International Journal of Electronic Commerce Studies
Vol.6, No.2, pp.259-268, 2015
doi: 10.7903/ijecs.1346

COMPARISON OF HASH STRATEGIES FOR FLOW-BASED LOAD BALANCING

Surasak Sanguanpong
Kasetsart University
Faculty of Engineering, 50 Ngamwongwan Rd., Chatuchak, Bangkok 10900, Thailand
Surasak.S@ku.ac.th

Witsarut Pittayapitak
Kasetsart University
Faculty of Engineering, 50 Ngamwongwan Rd., Chatuchak, Bangkok 10900, Thailand
g5614500031@ku.ac.th

Kasom Koht-Arsa
Kasetsart University
Faculty of Engineering, 50 Ngamwongwan Rd., Chatuchak, Bangkok 10900, Thailand
Kasom.K@ku.ac.th

ABSTRACT

Hashing is powerful tool and widely used for flow-based load balancing schemes in parallel processing. In this paper, we analyze and compare computing overhead and load dispersion characteristics of hash strategies using XOR and CRC operations under four hashing key schemes (from 2-tuple to 5-tuple). We conduct experiments with real-life 24-hour campus network traffic. The results show that XOR32 has the lowest computing overhead among all hash function groups. Moreover, XOR32 with 4-tuple and XOR32 with 5-tuple are the two outstanding strategies that provide very good uniform distribution of traffic across multiple links, thus achieving better load balancing for flow-based applications.

Keywords: Hashing, Hash Functions, Flow-based, Load Balancing

1. INTRODUCTION

In today's context, network speed has been continuously increasing to a high gigabit per second rate, while processor and memory speeds have not

advanced at the same pace. A single physical network node, such as Intrusion Detection System (IDS), firewall, etc., is inadequate for handling high-speed traffic in this circumstance. A widely used solution is to split traffic from a high-speed input link into multiple lower speed output links. Such traffic should be evenly distributed among the parallel output links to achieve load balancing. Otherwise, possible hot-link overload and inefficient link capacity may occur. Moreover, some applications, such as intrusion detection[1] and firewall[2, 3, 4] require that a TCP flow must be maintained on a same node from the beginning until the end of flow duration.

Hashing is a simple but powerful tool for flow-based load balancing. Related packets belonging to a flow are grouped by applying a combination of header fields as a hashing key. A value computed from the hash function indicates a flow ID to direct traffic to a designated node. In this paper, we analyze characteristics of load distribution using a set of XOR and CRC functions with different keys. Three load distribution characteristics have been studied: (1) distributions by number of packets; (2) distributions by number of byte counts; and (3) distributions by number of flows.

The rest of the paper is organized as follows: Section 2 presents related works, Section 3 describes hashing methodology, Section 4 presents performance measurements and comparative results, and, finally, Section 5 concludes the paper.

2. RELATED WORK

Hashing offers a simple stateless solution for load balancing by maintaining a flow with correct packet ordering over a specific link[5]. XOR and CRC are among well-known hash functions. Although neither of them are novel, they can provide high uniformity and low cost[6]. However, balancing load in practical cases may not always be perfect due to rapidly varying and unpredictable traffic patterns[7]. Cao et al.[8] simulated performance of several hash functions and showed that CRC16 provides the best performance tradeoff. However, the trace period was too short to adequately represent actual load characteristics under modern traffic. In contrast to Cao's works, Detal et al.[9] reported that CRC16 gave a rather poor packet distribution. Similarly, our experiments showed that CRC16 has worse performance compared to XOR16 and XOR32, however their distribution characteristics have no significant difference.

Guang et al.[10] compared the uniformity of distribution and computation speed of the IP Shift-XOR (IPSX) hash algorithm, the Bob algorithm based on XOR, and the CRC32 algorithm. The IPSX offered fastest execution time with good uniformity. Jiang et al.[11] proposed Fowler-Noll-Vo (FNV)

hash for balancing SIP server clusters. However, our evaluation indicated that FNV has poorer performance compared to XOR and CRC. In this paper, we complement prior works by analyzing six differences of the XOR and CRC hashing schemes (XOR8, XOR16, XOR32, CRC8, CRC16, and CRC32) when applied to flow based load balancing. Each hashing scheme is computed with four hashing keys from 2-tuple to 5-tuple based on a combination of packet header fields.

