

Efficient Packet Classification for Intrusion Detection Using FPGA

Haoyu Song, John W. Lockwood



*Applied Research Lab : Reconfigurable Network Group
Department of Computer Science and Engineering
<http://www.arl.wustl.edu/arl/projects/fpx/reconfig.htm>*

The research was funded by a grant from **Global Velocity**.



<http://www.globalvelocity.com/>

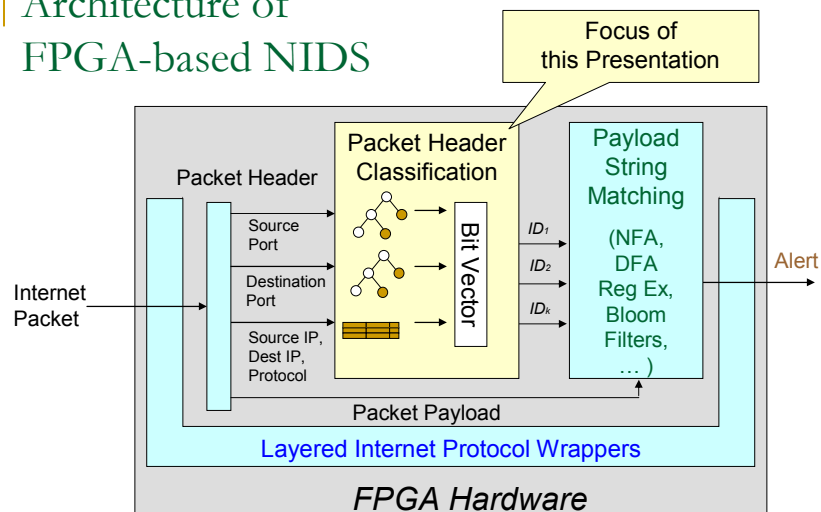
Network Intrusion Detection System (NIDS)

- Device that detects network activity symptomatic of an attack to network and computer systems.
- Critical part of a unified threat management system
- Performs
 - Protocol Processing
 - Packet Header Classification
 - Content Inspection (string matching)
- Traditionally Implemented as software on a PC, but can be implemented as hardware in an FPGA

Motivation & Challenges

- FPGAs proven effective for content scanning & string matching
 - High throughput
 - Great Flexibility
- FPGAs can be effective for header processing as well
 - There is a need for efficient packet header classification
 - Needed to block Denial of Service (DoS) attacks
 - Integral part of Intrusion Detection System
 - Linear search is not practical for a large header rule set.
 - Software-based system can't keep up with high-speed networks
 - Brute-force TCAM Implementations are inefficient on FPGAs
- Desirable properties of header processing circuits
 - Avoid use of off-chip memory
 - Prefer simple algorithm and architecture

Architecture of FPGA-based NIDS



Packet Classification

ID	Source IP	Destination IP	Protocol	Source Port	Destination Port
1	any	192.168.0.0/16	TCP	≥ 1024	2589
2	any	192.168.0.0/16	TCP	10101	any
3	128.252.158.203	192.168.50.2	TCP	any	443
4	192.158.0.0/16	any	UDP	49230	60000
5	any	any	TCP	any	< 110
6	any	any	TCP	146	1000:1300

- Header Rule
 - IP fields are specified as prefix
 - Protocol field is specified as exact value or wildcard
 - Port fields are specified as **arbitrary range** or exact value
 - Rules may share same specification for some fields
 - Rules may overlap
- Rule Matching
 - A rule is matched by a packet if all the corresponding fields are matched in the specified way
 - One packet can match multiple rules

Existing Packet Classification Schemes

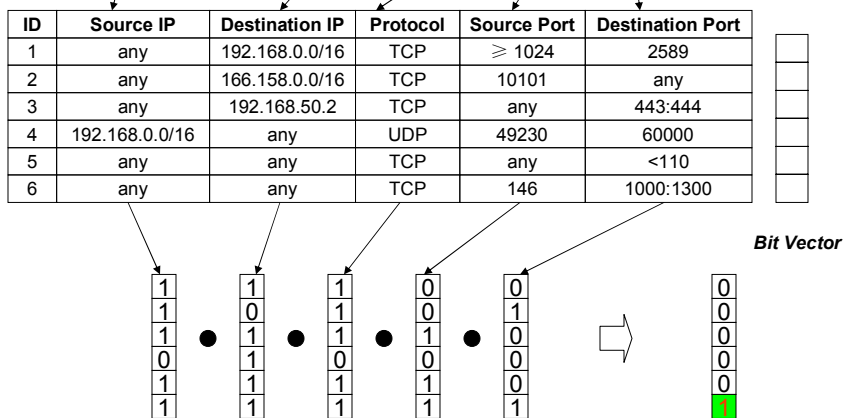
- Algorithmic solutions:
(e.g.: *HyperCuts, Aggregated Bit Vector*)
 - Poor worst case performance, or
 - Excessive memory usage
- Hardware (TCAM) solutions:
(e.g.: *Extended TCAM, Parallel Packet Classification*)
 - Lower density of entries
 - High power consumption
 - Cannot directly represent **arbitrary ranges**
 - Converting range to prefixes expands the rule set
- A hybrid architecture is more efficient

