Pushing the Envelope: Beyond Two Billion IP Routing Lookups per Second on Commodity CPUs

Marko Zec, Miljenko Mikuc
University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
zec@fer.hr, miljenko.mikuc@fer.hr

***Abstract*—Over the past two decades, implementing routing lookups in dedicated hardware has been accepted as an undisputable gold standard in core Internet routers due to ever increasing performance requirements and unabated global routing table growth. Several recent proposals depart from that line of thinking and suggest that software algorithms running on commodity multi-core CPUs might (again) become well suited for the task. In this article we describe a refined implementation of the DXR routing lookup scheme and subject it to a series of synthetic tests using BGP table snapshots from major Internet exchange points. Our measurements show that the algorithm scales nearly linearly on contemporary multi-core microprocessors, while the achieved peak aggregate throughput of almost 2.5 billion lookups per second presents an over a threefold increase over previously published results. Our experiments show that the aggregate throughput of a software routing lookup algorithm running on a modern commodity microprocessor can outperform a state-of-the-art ASIC chip by more than an order of magnitude, with reasonable expectations that this gap could easily double on the emerging 32- and 36- thread commodity CPUs.**

***Index Terms*—Internet, IP networks, Routing, Table lookup, Multicore processing, Performance evaluation**

I. Introduction

Internet router vendors had to work hard to start introducing 100 Gbit/s interfaces to the market, yet providing sufficiently high routing lookup throughput remains among the major challenges in designing router interface cards for even higher speeds (200, 400 Gbit/s and above). To forward minimum-sized IPv4 packets at 400 Gbit/s Ethernet line speed, a core Internet router must perform 579 million routing lookups per second (Mlps) in a database which today consists of nearly 680.000 network prefixes [1] and is continually growing. This requirement is more than four times higher than the capacity of a dedicated, state-of-the-art router application-specific integrated circuit (ASIC) from a dominant vendor, which is limited to 140 Mlps of unidirectional packet forwarding, or 280 Mlps bidirectional [2], with no (published) performance improvements since 2013.

The evolution of routing ASICs is bounded not only by technological challenges in contemporary silicon design and power dissipation management issues at peak operating conditions, but perhaps even more by the long, complex and risky development cycles, as well as prohibitively high cost of access to advanced silicon manufacturing processes, which permits only a handful of core router vendors to invest in the increasingly expensive design efforts. Consequently, smaller companies and especially academia are faced with a practically impenetrable barrier of entry to innovation in the field of high-performance router design, which negatively impacts the pace of further technological advances.

Compared to software-based routers, a significant drawback of routing ASICs is their relative inflexibility, which becomes more pronounced as network operators embrace various levels of routing function virtualization, and as the ability to quickly respond to unpredictable malicious threats and security challenges is becoming vital to real-world network operations.

A considerable interest has therefore arisen in (re)exploring the feasibility of utilizing general-purpose CPUs in the forwarding path of high-performance routers, a concept which has been all but abandoned around two decades ago, as it was deemed by far too slow for the rapid increases in transmission rates. However, compared to their counterparts from 15 years ago, contemporary multi-core general-purpose CPUs offer a completely different performance potential, which several recent proposals [3] [4] [5] [6] are leveraging to offer routing lookup throughputs which rival or exceed the performance of dedicated router ASICs. In this article, we focus on DXR [6] as one of the pioneering proposals among such routing lookup schemes, and following a few refinements in its implementation, we subject it to a series of synthetic tests on several modern commodity multi-core CPUs.

The rest of the paper is structured as follows: Section II outlines the related work in the field. Section III briefly presents DXR and discusses its runtime tradeoffs, along with the improvements we introduced while reimplementing the algorithm as a Click [7] processing module. Section IV contains a detailed performance evaluation under different operating conditions. Directions for further research and concluding remarks are provided in Section V.

II. Evolution, demise and revival of software-based routing lookup schemes

Early IP routers were all entirely software-based. Since by today's standards both line speeds and routing tables were miniscule, this worked well until mid-1990s when the Internet begun to expand at unprecedented rates. A wider adoption of faster transmission technologies, such as 155 Mbit/s ATM or 100 Mbit/s Ethernet, along with rapid increases in global routing table sizes and the introduction of Classless interdomain routing (CIDR) [8] pushed software routers to their limits

and called for rapid innovations. A comprehensive survey of software-based solutions up to the year 2001 can be found in Ruiz-Sanchez et al. [9] and Waldvogel et al. [10]. Those approaches involve tries [11], optimized by compressing long paths (Level-Compressed tries, [12]), or using n-ary branching (Multibit Tries, [13]). In its time, the Lulea scheme [14] offered promising lookup throughputs, by partitioning the trie in three levels (using 16, 8, 8 bits) and enabling the use of a compact pointers representation.

