Supplementary Material to Hashing Linearity Enables Relative Path Control in Data Centers [5]

Zhehui Zhang[1], Haiyang Zheng[2], Jiayao Hu[2], Xiangning Yu[2], Chenchen Qi[2], Xuemei Shi[2], Guohui Wang[2]
[1]*University of California, Los Angeles,* [2]*Alibaba Group*

A Proof of XOR/CRC hashing linearity

We first prove XOR hashing satisfies the hashing linearity. We define $src\_ip(h)$ as the function to extract source IP address field from the header, so as $dest\_ip(h)$, $src\_port(h)$, $dest\_port(h)$. XOR hash function with tuple of IP addresses and port numbers as input is $Hash(h) = src\_ip(h) \oplus dest\_ip(h) \oplus src\_port(h) \oplus dest\_port(h)$. Without loss of generality, assume h_i and h_j are arbitrary packet headers that all fields are the same except destination port number. For function $dest\_port()$, we have $dest\_port(h_i \oplus h_j) = dest\_port(h_i) \oplus dest\_port(h_j)$. Thus we have $Hash(h_i \oplus h_j) = Hash(h_i) \oplus Hash(h_j)$, which satisfies

$$Hash(h_i) \oplus Hash(h_j) = Hash(h_i \oplus h_j) \oplus Hash(0) \tag{1}$$

since $Hash(0) = 0$. The analysis of $dest\_port$ also applies to other functions to extract header fields.

CRC hashing is calculated based on division of input by a polynomial. There are several standardized polynomials mostly adopted by the communication system. For example, CRC-CCITT, a 16-bit CRC hash function, is used as the hash algorithm in commercial switches [2]. The polynomial is one bit longer than the output, i.e. 17 bits for CRC16 while 33 bits for CRC32. With polynomial p, the CRC hashing result is calculated by a polynomial long division $Hash(h_i) = division(init \oplus h_i, p)$, where $init$ is the initial value selected by CRC. Same as the previous proof, we check $Hash(h_i \oplus h_j) = division(init \oplus h_i \oplus h_j, p)$. We have $division(init \oplus h_i \oplus h_j, p) = division(init \oplus h_i, p) \oplus division(init \oplus h_j, p) \oplus division(init, p)$ since division by a polynomial is linear [4]. Then we have $Hash(h_i \oplus h_j) = Hash(h_i) \oplus Hash(h_j) \oplus Hash(0)$, where $Hash(0)$ is the hash result for an empty packet header with all 0's.

B Proof of ECMP linearity

In $Pre\_proc()$, bitwise operations like AND, OR, XOR and shift are commonly applied. Suppose we have AND operation with a constant cst for pre-processing, we define $Hash'(h) = Hash(h\&cst)$, where cst is a pre-installed constant and $Hash()$ is linear. Since

$$\begin{aligned} Hash'(h_i) \oplus Hash'(h_j) &= Hash(h_i\&cst) \oplus Hash(h_j\&cst) \\ &= Hash((h_i \oplus h_j)\&cst) \oplus Hash(0\&cst) \\ &= Hash'(h_i \oplus h_j) \oplus Hash'(0) \end{aligned}$$

, Equation 1 still holds for $Hash'()$. For the same procedure, OR also holds. For bit shifting $shift(h,k)$, where k is the index of shifted position, we have $shift((h_i \oplus h_j),k) = shift(h_i,k) \oplus shift(h_j,k)$. Equation 1 still holds for $Hash(shift(h,k))$. By the same methodology, Equation 1 also holds for masking function, which sets certain fields as zero in the result. With masking operation, hash function with pre-processing is $Hash(mask(h,s,e))$, where s and e are the starting index and the ending index of the mask.