Characteristics of Snort NIDS Rule Set

- Snort
 - Open source Network Intrusion Detection System
- Characteristics of Version 2.3.0 (September, 2004)
 - 2464 rules
 - 274 unique header rules
- Trends in growth
 - The number of rules increases 4 times in 4 years
 - However, the number of unique header rules stays relatively constant

Illustration of Bit Vector (BV) Algorithm

- Given input packet with { S.IP, D.IP, Proto, S.Port, D.Port } = {128.252.160.245, 192.168.50.2, TCP, 146, 1200}

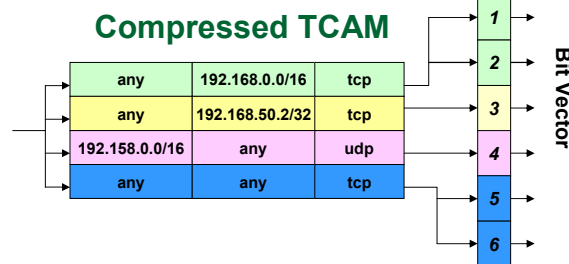


Our solution – BV-TCAM

- Utilizes Xilinx Coregen *TCAM* component
 - Unencoded output is exactly the Bit Vector we want
- Avoids rule set expansion by excluding source and destination port fields from TCAM
 - All other fields generate a unified *Bit Vector* from TCAM
 - Uses *Tree Bitmap* to implement the *Bit Vector algorithm* that classifies the port fields
- Matches rules using results from both *TCAM & Tree Bitmap lookup engines*

Original Header Rule Table

ID	Source IP	Destination IP	Protocol	Source Port	Destination Port
1	any	192.168.0.0/16	TCP	≥ 1024	2589
2	any	192.168.0.0/16	TCP	10101	Any
3	any	192.168.50.2	TCP	any	443
4	192.158.0.0/16	any	UDP	49230	60000
5	any	any	TCP	any	<110
6	any	any	TCP	146	1000:1300

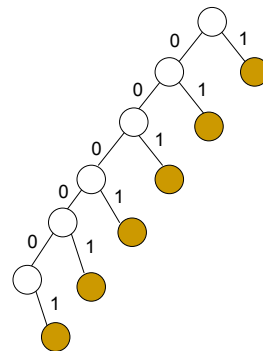


Compressed TCAM Implementation

- Built with Xilinx TCAM core
 - Utilizes SRL16E components
 - Performs lookup in one clock cycle
 - Content can be updated in only a few clock cycles
- For snort rule set, only 33 distinct entries, each of 72 bits, need to be programmed.
 - Coregen TCAM core uses 1188 SRL16Es
 - Only 3% of SRL16Es in XCV2000E

Store Port Field Bit Vectors in a Binary Trie

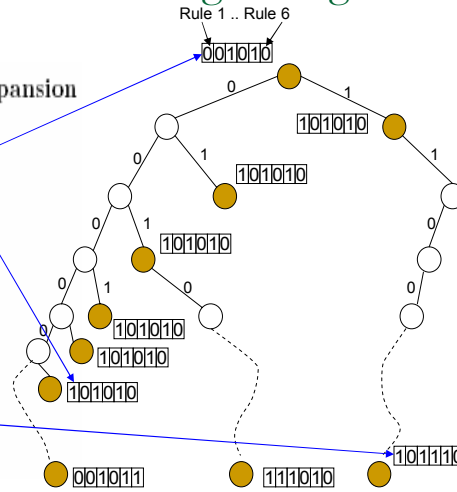
- Port ranges expand to prefixes, as they did with a TCAM
 - e.g. Expanding Port number ≥ 1024 would have required 6 TCAM Entries
 - 0000 01** **** ** (1024~2047)
 - 0000 1*** **** ** (2048~4095)
 - 0001 **** **** ** (4096~8191)
 - 001* **** **** ** (8192~16383)
 - 01** **** **** ** (16384~32767)
 - 1*** **** **** ** (32767~65535)
 - But, each prefix is just inserted in a prefix tree
 - Each valid prefix node now contains a Bit Vector