Nevertheless, none of the proposals could keep up with the exponential growth of both transmission speeds (1 and 10 Gbit/s) and the global routing table size, which by 1997 included over 40,000 prefixes. The schemes had quite large memory footprints, from 24 bytes per prefix of the Lampson-Varghese scheme [15] to 4.5 bytes per prefix of the Lulea [14], which prevented the lookup structures to fit into CPU caches as BGP table sizes continued to grow.

Both the research community and the industry shifted their focus to routing lookup methods optimized for dedicated hardware. Early implementations were constructed around ternary content-addressable memories (TCAMs) [16], but again those could not keep up with BGP table increases due to TCAM's low density and high power dissipation [17].

To cope with unabated BGP table growth, proposals to cache recent lookups in small but fast on-chip memories have surfaced occasionally (such as [18] or [19]) but never got embraced since both the vendors and operators learned that betting on traffic locality does not work well inside the Internet core due to unpredictable and constantly evolving traffic patterns.

A class of hardware-optimized approaches expands the root of the tree into a 2^k array of pointers to sub trees, such as DIR-24-8 [20] which could yield around 20 Mlps using a pipelined ASIC- or FPGA-based implementation and two external commodity DRAM chips. As more throughput could be achieved by simply throwing more parallel hardware (DRAMs) at the task, the major router vendors have been reportedly taking that route [21] to scale their ASICs into 100-300 Mlps throughput range, but cannot scale much further.

Recent proposals shift the focus back to CPUs for solving the problem of fast routing lookups. The original DXR report [6] claims compact Forwarding Information Base (FIB) encoding from 1.8 Bytes/prefix, and over 700 Mlps on an 8-core commodity CPUs. In [22] and [3] the authors propose an information-theoretic approach for FIB compression to less than a byte per prefix, and projected lookup speeds to around 18 Mlps per CPU core. Another proposal, Poptrie [4] reportedly peaks between 174 and 240 Mlps with a single core and tables with 500-800k routes, and can achieve 914 Mlps with four CPU cores. SAIL [5] claims 236 Mlps with random traffic and 625 Mlps with localities in traffic patterns.

III. DXR IMPLEMENTATION AND REFINEMENTS

DXR is a novel yet simple IPv4 routing lookup scheme which aims at leveraging key properties of modern microprocessors (large caches, short pipelines, out-of-order execution)

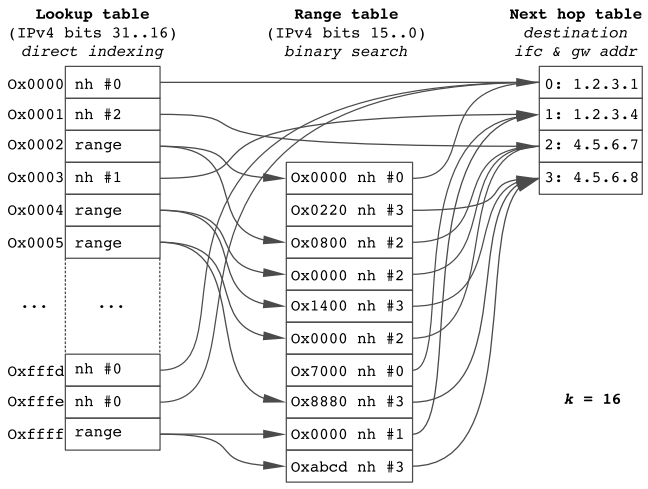

Figure 1. A simplified diagram of key DXR's data structures [6]. The lookup table has fixed size (2^K 32-bit entries), whereas the size of the range table is variable. In our re-implementation *chunks* in the range table are reference-counted objects, i.e. multiple lookup table entries can point to a single *chunk*, which further reduces an already compact memory footprint of DXR's lookup structures.