3. LOAD BALANCING WITH HASHING

A model of a network load balancing system has a traffic splitter attached between an incoming link and multiple outgoing links as shown in Figure 1. The splitter receives packets from a higher-speed incoming link and dispatches packets to lower-speed outgoing links attached with the processing nodes.

A hash function with keys for table lookup is used for mapping a hash value to a node ID. Let N be number of available links or nodes, a hash value h for a hash function H(X) that maps a given key X to an integer between [0, N-1] is defined as follows:

$h = H(X) \bmod N$

A flow-based hash function requires at least source and destination addresses as a 2-tuple {SA, DA} of hash keys for producing a flow identifier (flow ID). The flow ID, combined with the modulo operation, gives the final hash value as the node ID to handle traffic flow.

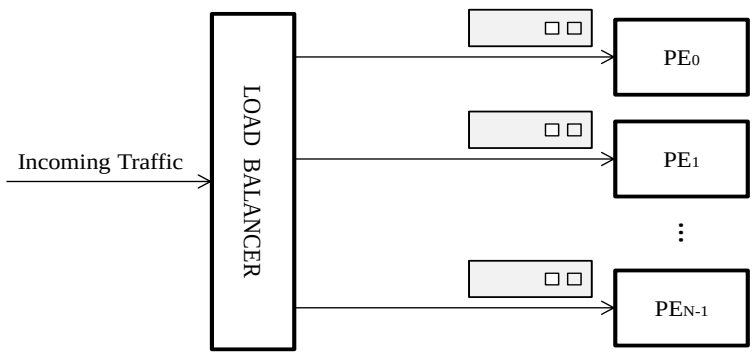


Figure 1. General model of load balancing

3.1 Hash Functions

Two hash functions are extensively used in load balancing: exclusive-OR (XOR) and Cyclic Redundancy Check (CRC). XOR is a basic Boolean operator such that if one of two bits is true, then so is the result, but if both are true, or both are false, then the result is false. The XOR function

can generate a pseudorandom number, hence it has been widely used for hashing. If K is a key with hash table size n, and $\oplus$ is the exclusive-OR operator, then the XOR operation can be expressed as:

$$H(X) = K_1 \oplus K_2 \oplus \ldots \oplus K_n$$

A main usage of CRC is for computing of a reliable checksum[12, 13], but its random output can be suitably applied as a hash function. The CRC is based on polynomial arithmetic in GF(2) (Galois field with two elements). The GF(2) polynomial has a single variable x whose coefficients are 0 or 1. We adopt 3 common CRC standards: CRC8, CRC16, and CRC32. These are shown as follows:

$$CRC8: x^8 + x^2 + x^1 + 1$$

$$CRC16: x^{16} + x^{12} + x^5 + 1$$

$$CRC32: x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

3.2 Hashing Keys

Combinations of a packet's header fields produce different hashing key schemes with *k*-tuples. Typical header fields (i.e. source address (SA), destination address (DA), source port (SP), destination port (DP), and protocol type (PT)) are used as hash keys. Table 1 illustrates four examples of hashing keys with different key sizes. Note that the 1-tuple is omitted due to its intrinsically poor distribution characteristics.

Table 1. Hashing keys schemes

Tuple	Keys	IPv4 Keys Size (bytes:bits)	IPv6 Keys Size (bytes:bits)
2-tuple	{SA, DA}	8:64	32:256
3-tuple	{SA, DA, SP}	10:80	34/272
4-tuple	{SA, DA, SP, DP}	12:96	36/288
5-tuple	{SA, DA, SP, DP, PT}	13:104	37/296

4. MEASUREMENT RESULTS

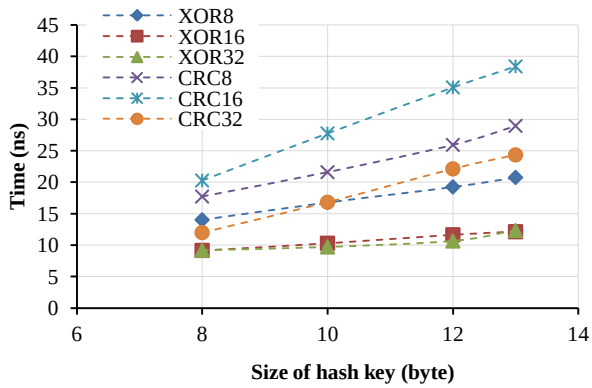
A performance of hash functions is measured in two aspects: (1) Computing overhead time and (2) Load distribution using the Coefficient of Variation[14] (CV). The CV is a statistical value representing a ratio of the standard deviation to the mean. The lower the CV, the lesser the variance (better balancing) is. We run the tests using a snapshot of the Kasetsart

University traffic repository, collected in June 2013. The 24-hour period of traffic contains about 1.53×10^{10} packets with total size of 14.25 TB. The hashing is computed on a machine with a quad-core 2.8 GHz Xeon 5560 processor, 28GB of memory, and running 64-bit CentOS Linux 6.4.

