Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup – Public Review

Luigi Rizzo
Università di Pisa
Pisa, Italy
rizzo@iet.unipi.it

IP address lookup and longest prefix match are well studied problems for which many solutions have been proposed over the years, targeting both hardware and software implementations.

The most performant schemes for software IP lookup rely on a (logical) trie data structure, preprocessed in various ways to reduce its depth – hence, reducing the processing complexity at run time, and at the same time avoid a space explosion. The latter is essential to make the data structures fit in the cache of the processor in spite of increasingly large sets of prefixes.

The software lookup schemes presented in the recent literature exceed 100 Million lookups per second (Mpls) per core, and scale almost linearly at least on system with 4-8 cores. Poptrie is a new proposal in this area, which addresses time and space savings with a combination of ideas, some well known, some less so. In particular, the name comes from the use of a "population count" CPU instruction, which here is ingeniously used to compress nodes in a 64-ary trie. Together with another well known technique, controlled prefix expansion, poptrie manages to complete a 32-bit lookup in 3-4 steps. Thanks to the small memory footprint, in the benchmarks this yields a a lookup rate up to 20-30% faster than competing solutions.

Benchmarks of course do not tell the full story. At these speeds, and for this type of problem, performance is affected dramatically by memory performance and data dependencies. When the working set of the program grows, requiring accesses to higher level caches or even DRAM, it is common to see large performance drops. Also, it is possible to see speed inversions between different schemes as the problem size changes.

Hence, more than absolute or relative speed, it is important to understand how each algorithm is affected by the problem's size and request patterns; how individual components of an algorithm influence the overall performance; and how performance may change with longer addresses – think for example to the application to IPv6 or other namespaces.

The authors do a great job in analysing these issues in detail for Poptrie and a number of competing algorithms, looking not only at the aggregate performance of the algorithms but trying to identify the behaviour of each of their components, even at the cycle count level.

The takeout from this paper is twofold. On the one hand, one learns another solution for fast IP lookup, and possibly another trick to use for compressed, random access arrays. But even more important is the comparative performance analysis in Section 4, which gives great insight on the behaviour of the different schemes considered, and may provide useful suggestions to design high speed data structures.