for obtaining high lookup throughputs. The key idea behind the scheme is a transformation of the traditional routing table notation as a set of {prefix, length, next hop} tuples, into a sorted array of address intervals or *ranges*, which can be iteratively searched in logarithmic time. During the process of constructing the lookup structures, adjacent ranges which point to an identical next hop are merged together, which inherently decreases both the size of the structures as well as the lookup time. To reduce the number of iterations per lookup, the entire address space is partitioned into 2^K smaller uniform blocks called *range chunks*. The initial K bits of the lookup key are used for directly indexing a *lookup table* in order to find the corresponding *chunk* inside *range table* entries. Figure 1 shows the arrangement for K = 16. The original paper [6] provides more details on data structures, in particular how the range table can be compactly encoded in DXR arrangements with $K >= 16$, and how further twofold compression can be achieved for *chunks* which reference only 8-bit next hop indices and correspond to prefix lengths up to 24 bits.

A suitable tradeoff between lookup table size and reduction in number of remaining iterative search steps can be tuned by choosing an appropriate value for K. As reducing the effective memory access latency depends on enabling data structures to reside as high as possible in the CPU cache hierarchy, in practice the most useful choices for K have been shown to be in 16 to 20 range, which corresponds to lookup structure footprints from around 1 to 5 MBytes, or 1.76 to 7.32 bytes per prefix, as shown in Table I.

The original DXR code was written in ANSI-C and targeted for execution inside the FreeBSD kernel. We recoded the algorithm in C++ and implemented it as a processing module inside the Click modular router [7]. For simplicity, we decided to retain the original BSD radix tree code [11] as a backing

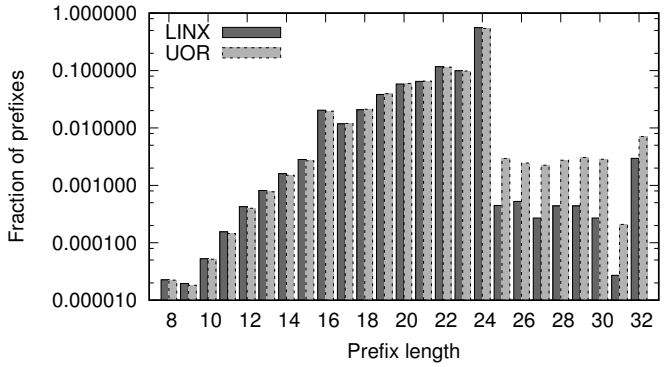

Figure 2. Distribution of prefix lengths for two of the linx.routeviews.org [23] BGP snapshots, April 2017. The University of Oregon snapshot has an unusually high proportion of prefixes with prefix lengths higher than 24, compared to other BGP snapshots.

store for the routing table, hence we encapsulated its ANSI-C implementation in an additional Click class / element. Such an approach simplified the construction of a portable synthetic testbench. Implementation inside Click also resulted in instant portability to other operating systems, such as Linux. More importantly, the new implementation allows multiple independent DXR instances to coexist inside a single Click configuration, which will permit us to conduct future experiments focused on network function virtualization implications.

As an improvement over the original implementation, our *DXR/Click* version implements *chunks* as reference counted objects, which reduces the lookup structure's memory footprint by the size of each identical chunk copy. In practice, this has been shown to yield virtually no impact with most compact ($K = 16$ or $K = 17$) lookup structure configurations, while for higher values of K the size reduction and thus lookup throughput becomes measureable, though still negligible.

Finally, we also reimplemented the *DirectIPLookup* Click element which embodies the DIR-24-8 lookup scheme [20]. This was necessary since the original DirectIPLookup implementation (dating from 2005) could not build lookup structures corresponding to contemporary databases of nearly 680.000 prefixes in reasonably short timeframes. A fully functional DirectIPLookup element permitted us not only to compare DXR against DIR-24-8 performancewise, but to check them both for correctness against the proven BSD radix tree implementation, which led to discovery of several subtle bugs in the original DXR version, which we subsequently rectified.

IV. Performance evaluation

We conducted a series of synthetic lookup throughput experiments using streams of randomly generated keys (IPv4 addresses) along with freely available IPv4 routing table snapshots from several major Internet exchange points [23]. From the available snapshots, we chose to focus on those with the largest number of adjacent next hops, as next hop diversity tends to reduce the opportunities for route / address range aggregations and therefore puts more strain on lookup operations. As shown in Table I, our test vectors include both recent (from April 2017), as well as several snapshots from 2014, as a reference for evaluating the impact of route table growth.

