The 28th International Conference on Distributed Computing Systems

Fast Path Session Creation on Network Processors

Bo Xu[1, 2], Yaxuan Qi[1, 4], Fei He[1, 2], Zongwei Zhou[1, 3] Yibo Xue[1, 4] and Jun Li[1, 4]

[1]Research Institute of Information Technology, Tsinghua University
[2]Department of Automation, Tsinghua University
[3]Department of Computer Science and Technology, Tsinghua University
[4]Tsinghua National Lab for Information Science and Technology
xb00@mails.tsinghua.edu.cn

Abstract

The security gateways today are required not only to block unauthorized accesses by authenticating packet headers, but also by inspecting connection states to defend against malicious intrusions. Hence session creation rate plays a key role in determining the overall performance of stateful intrusion prevention systems. In this paper, we propose a high-speed session creation scheme optimized for network processors. Main contribution includes: a) A high-performance flow classification algorithm on network processors; b) An efficient TCP three-way handshake scheme designed for fast-path processing using a two-stage intelligent hashing. Experimental results show that: a) The presented parallel optimized flow classification algorithm, Parallel Search Cross-Producting, outperforms the original Cross-Producting and Binary Search Cross-Producting algorithms with 300% and 60% increase of classification speed; b) The proposed fast path three-way handshake scheme, IntelliHash, achieves a TCP connection creation rate over 2M connections per second.

1. Introduction

Nowadays prevailing network security devices, such as firewalls, NIPS and UTMs, have raised the requirement of supporting stateful deep inspection, in which session processing is one of the most important building blocks. The two critical components of session processing scheme are session creation and session update modules. As a crucial performance metrics, session creation rate is attracting more and more attention of network security vendors besides the number of concurrent sessions. However, due to the complicated processing stages, reaching high-speed session creation to meet the ever-increasing network bandwidth still remains an open issue and motivates the research today. With the emerging of network processor as a competent alternative to deal with network traffic at transportation layer, the performance of session creation has a promising chance to be accelerated.

Many integrated circuit companies, such as Intel [1], AMCC [2], Freescale [3], and Agere [4] have provided their own programmable network processors, while Cavium [5] and RMI [6] have also developed their multiple MIPS cores for accelerating the network processing. So there is a big challenge for researchers to build high-performance network security devices with today's multi-core and multi-threaded network processors.

Different from traditional architecture based on general-purpose processors, packet processing on network processors usually has two different routes: data plane processing and control plane processing. Data plane tasks are performance-critical, and typically require straightforward processing per packet, while control plane tasks are usually complex and considered difficult to be implemented in the data plane [7]. In network processing, data plane processing is treated as fast path and control plane processing is treated as slow path. Because the session creation requires complicated processing such as flow classification and three-way handshakes, it is often implemented in the slow path of today's network processors.

In order to improve the session creation performance, an intuitive idea is to treat session creation as a fast path module. However, fast-path implementation is inherently hard because:

✧ Due to no OS running on fast-path cores, all the related operations in low-level network layers, such as packet classification and TCP three-way handshake, need to be implemented and optimized by the developers, according to hardware specifications of different NPs.

✧ Fast-path session creation requires light-weight data structure and high-speed state maintenance. Besides, to ensure holistic performance, the session creation module should not affect other fast-path functions, such as session update and teardown.

1063-6927/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDCS.2008.33

This paper investigates the fast-path session creation solution and proposes a high-speed session creation scheme based on a typical multi-core and multi-threaded network processor. Main contribution includes:

✧ An effective flow classification algorithm is developed to support fast path parallel processing. The algorithm, named Parallel Searching Cross-Producting (PSCP), makes optimization on space mapping based on the Cross-Producting algorithm [8] to improve the time performance on NP.

✧ An efficient TCP three-way handshake scheme is presented for fast path implementation. The Intelligent Hash (IntelliHash) scheme separates the processing and maintenance of half-open sessions takes the advantage of separation between SRAM and DRAM channels on NP.

The rest of this paper is organized as follows: Section 2 gives a background of existed research. Section 3 describes the PSCP flow classification algorithm and Section 4 presents the IntelliHash scheme for three-way handshakes. Experimental results are discussed in Section 5. And in section 6, we draw our conclusion and discuss the future work.

2. Background

In this paper, we mainly discuss how to effectively implement TCP session creation mechanisms on network processors. So we first give a brief introduction of a typical multi-core NP and then discuss existed work related to high-speed session creation.

2.1. NP Architecture and Challenges

Network processors are designed to scale up with high data processing rate characterized by distributed, multi-core, multi-threaded architectures, which are beneficial for hiding memory latencies. This section gives a brief overview of a typical multi-core network processor, the Intel IXP2850 NP.

