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

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

2007.10.11
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

- **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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Field 1</th>
<th>Field 2</th>
<th>Field F</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.163.190.69/21</td>
<td>166.163.80.11/32</td>
<td>UDP</td>
<td>A1</td>
</tr>
<tr>
<td>2</td>
<td>192.168.3.0/24</td>
<td>166.163.0.0/16</td>
<td>TCP</td>
<td>A2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N</td>
<td>0.0.0.0/0</td>
<td>0.0.0.0/0</td>
<td>ANY</td>
<td>An</td>
</tr>
</tbody>
</table>
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
Related Work: HiCuts

- 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

```
To build the decision tree
  Stride #cuts
    Constant stride
      Not rule-adaptive
        No t/s tradeoff
  Variable stride
    Non-deterministic worst-case search time
      Long pre-processing time
```
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

```
<table>
<thead>
<tr>
<th>byte0</th>
<th>byte1</th>
<th>byte3</th>
<th>byte4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension to Cut</td>
<td>Position to Cut</td>
<td>HABS</td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Network Security Lab, RIIT
Tsinghua University
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
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

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Single</th>
<th>Multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KMP</td>
<td>AC</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>AC-Modified</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Single</th>
<th>Multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM</td>
<td>CW(AC_BM)</td>
<td>WM</td>
</tr>
<tr>
<td>BMH</td>
<td>SBMH</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Factor</th>
<th>Single</th>
<th>Multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDM</td>
<td>SBDM</td>
<td></td>
</tr>
<tr>
<td>BOM</td>
<td>SBOM</td>
<td></td>
</tr>
</tbody>
</table>

RSI

MDH
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

Performance = E(Shift) / E(Comparison)
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
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 $h_1$
  - Reduce table size from $2^{32}$ to $2^{a}$ ($a<32$, e.g. $a=20$)
- PMT table
  - Compressed hash function $h_2$
  - 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: $2^a+2^b$

<table>
<thead>
<tr>
<th>Pattern set size</th>
<th>WM($B=2$)</th>
<th>WM($B=3$)</th>
<th>MDH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ZR (%)</td>
<td>MEM (MB)</td>
<td>ZR (%)</td>
</tr>
<tr>
<td>10k</td>
<td>14.2</td>
<td>0.95</td>
<td>0.059</td>
</tr>
<tr>
<td>25k</td>
<td>31.7</td>
<td>1.91</td>
<td>0.149</td>
</tr>
<tr>
<td>50k</td>
<td>53.3</td>
<td>3.5</td>
<td>0.297</td>
</tr>
<tr>
<td>75k</td>
<td>68.0</td>
<td>5.09</td>
<td>0.446</td>
</tr>
<tr>
<td>100k</td>
<td>78.3</td>
<td>6.69</td>
<td>0.594</td>
</tr>
</tbody>
</table>
MDH: Dynamic-cut Heuristics

WM

**pattern**

**left**

**align**

**length**

MDH

**pattern**

**left**

**align**

**length**

optimum $m$ window
MDH: Performance

- Lower ZR and APM
- Reasonable preprocessing overhead
- Improved searching throughput
- Real-life pattern set from ClamAV

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>20k</th>
<th>40k</th>
<th>60k</th>
<th>77k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Thr (Mbps)</td>
<td>Mem (MB)</td>
<td>Thr (Mbps)</td>
<td>Mem (MB)</td>
</tr>
<tr>
<td>MDH</td>
<td>250.56</td>
<td>3.82</td>
<td>203.28</td>
<td>5.2</td>
</tr>
<tr>
<td>WM</td>
<td>329.52</td>
<td>3.33</td>
<td>126</td>
<td>5.2</td>
</tr>
<tr>
<td>SBOM</td>
<td>69.68</td>
<td>81.87</td>
<td>56.16</td>
<td>162.5</td>
</tr>
</tbody>
</table>
MDH: Performance

- Searching time

![Graph showing searching time vs pattern number (K)]
MDH: Performance

Memory requirement