Prefixes with lengths of /24 dominate in all snapshots, followed by less specifics, similar to two snapshots shown in Figure 2. Since networks with prefix lengths more specific than /24 typically originate from peering links between BGP speakers in Internet exchange points, they are less often globally announced. A recent snapshot from the University of Oregon stands out from the rest by including a disproportionate amount of prefixes with prefix lengths higher than 24, which also contributes to the total number of prefixes which in that particular snapshot exceeds the average of other Internet exchange points by around 40,000, an anomaly which we did not further investigate. Instead, we decided to choose the recent LINX snapshot as the baseline for most of our experiments, since it includes the largest number of unique next hops, and as such minimizes the opportunities for route / range aggregation. Nevertheless, in an experiment where we compared the impact of routing table properties (number of prefixes and next hops) on aggregate lookup throughput shows that the performance variations with different snapshots are minimal (around 5%), as visible in Figure 3.

Figure 4 shows how the arrangement of DXR structures (parameter K) influences peak single-thread lookup throughput, driven by a stream of uniformly random (RND) keys. We ran the same experiment on five machines with different core counts, cache sizes, clock speeds and memory access latencies, as shown in Table II. The machines with bigger L3 caches benefited more from DXR configurations with higher values of K, but as soon as the size of lookup structures approached or exceeded the cache size, lookup throughputs collapsed due to excessive latencies of fetching data from the external DRAM. As there were no dependencies between successive queries, even when data has to be fetched from

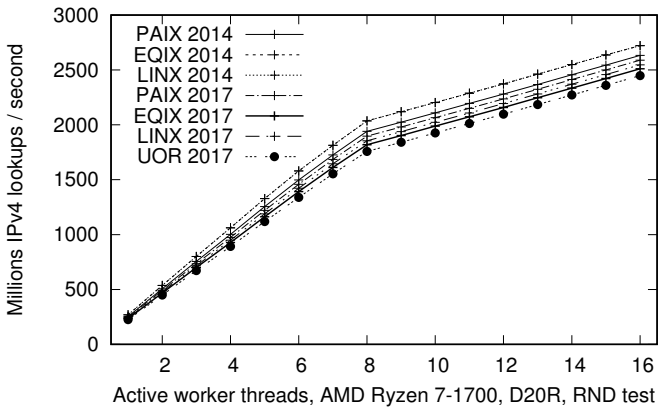

Figure 3. Aggregate lookup throughput for different BGP table snapshots as a function of the number of active worker threads subjected to streams of uniformly random lookup keys. The reduced slope in throughput increase beyond the 8th worker thread is due to a lower contribution of thread pairs scheduled on simultaneous multi-threading (SMT) virtual cores.

Table I
CHARACTERIZATION OF DXR DATA STRUCTURES FOR SEVERAL FULL-VIEW IPV4 ROUTING TABLE SNAPSHOTS.

			k = 16 (D16R scheme)					k = 20 (D20R scheme)				
Table snapshot	IPv4 prefixes	Next hops	Footprint (bytes)	Direct coverage	Range fragments short	Range fragments long	Build (ms)	Footprint (bytes)	Direct coverage	Range fragments short	Range fragments long	Build (ms)
PAIX 2014	504818	58	879964	75.6 %	269536	19687	70.1	4636992	94.0 %	210208	5568	316.0
EQIX 2014	493049	58	807280	77.6 %	236388	18090	92.0	4563208	94.7 %	177834	3309	687.2
LINX 2014	513644	239	954568	75.3 %	269850	38181	98.7	4706452	93.8 %	237728	9173	758.9
PAIX 2017	675791	85	970664	73.3 %	313376	20442	94.1	4694696	93.0 %	237316	6440	501.0
EQIX 2017	672790	159	952136	73.7 %	302256	21370	120.0	4663640	93.2 %	220446	7111	911.3
LINX 2017	663729	560	1170504	73.0 %	280318	86931	120.4	4856504	92.7 %	293006	19047	900.4
UOR 2017	713253	34	1192072	72.4 %	287726	88619	102.1	4873896	92.6 %	286844	26476	524.3

Table II
CHARACTERIZATION OF MICROPROCESSOR CACHE HIERARCHIES AND ACCESS LATENCIES.