Figure 1 extracts the components of the Intel IXP2850 network processor, which includes 1 XScale core, 16 Micro-engines (MEs), 4 SRAM controllers, 3 DRAM controllers, and high-speed bus interfaces. The XScale core is a general purpose 32-bit RISC processor taking charge of initializing and managing the MEs. Each ME has eight hardware-assisted threads and uses the shared buses to access off-chip SRAM (4 channels, 256MB) and DRAM (3 channels, 2GB). The average access latency for SRAM is about 150 cycles, and that for DRAM is about 300 cycles. Detailed information of IXP2850 can be referred in [9-12].

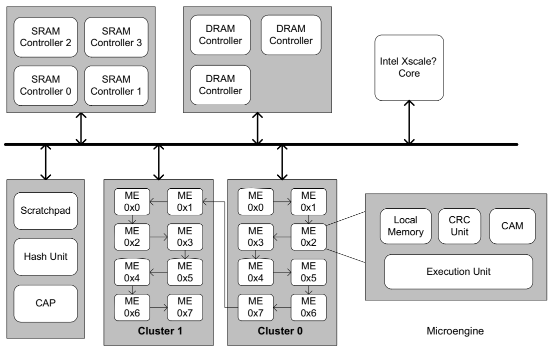


Figure 1. Hardware blocks of Intel IXP 2850

2.2. Session Creation on Network Processors

Generally, session creation includes two main steps: flow classification for the SYN packets and state tracking for TCP three-way handshakes.

2.2.1. The Flow Classification Problem

In existing literatures, a variety of flow classification algorithms are proposed [15-19]. All these algorithms can be categorized as field-independent search and field-dependent search [20]. In our research, we focus on field-independent algorithms because:

✧ Algorithms using field-independent search have better classification rate: Algorithms using field-independent search have explicit worst-case bound, which is the most important performance metrics required by the system.

✧ Algorithms using field-independent search is simple to implement: This type of algorithms only uses table lookups and some add/shift operations, which are more suitable for fast path implementation than those complicated operations required by field-dependent algorithms.

✧ In addition, although field-independent algorithms often require large memory storages, today's new-generation multi-core NPs can satisfy this requirement. For example, the Intel IXP 2850 NP has up to 256MB SRAM and 2GB DRAM, which is far more sufficient for most field-independent algorithms.

Many field-independent search algorithms are based on the Cross-Producting algorithm proposed by Srinivasan et al. [8]. These algorithms perform independent searches on each packet header field; and combine the results of the single-field searches by Cross-Producting. Then, Cross-Producting performs longest-prefix-matching on each field, resulting in O(W) search time, where W is the packet header bit-width. As an improvement, HSM [17] uses binary search to find the best-match on each fields, achieving O(d*$\log_2(N)$) search speed. Our research focuses on how to further improve the performance of the Cross-Producting algorithm on multi-core network processors.

2.2.2. TCP Three-way Handshake Handling

To our knowledge, today's most NP-based network security products handle the TCP three-way handshake in slow path. This means, incoming packets not belonging to the established sessions will be sent from the fast path to the slow path. Session creation in slow path typically results in: a) Memory copies for packet or packet handling duplication; b) Traversing of a full TCP/IP stack for state tracking; c)Locking and unlocking global session table for mutual exclusion.

Therefore, the three-way handshake processing will be slow down due to these redundant processing mechanisms. In this paper, we strive to realize a fast path TCP three-way handshake scheme on network processors by simplified TCP state maintenance, separated session state table, and elaborated data structures.

3. Fast Flow Classification on NP

This section introduces a fast flow classification algorithm optimized for multi-core network processors. The basic ideas are based on the algorithm Cross-Producting, which is proposed by Srinivasan et al. [8].

3.1. Basic Ideas

Cross-Producting employs the divide-and-conquer strategy, which first performs longest-prefix-matching at each field using trie structures, and then searches a cross-product table for every possible combination of results from the d field longest-prefix searches. If we have the rules as shown on the left of Figure 2, the right picture in Figure 2 depicts the single field search of the first step in Cross-Producting.

Although the divide-and-conquer strategy introduced by Cross-Producting simplifies the processing of multi-field flow classification, the two steps have inherent weakness that limits the overall performance:

✧ In the first step, doing longest-prefix search on each field is time-consuming due to the O(dW) complexity, where W is the bit-width of each field. For 5-tuple flow classification, the worst-case memory access of Cross-Producting is over 100, which is too many to reach ideal performance even on new-generation multi-core network processors [21]. Moreover, because some rules are specified by ranges rather than prefix, using longest-prefix-matching for search require range-to-prefix conversion, which will cause additional memory storage [22].