![Graph showing memory requirement vs. pattern number for different methods: AC, AC_BM, WM, SBM, SBOM, and MDH. The graph indicates a linear relationship between memory usage and pattern number. Each method has a distinct line on the graph, with AC_BM showing the highest memory usage, followed by SBM and SBOM, and then AC, WM, and MDH showing the least memory usage.](image-url)
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 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
IPP handles two independent rule sets by
- Integrated space segmentation
- Independent space aggregation
# Temporal Performance

## Memory Access (Unit: 32bit-word)

<table>
<thead>
<tr>
<th>RULESET</th>
<th>#RULE</th>
<th>FW+IDS</th>
<th>IPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL01</td>
<td>68</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>ACL02</td>
<td>136</td>
<td>31</td>
<td>22</td>
</tr>
<tr>
<td>ACL03</td>
<td>340</td>
<td>34</td>
<td>24</td>
</tr>
<tr>
<td>ACL04</td>
<td>500</td>
<td>34</td>
<td>24</td>
</tr>
<tr>
<td>ACL05</td>
<td>1,000</td>
<td>36</td>
<td>26</td>
</tr>
<tr>
<td>ACL06</td>
<td>1,530</td>
<td>36</td>
<td>26</td>
</tr>
<tr>
<td>ACL07</td>
<td>1,945</td>
<td>36</td>
<td>26</td>
</tr>
</tbody>
</table>
## Spatial Performance

### Memory Usage (Unit: Byte)

<table>
<thead>
<tr>
<th>RULESET</th>
<th>#RULE</th>
<th>FW+IDS</th>
<th>IPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL01</td>
<td>68</td>
<td>563,350</td>
<td>563,484</td>
</tr>
<tr>
<td>ACL02</td>
<td>136</td>
<td>584,680</td>
<td>584,944</td>
</tr>
<tr>
<td>ACL03</td>
<td>340</td>
<td>672,922</td>
<td>673,492</td>
</tr>
<tr>
<td>ACL04</td>
<td>500</td>
<td>609,880</td>
<td>610,664</td>
</tr>
<tr>
<td>ACL05</td>
<td>1,000</td>
<td>1,002,096</td>
<td>1,003,690</td>
</tr>
<tr>
<td>ACL06</td>
<td>1,530</td>
<td>898,422</td>
<td>899,902</td>
</tr>
<tr>
<td>ACL07</td>
<td>1,945</td>
<td>937,998</td>
<td>939,586</td>
</tr>
</tbody>
</table>
# Hardware Performance

## Throughput (Unit: Gigabits/Second)

<table>
<thead>
<tr>
<th>RULESET</th>
<th>#RULE</th>
<th>FW+IDS</th>
<th>IPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL01</td>
<td>68</td>
<td>3.72</td>
<td>4.64</td>
</tr>
<tr>
<td>ACL02</td>
<td>136</td>
<td>3.55</td>
<td>4.48</td>
</tr>
<tr>
<td>ACL03</td>
<td>340</td>
<td>3.37</td>
<td>4.46</td>
</tr>
<tr>
<td>ACL04</td>
<td>500</td>
<td>3.28</td>
<td>4.37</td>
</tr>
<tr>
<td>ACL05</td>
<td>1,000</td>
<td>3.10</td>
<td>4.03</td>
</tr>
<tr>
<td>ACL06</td>
<td>1,530</td>
<td>3.19</td>
<td>4.04</td>
</tr>
<tr>
<td>ACL07</td>
<td>1,945</td>
<td>3.16</td>
<td>3.97</td>
</tr>
</tbody>
</table>
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 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

<table>
<thead>
<tr>
<th>Task</th>
<th>Receive</th>
<th>Processing</th>
<th>Scheduling Queue</th>
<th>Transmit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of MEs</td>
<td>2</td>
<td>1–9</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
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

### Table: Rule Set Selection

<table>
<thead>
<tr>
<th>Set</th>
<th>#Rules</th>
<th>Length of Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW1</td>
<td>68</td>
<td>University gateway firewall rules</td>
</tr>
<tr>
<td>FW2</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>FW3</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>CR1</td>
<td>500</td>
<td>Large ISP core router ACLs</td>
</tr>
<tr>
<td>CR2</td>
<td>1,000</td>
<td></td>
</tr>
<tr>
<td>CR3</td>
<td>1,530</td>
<td></td>
</tr>
<tr>
<td>CR4</td>
<td>1,945</td>
<td></td>
</tr>
</tbody>
</table>
ExpCuts Performance

### Throughput Table

<table>
<thead>
<tr>
<th>#SRAM Channel</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4963Mbps</td>
</tr>
<tr>
<td>2</td>
<td>5357Mbps</td>
</tr>
<tr>
<td>3</td>
<td>6483Mbps</td>
</tr>
<tr>
<td>4</td>
<td>7261Mbps</td>
</tr>
</tbody>
</table>

### Graph

- **Throughput**
- **Number of Threads**

**Legend:**
- Multi-thread Microengine impact
- Multi-channel SRAM impact

Network Security Lab, RIIT
Tsinghua University
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
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


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.


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
Thanks!
Questions?