				Level 1 Cache			Level 2 Cache			Level 3 Cache			DRAM	
		Cores /	Clock	Size	Latency		Size	Latency		Size	Latency		Latency	
Processor	Year	Threads	GHz	KB	cycles	ns	KB	cycles	ns	KB	cycles	ns	cycles	ns
Intel i5-3210M	2012	2 / 4	2.5	32	7	2.9	256	13	5.3	3072	24	9.7	214	85.8
Intel i3-4150	2014	2 / 4	3.5	32	9	2.6	256	16	4.6	3072	35	10.1	243	69.6
Intel i7-4771	2013	4 / 8	3.5	32	9	2.6	256	16	4.6	8192	38	10.9	260	74.3
Intel i7-5930K	2014	6 / 12	3.5	32	9	2.6	256	15	4.3	15360	51	14.6	263	75.2
Intel E5-2658	2013	10 / 20	2.4	32	9	3.8	256	16	6.7	25600	46	19.2	237	98.8
AMD R7-1700	2017	8 / 16	3.4	32	10	3.0	512	21	6.2	2 * 8192	43	12.7	352	103.8

cache layers far from the processor core or even DRAM, out-of-order execution mechanics could begin to resolve the next key, thus effectively interleaving several lookups.

Figure 5 shows the effects of introducing artificial dependencies between successive lookups by logically XORing each key with the result of the previous query. In such a setting the CPU's out-of-order scheduler was unable to pipeline memory reads since the address of each memory read could not be computed before the previous lookup was completely resolved. The top effective throughput was significantly lower compared to operation on independent keys (70 Mlps vs. 235 Mlps).

How the algorithm scales on multiple execution cores depending on the choice of parameter K is shown in Figure 6. The graph shows the increases in lookup throughput with additional worker threads on an AMD Ryzen 7-1700 machine, using a stream of independent, uniformly random lookup keys (RND test) as a stimulus. Similar trending was observed on other machines as well, with different choices of K yielding the best overall throughput which can be correlated to the size of CPU caches, as previously shown in Figures 4 and 5. We decided to show only the graph for the AMD machine, since it was the most modern one we had at our disposal, and since it yielded the top aggregate throughput among all tested CPUs at 2,449 Mlps (i.e. 2.45 *billion* lookups per second). Another important result that can be observed in Figure 6 is how well the DXR scheme scales compared to DIR-24-8, which has a working-set footprint of around 33 MB, and which thus does not fit the lookup structures in CPU's caches. While DXR and DIR-24-8 yielded comparable throughputs on a single CPU core, as soon as more worker threads were introduced, the throughput of DIR-24-8 saturated and even slightly collapsed due to the inability of the DRAM subsystem to service random access read patterns beyond a certain threshold.

Again, introducing artificial dependencies between subsequent queries had negative impact with multiple worker threads, as shown in Figure 7 (SEQ test). Conversely, Figure 8 illustrates how the tested algorithms could behave when subjected to traffic patterns with certain degree of locality (REP test), which is a natural property of regular (nonmalicious) transfers in the Internet. In this test we introduced a small sliding window under which random keys were repeatedly used, which permitted the lookup structures to be reused for several times after they migrated to L1 cache, before being displaced by other random keys. Both DXR and DIR-24-8 show an increase in overall throughput under such conditions, though DIR-24-8 benefits more from traffic locality, as its pressure on the DRAM subsystem gets reduced.

Since the drivers for the built-in monitoring infrastructure on AMD Ryzen CPU were not yet available, we conducted further tests on Intel machines which have better support for hardware performance counters, in order to determine the levels of pressure on DRAM subsystem by tracking last-level-cache miss counters [24]. Figure 9 shows the correlation between lookup throughput for DXR configured with $K = 19$ and DIR-24-8 on a low-end Intel CPU. Per statistics harvested from the LLC miss counter, DIR-24-8 saturated the DRAM subsystem already with two worker threads at around 120 millions L3 cache misses per second, while achieving the top lookup throughput with three worker threads. In contrast to this, DXR scaled well to all CPU cores, while achieving peak lookup throughput roughly four times higher than DIR-24-8, at a fraction of L3 cache misses. It should be noted that a certain portion of L3 misses in all cases can be attributed to unavoidable fetching of random keys from the memory, as well as storing the results (next hop indices) back to another memory array. Figure 10 shows the similar effect on L3

cache trashing with the SEQ test which introduces artificial dependencies between successive lookups.