✧ In the second step, the cross-product table suffers from exponential memory requirements. This is because, for a set of N filters containing d fields each, the size of the cross-product table can grow to O(N^d). It is considered that Cross-Producting can only handle less than 100 classification rules [20, 22]. However, real-life rule sets on modern firewalls and security gateways may have tens of thousands rules, making Cross-Producting infeasible in practice.

Rule	Prefix	Range
R1	******	00~63
R2	0000**	00~03
R3	001***	08~15
R4	0011**	12~15
R5	010111	23~23
R6	0110**	24~27
R7	10****	32~47
R8	1000**	32~35
R9	1001**	36~39
R10	110100	52~52
R11	1111**	60~63

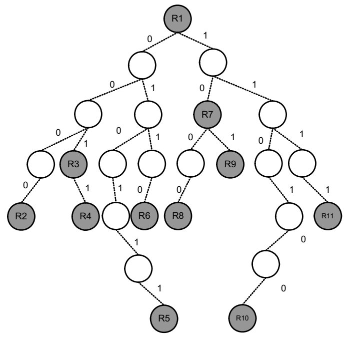


Figure 2. Flow Classification Rules and Longest-Prefix-Matching Trie

Therefore, to seek a more effective flow classification algorithm, we should not only improve the single-field search speed, but also limit the size of the cross-product table for multiple field combination result.

3.2. Parallel Search Cross-Producting

3.2.1. Improving Single-field Search Speed

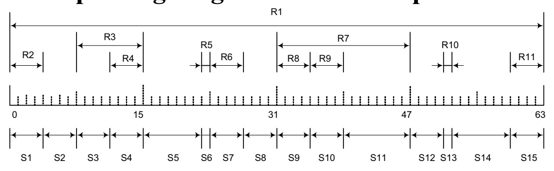


Figure 3. Single-Field Segmentation

To improve the single-field search speed, we consider using range-match to replace the prefix matching in Cross-Producting. The basic idea of range-match is to segment the search space by rule projections on each field [16] [17]. Figure 3 depicts the rule segmentation result with the rules in Figure 3. To find the best match of an incoming packet in a certain field, we only need to find the exact segment that the packet (represented as a dot in single-dimensional search space) falls in.

Intuitively, we can record all the boundaries (starting and end points) of the segments and using binary search to find the best match, which is illustrated in Figure 4. Because N rules generate at most $2N$-1 segments [22], the worst-case search time for binary search on a single field is $\log_2(2N\text{-}1)$. For example, given 1K rules, the worst-case binary search required in each field is 11, and totally 55 searches on all five dimensions of the 5-tuple packet header.

To further improve the single-field search speed, we can extend the binary search trie to a multi-search trie, which performs multiple balanced searches at a time on each field. For example, if we use quad-search trie shown in Figure 5 to find the best-match segment, the worst-case search time then becomes $\log_4(2N\text{-}1)=1/2*\log_2(2N\text{-}1)$, i.e. the search speed doubles. Note that using m times of balanced search at a time (m-search), each time we need to read (m-1) boundaries. For example, using quad-search to find the best match, each time we need to read three bounds to split the current search space into four sub-spaces. In comparison, binary search needs only one memory access at a time. Fortunately, modern multi-core network processors have built-in multi-channel SRAMs and burst-supported DRAMs, both of which provide the capability to access consecutive memory words once a time without extra latency [6, 9, 10]. Thus, because the m-1 boundaries are independent memory access, they can be loaded in parallel-mode from multiple memory channels or in burst-mode from a single channel of memory by issuing only one memory access.

Therefore, for the 32-bit source and destination IP fields, we choose to use multi-search rather than longest-prefix-matching to find the best match. This m-search reduces the memory access time from 32 to $\log_m(2N\text{-}1)$. When m=4 and N=1K, the exact memory access time for IP fields is less than 6, which is 5 times faster than longest-prefix-matching used by the original Cross-Producting algorithm. For the 16-bit source and destination port fields, we choose to use two 2^{16}-entry arrays to map each port to its best match segments, which are originally proposed by P. Gupta in the RFC algorithm [16]. Using such arrays the best-match for a 16-bit port field can be done by only one memory access.