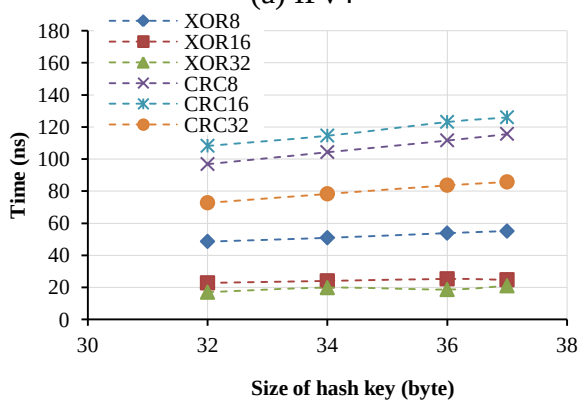
4.1 Computing Overhead

Since IPv4 and IPv6 have different sizes, which directly affect hashing performance, we then measure the computational overhead of the hash function for both IPv4 and IPv6, respectively. Since the size of the IPv6 address is four times larger than the size of IPv4, the time consumed to calculate IPv6 hashing represents the worst-case overhead.

Overhead measurements for IPv4 and IPv6 exhibit very common characteristics as shown in Figures 2(a) and 2(b), respectively. XOR32 has the lowest computing overhead for all tuple schemes, while CRC16 shows the worst overhead, followed by CRC8. For each class of hash functions, XOR32 and CRC32 are the best candidates in their class because they take the least amount of time to hash packets.



(a) IPv4



(b) IPv6

Figure 2. Computing overhead of hashing for (a) IPv4 and (b) IPv6

4.2 Distribution

We select XOR32 and CRC32 as the representative hash functions due to their minimal computational overhead. We compare load dispersion of four hashing key schemes in three distribution scenarios using CV against a number of nodes, ranging from 2 to 256. The 256-buckets size is selected in order to exploit the fast memory access of the CPU's L1 cache. The following are observations from the experimental results:

Packet Distribution: Figures 3(a) and 3(b) show that 3, 4, and 5-tuples deliver very good packet distribution in the same class. The 2-tuple distinctly performs the worst.

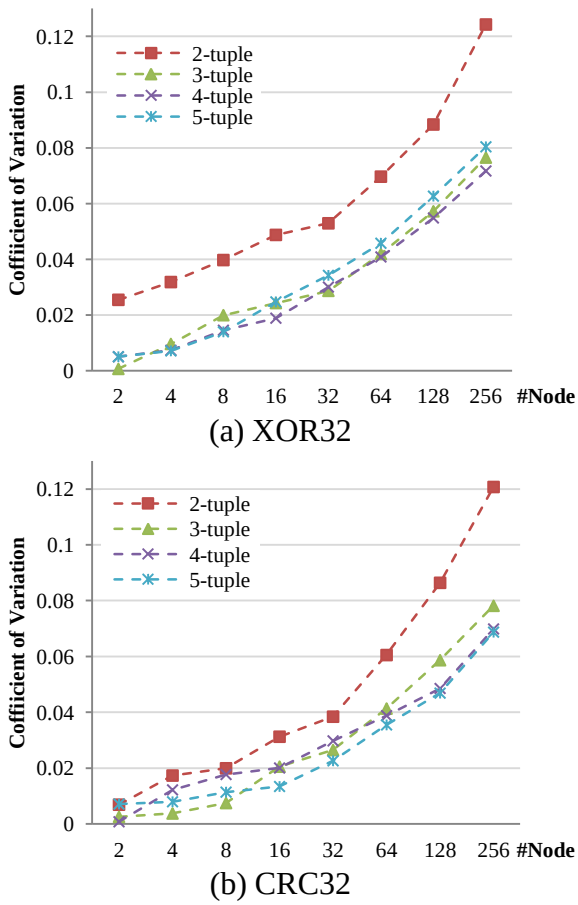
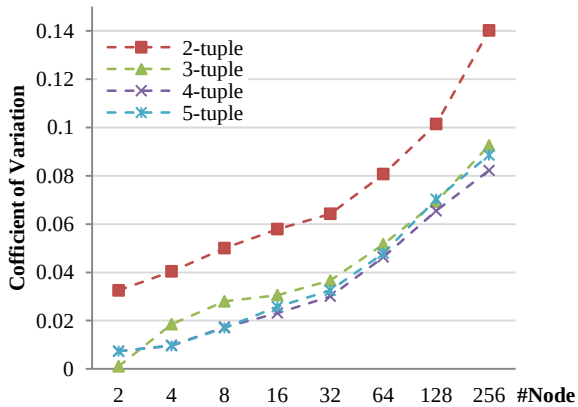