V. Conclusion and future work

The main contribution of our paper is an empiric proof that software routing lookups can easily achieve throughputs of 2.45 billion lookups per second on modern commodity multi-core microprocessors, with a healthy margin for executing other CPU-intensive tasks, or streaming large amounts of data over I/O and DRAM buses. The peak throughputs we achieved are more than three times higher than previously published top scores for software-based schemes, and over an order of magnitude higher than the capacities of today's top-of-the-line router ASICs. More importantly, executing the synthetic tests on machines capable of supporting many (16 to 20) hardware threads, shows that near linear scaling to multiple execution cores can be sustained by careful choice of lookup structure configuration, which raises our expectations that the algorithm could continue to scale smoothly on emerging commodity CPUs which will support 32 to 36 execution threads, which could translate to throughputs in 5 GLps range in the near future.

Another important result of our experiments is a demonstration that the significant growth in global routing table size has only a moderate impact on routing lookup performance, as the increase of around 35% in the number of prefixes observed over the past three years yields variations of only around 5% in routing lookup throughput. This leads to a conclusion that the imminent future gains in global routing table size will be easily offset by sheer increases in the number of cores per die of the emerging many-core microprocessors.

We expect that the compact memory footprint of modern routing lookup structures will make it a feasible choice for various network function virtualization scenarios and applications, with acceptable performance penalties due to unavoidable increases in working set scope and thus cache trashing and more intensive DRAM traffic. We are planning to shift the focus of our further research and experimentation in that direction.

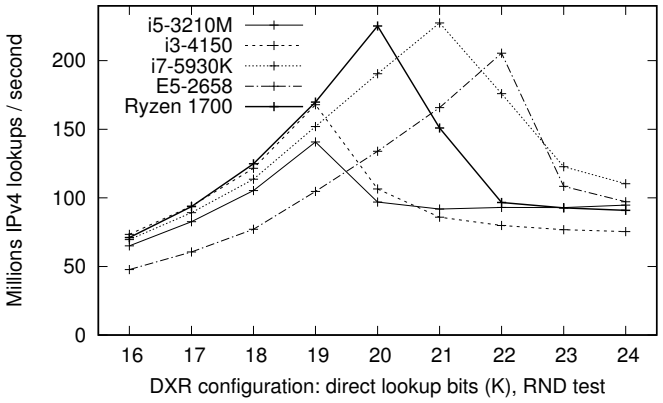

Figure 4. Single-thread performance for different DXR configurations. LINX 2017 snapshot, RND test (uniformly random keys).

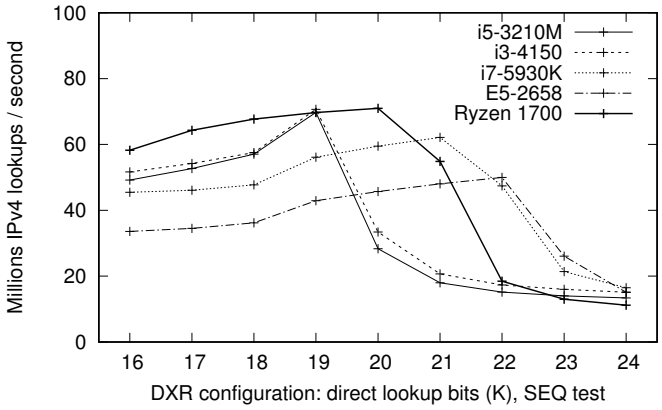

Figure 5. Single-thread performance for different DXR configurations. LINX 2017 snapshot, SEQ test.

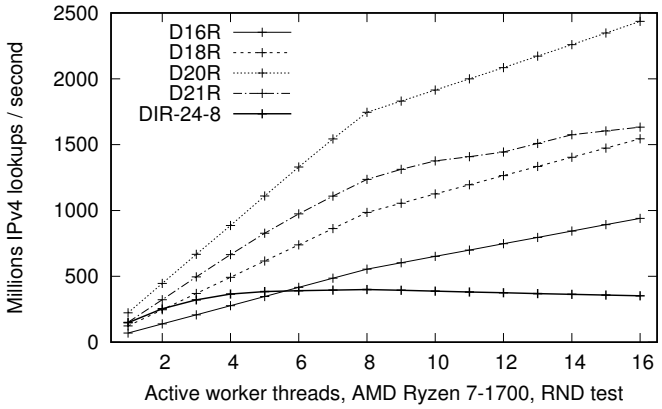

Figure 6. Aggregate lookup throughput for DIR-24-8 and four different DXR configurations as a function of the number of active worker threads, using LINX-2017 table snapshot (663729 prefixes, 560 nexthops) and streams of uniformly random lookup keys. The throughput with DXR configurations up to K=20 scales nearly linearly with additional CPU cores due to lookup structures fitting in cache hierarchies, while D21R and DIR-24-8 are limited by DRAM's random access throughput.