3.2.2. Reducing Multi-field Cross-Product Table Size

To reduce the size of the cross-product table generated by Cross-Producting, we employ the Hierarchical Space Mapping [17] strategies, which effectively eliminated the redundant table entries generated by the single step Cross-Producting. The idea of multi-field Cross-Producting consists of: a) combining pairs of the single-field search results to generate 2-dimensional intermediate cross-product tables; b) combining pairs of these 2-dimensional cross-product results to generate next phase cross-product tables; c) recursively, PSCP generate the finally match results in the last cross-product table.

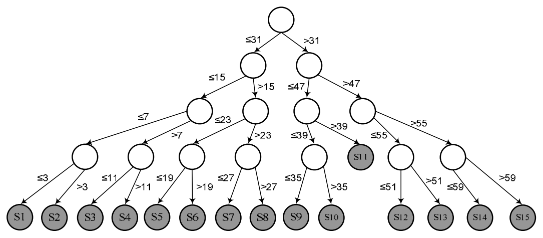


Figure 4. Binary Search Cross-Producting Tree

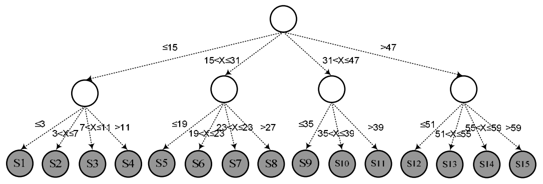
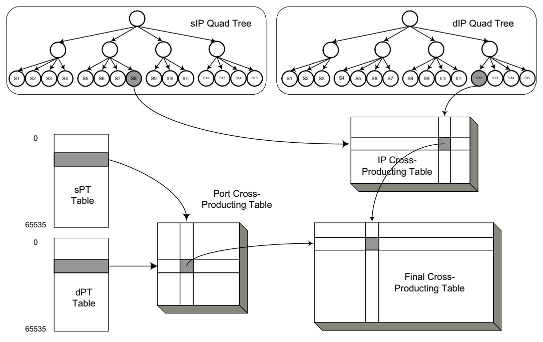


Figure 5. Quad Search Cross-Producting Tree



sIP Quad Tree
dIP Quad Tree
IP Cross-Producting Table
0
sPT Table
65535
Port Cross-Producting Table
0
dPT Table
65535
Final Cross-Producting Table

Figure 6. PSCP Data Structure

More specifically, in Figure 6, sIP/dIP quad trees and sPT/dPT tables are single field-search results; IP Cross-Producting Table (ICPT) is the cross-product table that combines the search results of sIP and sIP, and likewise, Port Cross-Producting Table (PCPT) is the cross-product table that combines the sPT and dPT search results; Final Cross-Producting Table (FCPT) combines the search results of ICPT and PCPT, giving output of the final search result. The protocol field is processed separately in PSCP.

The reasons why PSCP can significantly compress the original cross-product table are:

✧ Each cross-product table in PSCP is 2-dimemsional, which is exponentially smaller than the d-dimensional cross-product table generated by original Cross-Producting.

✧ Although mathematically, the final FCPT table is also $O(N^d)$ in magnitude, the actual FCPT is likely to be fairly small because the number of input of FCPT has been compressed by ICPT and PCPT.

3.3. PSCP on the IXP 2850 Network Processor

This section discusses how to implement PSCP on a typical multi-core network processor, the Intel IXP2850 NP. We only focus on the processing of source/destination IP fields and source/destination Port fields, because the protocol field and flag field are fairly simple and can be processed seperately [17]. The data structure of PSCP includes: a) two *m*-search balanced tree for sIP and dIP; b) two 2^{16}-entry port-to-segment array for sPT and dPT; c) three hierarchical 2-dimensional Cross-Producting tables: ICPT, PCPT and FCPT.

Because the IXP2850 NP has 4 independent SRAM channels, each supporting low-latency word-oriented memory accesses, we choose to use three of them to store the three boundaries and the left one to store the pointer to the next multi-search. Thus reading three boundaries together with a pointer can be done by issuing 4 memory accesses simultaneously, making the latency nearly the same as binary search while reducing half of the memory accesses. Because both the sPT and dPT tables have only 2^{16} entries, they can be stored in any of the 4 SRAM channels.

Considering the IXP2850 hardware, if the table size is less than 64MB, it can be stored in SRAM. Otherwise, it will be stored in DRAM. Because each table is accessed only once per packet, using DRAM does not significantly increase the overall latency.

4. IntelliHash Scheme For Fast Path Session Creation

This section presents a new session handling scheme optimized for fast path TCP three-way handshake on multi-core network processors. In this section, the TCP three-way handshake packets are denoted as *handshake packets* and subsequent packets are denoted as *data packets*.