Figure 3. Distribution by number of packets

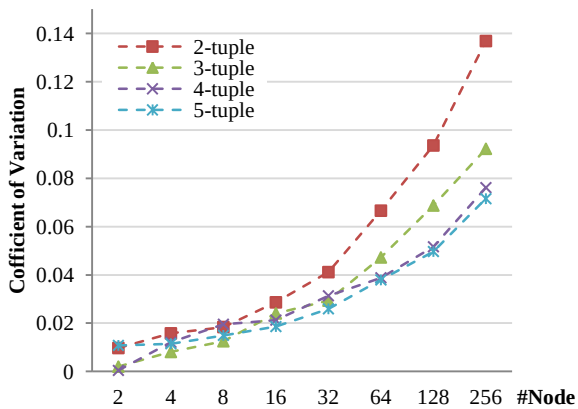
Size Distribution: XOR32 and CRC32 show similar results for the distributions by byte counts as illustrated in Figures 4(a) and 4(b). The hashing key schemes with 4 and 5-tuples show smooth distribution in the

same class, while the 2-tuple again shows poorer distribution for the large number of nodes.

Flow Distribution: XOR32 and CRC32 show similar results for the flow distributions as illustrated in Figures 5(a) and 5(b). The hashing key schemes with 3, 4 and 5-tuples have very identical distribution characteristics and show excellent flow distribution. The 2-tuple performs extremely poorly for almost every number of nodes, since load distribution is based only on the IP addresses. We conclude that good distribution can be achieved for *k*-tuple, when $k > 3$



(a) XOR32



(b) CRC32

Figure 4. Distribution by total size in bytes count

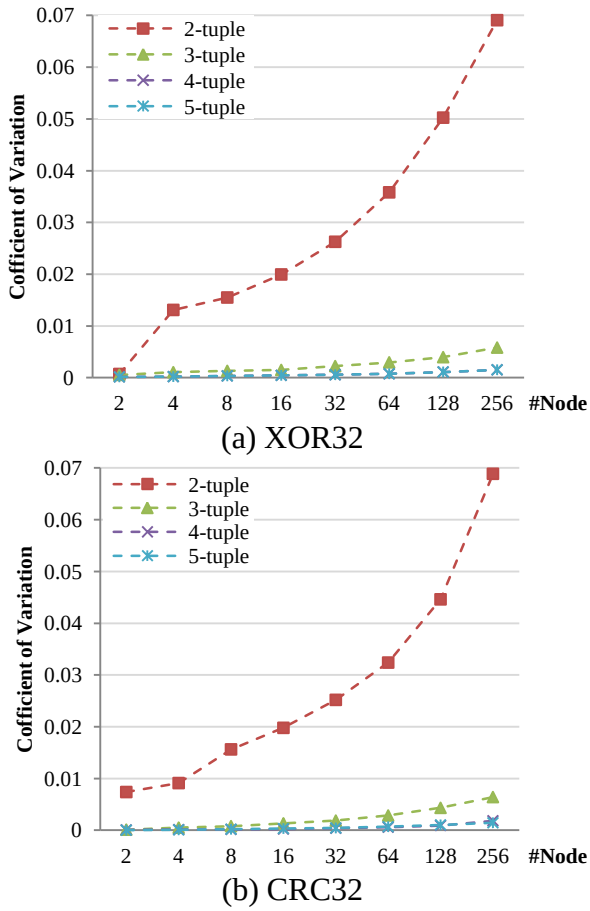


Figure 5. Figure 5. Distribution by number of flows

5. CONCLUSION

We present experimental results of computational overhead for XOR/CRC hash functions and their load distribution characteristics under four hashing key schemes. Our experiments show that XOR32 has the lowest computational overhead, followed by XOR16. We found that for every hash function, the 4-tuple and the 5-tuple show the best load distribution, respectively. The combination of XOR32 with 4-tuple key is the best candidate strategy to provide excellent random hash indices with minimal computational overhead