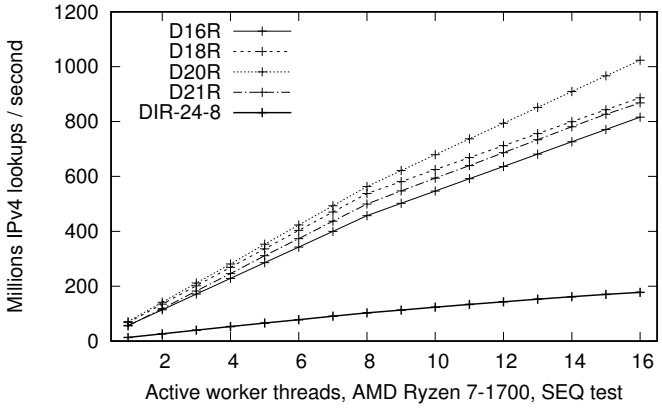

Figure 7. Aggregate lookup throughput, LINX-2017 table snapshot, uniformly random lookup keys with artificial dependencies. Working threads operate independently, but within a thread each lookup must be resolved before proceeding to the next key.

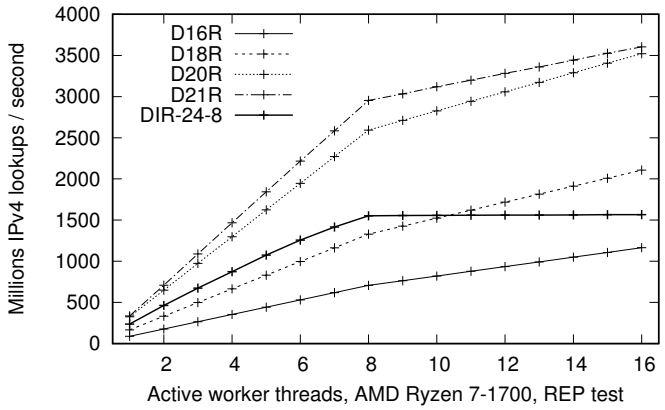

Figure 8. Aggregate lookup throughput, LINX-2017 table snapshot, uniformly random lookup keys with repeated queries. Each key within a sliding window is looked up 8 times, in an attempt to emulate locality in traffic patterns, while still preventing CPU's branch predictors to become over-trained and yield overly optimistic results.

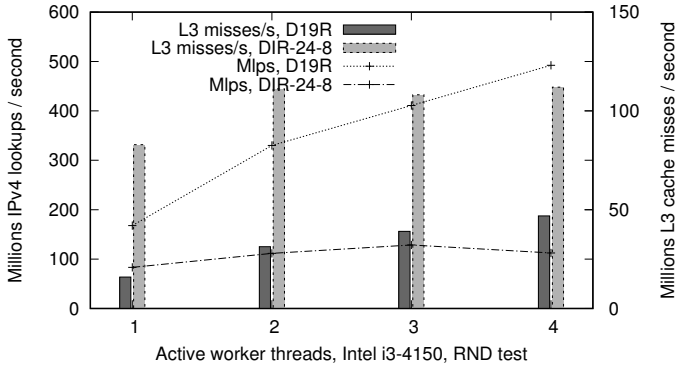

Figure 9. L3 cache misses for DXR with $K = 19$ and DIR-24-8. RND test, LINX 2017 snapshot, Intel i3-4150 CPU.