4.1. Basic Ideas

The traditional hash scheme, which is called DirectHash in this paper, processes the three-way handshake packets and the subsequent data packets in a single hash table and uses link lists to handle hash collisions. Although DirectHash is easy to implement, this scheme suffers from:

✧ **Excessive Collisions between handshake and data packets processing:** Session creation and session update are processed through the same hash table in DirectHash scheme. This introduces additional hash collisions between handshake packets and data packets, which will bring excessive MUTEX locks in traversing the hash link lists.

✧ **Isomorphic session entries for handshake and data packets:** DirectHash maintains the same size of session entry for handshake packets and data packets. Assuming that 4M concurrent sessions were to support and each session entry is 256 bytes, DirectHash will consume at least 1G memory totally, which is too large compared to the 256M SRAM on IXP 2850. As a result, the hash table can only reside in DRAM banks, which have longer access latencies and hence limit the session creation rate.

To overcome these defections, an effective fast path session handling scheme should separate the session tables for handshake packets apart from data packets and design appropriate session entries for them each.

4.2. The IntelliHash Scheme

4.2.1. Separating Processing of Handshake Packets and Data Packets

Illumined by the idea of TCP half-proxy, we designed a novel hash scheme to isolate the processing of handshake packets from data packets. As shown in Figure 7, the IntelliHash scheme has two hash tables: the *Digested Session Table* in SRAM handles all the handshake packets processing in session creation and the *Full Session Table* in DRAM deals with data packet processing in session update. Both tables are indexed by the hash value of packet header returned by the hash unit. However, the two tables can be different in size and can be accessed simultaneously and independently. For instance, we can use the lower 20bits of the hash value to index the Digested Session Table, while use the lower 24bits of the hash value to index the Full Session Table.

Under such deployment, IntelliHash surpasses DirectHash by avoiding the excessive collisions between handshake and data packets processing. Consequently, the session update performance will not be impacted by the session creation process.

4.2.2. Differentiating Session Entries for Handshake Packets and Data Packets

TCP three-way handshake can be handled with relatively small amount of memory, which motivates us to design a standalone data structure for the handshake processing.

Figure 8 shows the 28 bytes entry data structure of the Digested Session Table in SRAM. The handshake state and the 5-tuple fields along with sequence number and ACK number are used for TCP three-way handshake. The MUTEX lock and next session pointer are used for handling session collisions and the timestamp and timeout threshold are used for tearing down the half-open session when it expires. Such a compact data structure enormously decreases the memory requirement for handling handshake packets and makes it possible to place the Digested Session Table entirely in low-latency SRAM banks.

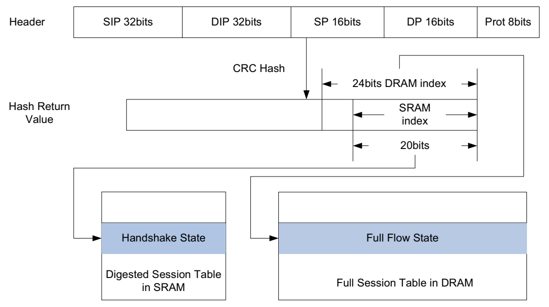
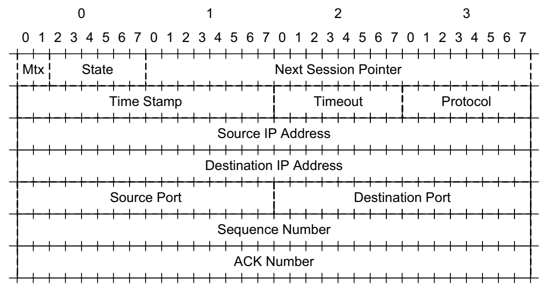


Figure 7. Data Structure of IntelliHash



Mtx indicates the 2 bits for MUTEX lock. State indicates the handshake state value.

Figure 8. Digested Session Table Entry in SRAM

The entries of Full Session Table could simply take the design of DirectHash, typically 200-300 bytes in size, which we do not extract here in detail.

4.3. IntelliHash on the IXP 2850 NP

4.3.1. Packet Processing in IntelliHash

Figure 9 gives a holistic packet processing route on IXP 2850, including three-way handshake packets and subsequent data packets. The processing procedure includes the following branches:

✧ SYN packet: handled by Digested Session Table in SRAM. If the half-open session already exists, update time stamp and forward the packet. Otherwise, create a half-open session entry in the Digested Session Table with the handshake state of SYN_RECV.