Retrieving Bitmap Vector through Longest Prefix Matching

Example Source Port Prefixes Expansion

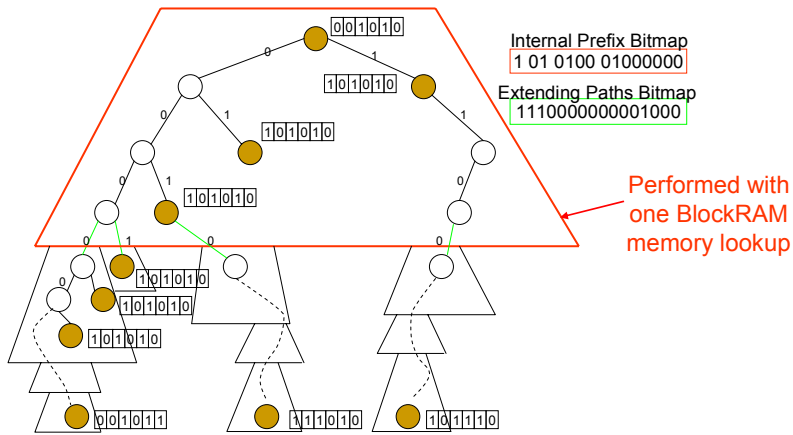
Prefix	Rule ID
*	3,5
0000 01** *****	1 (and 3,5)
0000 1*** *****	1 (and 3,5)
0001 *****	1 (and 3,5)
001* *****	1 (and 3,5)
01** *****	1 (and 3,5)
1*** *****	1 (and 3,5)
0010 0111 0111 0101	2 (and 1, 3,5)
1100 0000 0100 1110	4 (and 1, 3,5)
0000 0000 1001 0010	6 (and 3,5)



Source Port: **49230** → 1100000001001110

Efficient Tree Bitmap Implementation

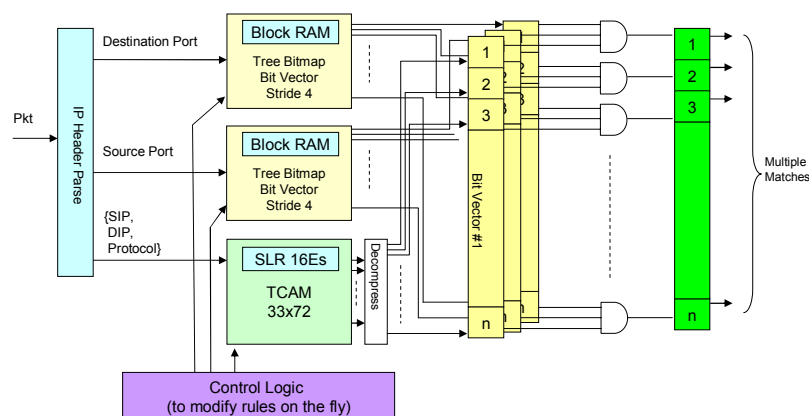
- Multi-bit Trie enables traversing multiple bits per memory access
- Tree Bitmap is an efficient hardware implementation of Multi-bit Trie



Tree Bitmap Implementation Results (with 4-bit stride)

- **Statistics**
 - 56 distinct source port ranges → 87 distinct prefixes
 - 143 tree nodes for source port
 - 124 distinct destination port ranges → 177 distinct prefixes
 - 400 tree nodes for destination port
- **Resource Usage**
 - Data structure for both tries use < 100 Kbits of Block RAMs
 - ≤15% of total available memory
 - The control logic uses less than 2% of resources
- **Worst-case Lookup time**
 - Compressed TCAM : 1 clock cycle
 - Trie Lookup : 4 memory lookups in 8 clock cycles

BV-TCAM Architecture



Results on Snort Rule Set

- Synthesis results on Xilinx Virtex XCV2000E-8
 - Throughput
 - OC48 (2.5 Gigabit/second link speed)
 - Circuit runs at 100MHz clock rate with a 32-bit data width
 - Tree Bitmap algorithm requires 4 memory lookups to classify packet in worst case
 - Circuit uses 8 clock cycles to classify a packet in worst case
 - Lookup engine can perform 12.5M lookups/second
 - FPX platform can process 8M minimum-length packets per second
 - Resource Usage
 - < 10% of XCV2000E logic resources
 - < 20% of XCV2000E block RAMs
- Projected speed up
 - Utilize multiple tree-bitmap lookup engines
 - Dual engines could interleave the accesses to BlockRAMs to perform 25M lookups/second
 - Deploy circuit on Virtex 4 rather than VirtexE
 - Assuming 4x Speedup of (Virtex4 / VirtexE), achieve 100M Lookups/second

Conclusion

- BV-TCAM architecture designed to provide efficient packet classification in NIDS
 - High Throughput
 - Performs matching with parallel circuits
 - Utilizes both TCAM & Multibit Trie LPM
 - Low Resource Consumption
 - Reduces resource usage by using Compressed TCAM
 - Minimizes memory usage by using Tree Bitmap Algorithm for port field Bit Vector retrieval
- BV-TCAM circuit evaluated with Snort ruleset
 - Occupies < 20% of a Virtex 2000E FPGA
 - Classifies 8M packets/second on FPX platform
 - Scales to classify 100M packets/second