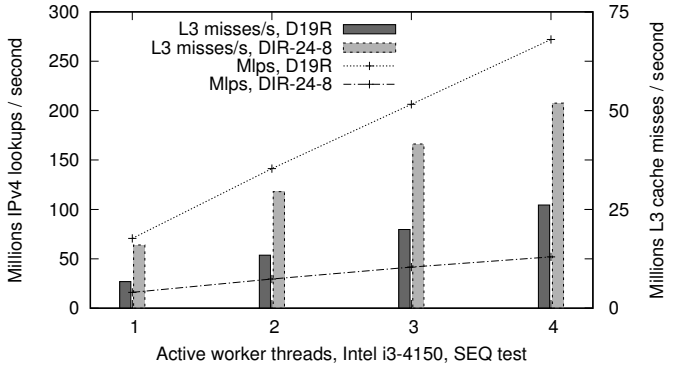

Figure 10. L3 cache misses for DXR with $K = 19$ and DIR-24-8. SEQ test (artificial dependencies between queries), LINX 2017 snapshot, Intel i3-4150 CPU.

REFERENCES

[1] G. Huston, "BGP routing table analysis reports," *http://bgp.potaroo.net/*, 2017.

[2] L. Wobker, "Evolution of core routing hardware and software," *Cisco-Live*, 2014.

[3] G. Rétvári, J. Tapolcai, A. Kőrösi, A. Majdán, and Z. Heszberger, "Compressing IP forwarding tables: towards entropy bounds and beyond," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 149–162, 2016.

[4] H. Asai and Y. Ohara, "Poptrie: A compressed trie with population count for fast and scalable software IP routing table lookup," in *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 57–70, ACM, 2015.

[5] T. Yang, G. Xie, Y. Li, Q. Fu, A. X. Liu, Q. Li, and L. Mathy, "Guarantee IP lookup performance with FIB explosion," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 39–50, 2015.

[6] M. Zec, L. Rizzo, and M. Mikuc, "DXR: towards a billion routing lookups per second in software," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 5, pp. 29–36, 2012.

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Frans Kaashoek, "The Click modular router." ACM Trans. on Computer Systems 18(3), August 2000, pages 263-297.

[8] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless inter-domain routing (cidr): an address assignment and aggregation strategy," tech. rep., 1993.

[9] M. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, vol. 15, pp. 8–23, 2001.

[10] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high-speed prefix matching," *ACM Trans. Comput. Syst.*, vol. 19, pp. 440–482, Nov. 2001.

[11] K. Sklower, "A tree-based packet routing table for Berkeley Unix," in *USENIX Winter Conference*, pp. 93–104, 1991.

[12] S. Nilsson and G. Karlsson, "IP-address lookup using LC-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1083–1092, 1999.

[13] V. Srinivasan and G. Varghese, "Faster IP lookups using controlled prefix expansion," in *SIGMETRICS '98/PERFORMANCE '98*, pp. 1–10, ACM, 1998.

[14] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," *SIGCOMM Computer Communication Review*, vol. 27, pp. 3–14, Oct. 1997.

[15] B. Lampson, V. Srinivasan, and G. Varghese, "IP lookups using multiway and multicolumn search," *IEEE/ACM Trans. on Networking*, pp. 324–334, 1998.

[16] A. J. McAuley and P. Francis, "Fast routing table lookup using CAMs," in *INFOCOM'93. Proceedings. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE*, pp. 1382–1391, IEEE, 1993.

[17] F. Zane, G. Narlikar, and A. Basu, "CoolCAMs: Power-efficient TCAMs for forwarding engines," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1, pp. 42–52, IEEE, 2003.

[18] Tzi-cker Chiueh and P. Pradhan, "High performance IP routing table lookup using CPU caching," in *INFOCOM*, pp. 1421–1428, 1999.

[19] H. Song, F. Hao, M. S. Kodialam, and T. V. Lakshman, "IPv6 lookups using distributed and load balanced bloom filters for 100 Gbps core router line cards," in *INFOCOM*, pp. 2518–2526, 2009.

[20] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *INFOCOM*, pp. 1240–1247, 1998.

[21] J. Scudder, "Router scaling trends," in *APRICOT Meeting*, 2007.

[22] G. Rétvári, J. Tapolcai, A. Kőrösi, A. Majdán, and Z. Heszberger, "Compressing IP forwarding tables: towards entropy bounds and beyond," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 111–122, ACM, 2013.

[23] "University of Oregon RouteViews project," *Eugene, OR.[Online]. Available: http://www.routeviews.org*.

[24] C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse engineering Intel last-level cache complex addressing using performance counters," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 48–65, Springer, 2015.