✧ SYN_ACK packet: handled by Digested Session Table in SRAM. If the half-open session exists with the state SYN_RECV and the sequence number and ACK number match with the previous ones in the session entry, set handshake state to SYN_ACK_RECV, update sequence number, ACK number, time stamp and forward it. Otherwise, drop the packet.

✧ Packet with ACK flag: First, check the Full Session Table. If the session exists: a) if sequence number and ACK number is in the TCP transmission window, update session entry and forward the packet; b) if sequence number and ACK number is not in the window, drop the packet. If the session does not exist: c) if payload length does not equal to zero, drop the

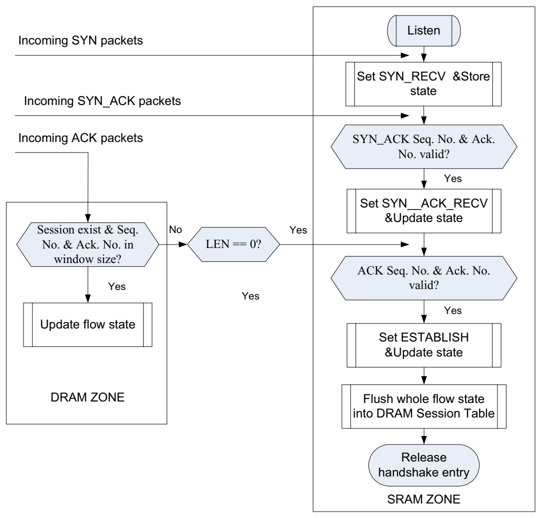

Incoming SYN packets
Incoming SYN_ACK packets
Incoming ACK packets
Listen
Set SYN_RECV &Store state
SYN_ACK Seq. No. & Ack. No. valid?
Yes
Set SYN__ACK_RECV &Update state
Session exist & Seq. No. & Ack. No. in window size?
No
LEN == 0?
Yes
Yes
Yes
Update flow state
ACK Seq. No. & Ack. No. valid?
Yes
Set ESTABLISH &Update state
Flush whole flow state into DRAM Session Table
Release handshake entry
DRAM ZONE
SRAM ZONE

Figure 9. Handshake Packets Processing

packet; d) if payload length equals to zero, use Digested Session Table to examine it sequentially.

In case d), the processing in Digested Session Table is as follows: e) if the half-open session exists with the state SYN_ACK_RECV and the sequence number and ACK number matches with the previous ones in the session entry, set handshake state to ESTABLISHED, update session entry and forward the packet; f) otherwise, drop the packet.

✧ Data packet without ACK flag: directly handled by Full Session Table in DRAM. If the session exists with sequence number and ACK number in TCP transmission window, update session entry and forward the packet. Otherwise, drop the packet.

The only overhead of IntelliHash scheme is that it forces all the packets with ACK flag to traverse the Full Session Table first. Fortunately, the session update speed is fast enough that the overhead can be properly concealed.

4.3.2. Session Table Size Adjusting

Assuming that new session creation rate is *C* (new sessions per second) and each entry of Digested Session Table is *D* bytes in size, with the load factor *L* and average session creation latency *T*, the total size of Digested Session Table will be $C*T*D/L$. Here, load factor *L* is defined by *N/M*, where *M* is the total number of buckets in the hash table and *N* is the number of concurrent sessions supported. Average session creation latency *T* means the time needed for completing the three-way handshake. Similarly, assuming *N* concurrent sessions were to be supported and each Full Session Table entry is *F* bytes in size, with the load factor *L*, the total size of Full Session Table will be $N*F/L$. For example, if *C* is 1M new sessions per second, *D* is 28 bytes, *T* is 1 second, *N* is

