007 (to appear) .
- Y. Qi, B. Yang, B. Xu and J. Li, “Towards System-level Optimization for High Performance Unified Threat Management,” ICNS, 2007.

# Summary (IV)

- Hardware-aware Implementation
  - New algorithms
    - Set-aware algorithms
    - Traffic-aware algorithms
  - New hardware
    - Multi- MIPS core network processors (large L2 cache)
    - FPGA or ASIC implementation
- Y. Qi, B. Xu, F. He, X. Zhou, J. Yu, and J. Li, “Towards Optimized Packet Classification Algorithms for Multi-Core Network Processors,” ICPP, 2007.
- L. Shi, Y. Zhang, J. Yu, B. Xu, B. Liu, and J. Li, “On the Extreme Parallelism Inside Next-Generation Network Processors,” INFOCOM, 2007.

# Thanks!
# Questions?

Network Security Lab, RIIT
Tsinghua University