6. REFERENCES

[1] K. Nam-Uk, S. Jung, and T. Chung, An efficient hash-based load balancing scheme to support parallel NIDS. *Lecture Notes in Computer Science*, 6782, p537-549, 2011.

http://dx.doi.org/10.1007/978-3-642-21928-3_39.

[2] K. Koht-arsa, and S. Sanguanpong, A centralized state repository approach to highly scalable and high-availability parallel firewall. *Journal of Computers* 8(7), p1664-1676, 2013. http://dx.doi.org/10.4304/jcp.8.7.1664-1676.

[3] K. Koht-arsa, and S. Sanguanpong, High availability and scalable parallel stateful firewall design. *Presented at the International Conference on Internet Studies*, Bangkok, August 17-19, 2012.

[4] P. N. Ayuso, R. M. Gasca, and L. Lefevre, Demystifying cluster-based fault-tolerant firewalls. *Internet Computing*, 13(6), p31-38, 2009. http://dx.doi.org/10.1109/MIC.2009.128.

[5] S. Prabhavat, H. Nishiyama, N. Ansari, and N. Kato, On load distribution over multipath networks. *IEEE Communications Surveys and Tutorials*, 14(3), p662-680, 2011. http://dx.doi.org/10.1109/SURV.2011.082511.00013.

[6] B. Xiong, K. Yang, F. Li1, X. Chen, J. Zhang, Q. Tang, and Y. Luo, The impact of bitwise operators on hash uniformity in network packet processing. *International Journal of Communication Systems*, 27(11), p3158-3184, 2014. http://dx.doi.org/10.1002/dac.2532.

[7] M. Molina, S. Niccolini, and N.G. Duffield, A comparative experimental study of hash functions applied to packet sampling. *Presented at the 19th International Teletraffic Congress*, Beijing, August 29-September 2, 2005.

[8] Z. Cao, Z. Wang, and E. Zegura, Performance of hashing-based schemes for Internet load balancing. In F. Bauer (Ed.), *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (vol.1)* (p332-341). Tel Aviv: IEEE Press, 2000. http://dx.doi.org/10.1109/INFCOM.2000.832203.

[9] G. Detal, C. Paasch, S. Linden, P. Mal'rindol, G. Avoine, and O. Bonaventure, Revisiting flow-based load balancing: Stateless path selection in data center networks. *Computer Networks*, 57(5), p1204-1216, 2013. http://dx.doi.org/10.1016/j.comnet.2012.12.011.

[10] C. Guang, Z. Wei, and G. Jian, XOR hashing algorithms to measured flows at the high-speed link. In B. Werne (Ed.), Proceedings of the International Conference on Future Generation Communication and Networking (vol. 1) (p152-155). Hainan Island: IEEE Press, 2008. http://dx.doi.org/10.1109/FGCN.2008.110.

[11] H. Jiang, H. Jiang, A. Iyengar, E. Nahum, W. Segmuller, A. Tantawi, and C. Wright, Design, implementation, and performance of a load balancer for SIP server clusters. *IEEE Transactions on Networking*, 20(4), p1190-1202, 2012. http://dx.doi.org/10.1109/TNET.2012.2183612.

[12] A. Doering, and M. Waldvogel, Fast and flexible CRC calculation.

Electronics Letters, 40(1), p10-11, 2004. http://dx.doi.org/10.1049/el:20040032.

[13] M.E. Kounavis, and F.L. Berry, A systematic approach to building high performance software-based CRC generators. In Danielle C. Martin (Ed.), *Proceedings of the 10th IEEE Symposium on Computers and Communications* (p855-862). Washington, DC, USA: IEEE Press, 2005. http://dx.doi.org/ 10.1109/ISCC.2005.18.

[14] R. Feldman, and C. Valdez-Flores, *Applied probability and stochastic processes*. New York: Thomson Publishing, 2010.