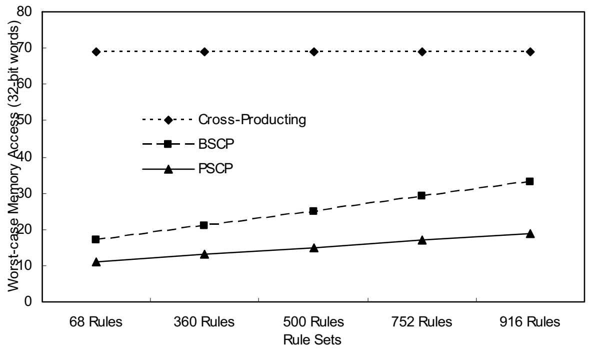


Figure 10. Worst-case Memory Access

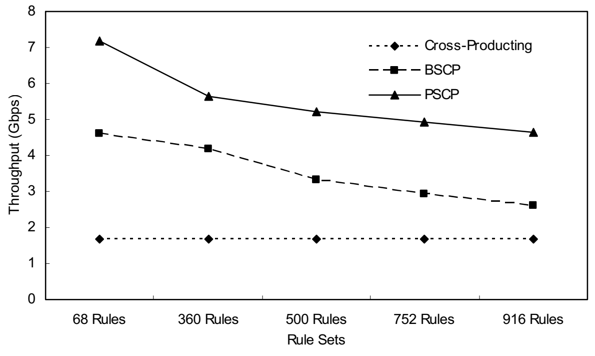


Figure 11. Throughput on IXP 2850

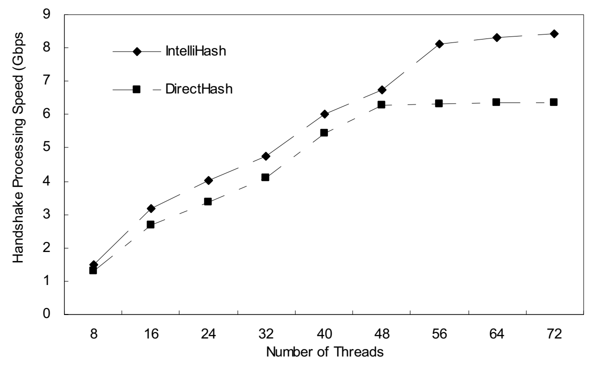


Figure 12. Handshake Processing Rate

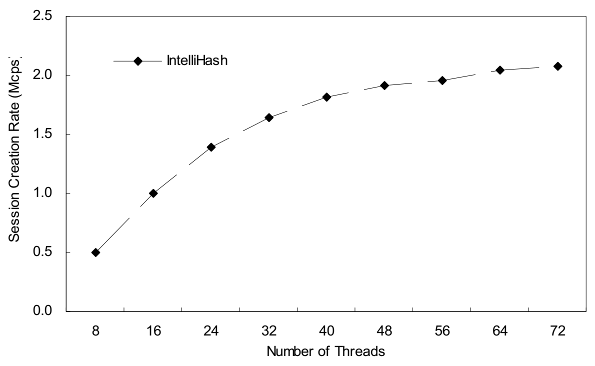


Figure 13. Session Creation Rate

4M, F is 256 bytes and L is 1/2, the size of Digested Session Table will be 112M bytes and the size of Full Session Table will be 2G bytes. These volumes of the two session tables are acceptable on IXP2850 platform.

Furthermore, with a deterministic Full Session Table size in DRAM for handling a fixed number of concurrent sessions, the size of Digested Session Table could be adjusted for adapting different network conditions.

5. Performance Evaluation

5.1. Experiment Environments

To evaluate the performance, the applications were tested in the Intel SDK 4.0 Developer Workbench, which provides a cycle-accurate simulator of the IXP2850 NP. We also test the applications on a real dual-IXP2850 platform using Smartbit600 to verify the compatibility on hardware.

5.2. Performance of PSCP

5.2.1. Worst-case Memory Access

Worst-case memory accesses, as the most important performance metrics of flow classification algorithms, provide an evaluation of the worst-case processing speed. Figure 10 shows the worse-case memory accesses of the original Cross-Producting (Cross-Producting), Binary Search Cross-Producting (BSCP) and Parallel Search Cross-Producting (PSCP) with five real-life rule sets, from which we can see that the worse-case memory accesses of PSCP is nearly 1/2 of that of BSCP and nearly 1/4 of that of Cross-Producting algorithm. This is because PSCP uses quad-search tire in the single-field search, while BSCP uses binary-search tire and the original Cross-Producting uses longest-prefix-matching respectively.

5.2.2. Throughput on Network Processor

To evaluate the worst-case throughput on IXP2850, minimum 64Byte Ethernet packets are engaged here as the input traffic and all the packets are designed to match the longest prefix and reside in the leaf nodes of the tries. Figure 11 gives the worse-case throughput achieved by PSCP, BSCP and the original Cross-Producting. It is shown that PSCP reaches a throughput of 5-7Gbps with different sizes of rule sets, whereas BSCP obtains a throughput of 2.5-4.5Gbps and Cross-Producting only reaches 1.7G throughput. Besides, the worse-case performance of PSCP and BSCP decreases slowly as the number of rules increases, because their single-field search is on the magnitude of log(N).

5.2. Performance of IntelliHash

To evaluate the performance of IntelliHash, we do experiments to compare its TCP three-way handshake processing speed with the DirectHash scheme. From Figure 12, we can see that the DirectHash scheme reaches 6.5 Gbps handshake processing speed with 72 threads, while the IntelliHash scheme reaches 8.5 Gbps speed. This is because the handshake processing of IntelliHash is implemented on SRAM while that of DirectHash can only be implemented on DRAM.