Hashing seed is used to initialize CRC registers to vary hash results of switches at different tier to avoid traffic polarization in $Pre\_proc()$ [1]. Hash seed does not affect linearity since the XOR operation is proved to be linear. Suppose we have hash seed for switch as $Hash''(h) = Hash(h \oplus seed)$, where $Hash()$ is linear, we have

$$\begin{aligned} Hash''(h_i) \oplus Hash''(h_j) &= Hash(h_i \oplus seed) \oplus Hash(h_j \oplus seed) \\ &= Hash(h_i \oplus h_j \oplus seed) \oplus Hash(seed) \\ &= Hash''(h_i \oplus h_j) \oplus Hash''(0) \end{aligned}$$

, which proves Equation 1 still holds for $Hash''()$.

Post-processing might use XOR-folding of 32-bit hashing results to get a 16-bit result [3]. Denote the folding as $fold(r[1:32]) = r[1:16] \oplus r[17:32]$, where $[i:j]$ denotes the segment from ith bit to jth bit. Following similar logic, we can prove $Post\_proc(Hash(h)) = fold(mask(Hash(h)))$ also satisfies Equation 1.

C Discussion on hashing seeds

As we presented in the paper, the linearity guarantees

$$ECMP(h_j \oplus \Delta) = ECMP(h_i) \oplus ECMP(\Delta) \oplus ECMP(0) \tag{2}$$

, where $ECMP(\Delta)$ is decided by the hashing algorithm and $ECMP(0)$ is decided by the hashing seed (initial hashing value with an empty packet header).

Based on Equation 2, we derive that multi-hop linearity requires the selection of s_1 will not affect $ECMP_{s_1}(\Delta)$ and $ECMP_{s_1}(0)$, i.e. all switches to be select at hop 1 need to be configured with the same hashing algorithm and hashing seed. In fact, we can alter Equation 2 to relax the requirements on hashing seeds. We can derive that

$$ECMP(h_j \oplus \Delta_1 \oplus \Delta_2) = ECMP(h_i) \oplus ECMP(\Delta_1) \oplus ECMP(\Delta_2) \tag{3}$$

For two hops $s_1 \cdot s_2$, concatenation of $s_1 = ECMP_{sender}(h)$ and $s_2 = ECMP_{s_1}(h)$ still satisfies

$$\begin{aligned} &ECMP_{sender}(h \oplus \Delta_1 \oplus \Delta_2) \cdot ECMP_{s_1}(h \oplus \Delta_1 \oplus \Delta_2) \\ =&(ECMP_{sender}(h) \cdot ECMP_{s_1}(h)) \\ &\oplus (ECMP_{sender}(\Delta_1) \cdot ECMP_{s_1}(\Delta_1)) \\ &\oplus (ECMP_{sender}(\Delta_2) \cdot ECMP_{s_1}(\Delta_2)) \end{aligned} \tag{4}$$

under the assumption that the selection of s_1 will not affect $ECMP_{s_1}(\Delta)$. One might wonder how we can design the pathmap based on Equation 3. We can keep the same pathmap as presented in the paper. To decide a path offset, we get a mapping from all possible path offsets $O(\Delta_1) + O(\Delta_2)$ to $\Delta_1 \oplus \Delta_2$.

References

[1] P. Koopman and T. Chakravarty. Cyclic redundancy code (crc) polynomial selection for embedded networks. In *International Conference on Dependable Systems and Networks, 2004*, pages 145–154, 2004.

[2] Brad Matthews and Puneet Agarwal. Flow based path selection randomization, 2013. US Patent US8503456.

[3] Jarno Rajahalme. Performing a finishing operation to improve the quality of a resulting hash, 2014. US Patent US10193806B2.

[4] Mathys Walma. Pipelined cyclic redundancy check (crc) calculation. In *2007 16th International Conference on Computer Communications and Networks*, pages 365–370. IEEE, 2007.

[5] Zhehui Zhang, Haiyang Zheng, Jiayao Hu, Xiangning Yu, Chenchen Qi, Xuemei Shi, and Guohui Wang. Hashing linearity enables relative path control in data centers. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, July 2021.