Figure 13 illustrates the entire fast path session creation performance of IntelliHash scheme integrating the PSCP algorithm proposed in this paper, where packet classification is applied on the SYN packets. From the figure, we can see that a session creation rate of up to 2 Mcps (connections per second) could be achieved with 72 Micro-engine threads and the performance increases slowly with the number of threads growing.

6. Conclusion

In this paper, we presented a fast path session creation mechanism based on multi-core and multi-threaded network processors. To achieve it, a hardware optimized flow classification algorithm based on Cross-Producting is proposed. Besides, an efficient fast path session handling scheme using separate session tables for handshake and subsequent data packets is proposed to get high session creation rate along with high session update rate.

Experimental results show that the PSCP algorithm optimized for network processor reaches a throughput of 5-7Gbps with real-life rule sets and the IntelliHash scheme achieves a session creation rate of 2M new sessions per second.

Our future work includes implementing TCP stream reassembly and holistic deep inspection system on the IXP series network processors.

Acknowledgments

This work was granted by National High-Tech R&D (863) Plan of China (No. 2007AA01Z468). The authors also would like to acknowledge the colleagues in Network Security Lab for their suggestions.

References

[1] Intel, IXP2XXX Product Line of Network Processor, http://www.intel.com/design/network/products/npfamily/ixp2xxx.htm.

[2] AMCC, Network Processor, https://www.amcc.com/MyAMCC/jsp/public/browse/controller.jsp?networkLevel=COMM&superFamily=NETP.

[3] Freescale, C-Port Network Processors, http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=02VS01DFTQ3126.

[4] Agere, Network Processor, http://www.agere.com/telecom/network_processors.html.

[5] Cavium, http://www.cavium.com/.

[6] RMI, http://www.razamicroelectronics.com/.

[7] U. R. Naik and P. R. Chandra, "Designing High-performance Networking Applications", Intel Press, 2004.

[8] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, "Fast and Scalable Layer Four Switching", Proc. ACM SIGCOMM, 1998.

[9] Intel Corporation, "Intel IXP2850 Network Processor Hardware Reference Manual", 2004.

[10] Intel Corporation, "Intel IXDP2850 Advanced Development Platform System User's Manual", 2004.

[11] B. Carlson, "Intel Internet Exchange Architecture and Applications", Intel Press, 2003.

[12] E. J. Johnson and A. R. Kunze, "IXP2400/2850 Programming", Intel Press, 2003.

[13] M. Venkatachalam, P. Chandra and R. Yavatkar, "A Highly Flexible, Distributed Multiprocessor Architecture for Network Processing", Computer Networks, 2003.

[14] M. H. Overmars and A. F. van der Stappen, "Range Searching and Point Location among Fat Objects", Journal of Algorithms, 21(3), 1996.

[15] P. Gupta and N. McKeown, "Packet Classification Using Hierarchical Intelligent Cuttings", Proc. Hot Interconnects, 1999.

[16] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields", Proc. ACM SIGCOMM, 1999.

[17] B. Xu, D. Jiang and J. Li, "HSM: A Fast Packet Classification Algorithm", Proc. of the 19th International Conference on Advanced Information Networking and Applications (AINA), 2005.

[18] S. Singh, F. Baboescu, G. Varghese and J. Wang, "Packet Classification Using Multidimensional Cutting", Proc. of ACM SIGCOMM, 2003.

[19] M. E. Kounavis, A. Kumar, H. Vin, R. Yavatkar and A. T. Campbell, "Directions in Packet Classification for Network Processors", Proc. of the 2nd Workshop on Network Processors (NP2), 2003.

[20] D. E. Taylor, "Survey & Taxonomy of Packet Classification Techniques", Technical Report, Washington University in Saint-Louis, USA, 2004.

[21] Y. X. Qi, B. Xu, F. He, X. Zhou, J. M. Yu, and Jun Li, "Towards Optimized Packet Classification Algorithms for Multi-Core Network Processors", Proc. of the 2007 International Conference on Parallel Processing (ICPP), 2007.

[22] P. Gupta and N. McKewon, "Algorithms for Packet Classification", IEEE Network, March/April, 2001.

[23] Y. Qi, B. Xu, F. He, B. H. Yang. J. M. Yu and J. Li, "Towards High-performance Flow-level Packet Processing on Multi-core Network Processors", Proc. of ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2007.

[24] Intel Corporation, "Intel IXP2400 and IXP2800 Network Processor Programmer's Reference Manual", 2004.