Named Data Networking on a Router: Forwarding at 20Gbps and Beyond

Won So, Ashok Narayanan, David Oran and Mark Stapp
Cisco Systems, Inc. Boxborough, MA, USA
{woso, ashokn, oran, mjs}@cisco.com

ABSTRACT

Named data networking (NDN) is a new networking paradigm using named data instead of named hosts for communication. Implementation of scalable NDN packet forwarding remains a challenge because NDN requires fast variable-length hierarchical name-based lookup, per-packet data plane state update, and large-scale forwarding tables.

We have designed and implemented an NDN data plane with a software forwarding engine on an Intel Xeon-based line card in a Cisco ASR9000 router. In order to achieve high-speed forwarding, our design features (1) name lookup via hash tables with fast collision-resistant hash computation, (2) an efficient and secure FIB lookup algorithm that provides good average and bounded worst-case FIB lookup time, (3) PIT partitioning that enables linear multi-core speedup, and (4) an optimized data structure and software prefetching to maximize data cache utilization.

In this demonstration, we showcase our NDN router implementation on the ASR9000 and demonstrate that it can forward real NDN traffic at 20Gbps or higher.

Categories and Subject Descriptors

C.2.6 [**Internetworking**]: Routers

Keywords

Named data networking, router, packet forwarding engine

1. INTRODUCTION

Named data networking (NDN) is a networking paradigm that tries to address issues with the current Internet by using *named data* instead of named hosts for the communication model. NDN communication is consumer-driven; a client requests a named content via an *Interest* packet, then any node receiving the interest and having the content satisfies it by responding with a *Data* packet [2].

Implementation of scalable NDN packet forwarding remains a big challenge because NDN requires (1) fast lookups with variable-length hierarchical tokenized names, (2) per-packet data plane state update on PIT (Pending Interest Table), and (3) large-scale forwarding tables – approximately $O(10^8)$ for FIB (Forwarding Information Base) and $O(10^7)$ for PIT and CS (Content Store) based on our assessment of the likely traffic matrix seen by an NDN router.

We have designed an NDN data plane and implemented it entirely in software on an Intel Xeon-based line card in a Cisco ASR9000 router. To the best of our knowledge, this is the first implementation of an NDN data plane on a real router with high-speed forwarding.

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.
ACM 978-1-4503-2056-6/13/08.

2. DESIGN & SYSTEM OVERVIEW

To implement our NDN data plane, we have selected the Cisco ASR9000 router with the Integrated Service Module (ISM). This is a flexible platform because forwarding on an ISM is purely done by the software running on 64-bit Linux. The ISM is currently used for video streaming and NAT applications; it features 2 Intel 2.0GHz hexa-core Westmere EP (Xeon) Processors with Hyper-threading and 4 Intel Niantic 10GE interfaces. Additionally, it can include up to 3.2 TB of SSD with 2 modular flash SAM (Service Accelerator Module) that can be used as NDN cache storage.

An efficient name lookup engine is a critical component of fast NDN packet forwarding. Since NDN lookup is based on variable-length names, it can be seen as a string match problem. After investigating 2 approaches – DFA (Deterministic finite automata) vs. hash table (HT), we chose HT because it is simpler and we believe better matched the nature of the NDN lookup problem. A basic HT-based lookup algorithm for NDN Interest works as follows: (1) the full-name hash is generated from the Interest name and matched against the CS, PIT then FIB HT and (2) after decoding the name components, successive lookups are performed with shorter name prefix hashes against the FIB, which stores all name prefixes in a large HT. Data packet forwarding only needs CS and PIT lookups with the full-name hash.

The choice of a hash function significantly affects the performance of HT-based lookup. Non-crypto hash functions like CityHash show noticeably better performance than cryptographic ones such as MD5 [3]. However, a security risk is involved in using a non-crypto hash because a lack of collision resistance makes a router vulnerable to hash flooding attacks. This is particularly dangerous for an NDN router because a PIT entry is inserted based on the name of an Interest packet; if a malicious user generates Interest packets with a set of names that cause hash collisions, a router will suffer significant lookup performance degradation. The recently proposed *SipHash* [1] offers a good balance as it provides collision resistance and comparable performance to non-crypto hashes. We have implemented our own SipHash that computes all prefix hashes with one pass while simultaneously parsing the name components.

Since a single data cache (D$) miss introduces more than 100 stall cycles in an Intel-based platform, we have designed a D$-

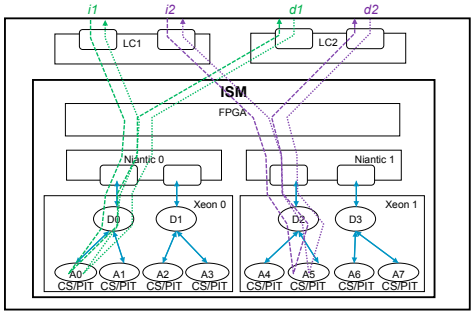

Figure 1: NDN packet flow in ASR9000 with ISM

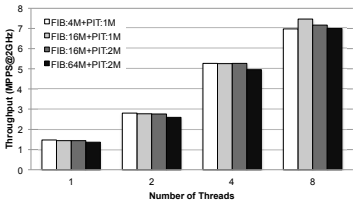

Figure 2: Performance of NDN forwarding engine on ISM

friendly HT data structure using *compact array*, where each hash bucket contains 7 pre-allocated entries in 1 cache line (64 bytes). Compared with linked-list bucket design, it shows better performance with a smaller memory footprint as long as the HT load factor (#entries/#buckets) stays no greater than 4. Additionally, we apply software prefetching so that buckets used for the subsequent lookups are prefetched earlier to minimize the D$ miss impact.

In a typical Interest forwarding scenario, the FIB HT lookup can be a bottleneck because lookup is iterated until it finds the longest prefix match (LPM). A basic LPM algorithm that starts from the full name (and continues to shorter prefixes) is not desirable because (1) FIB matches tend to happen at prefixes considerably shorter than the full name, and (2) non-matching names consume the most lookup time; it makes a router vulnerable to an attack against FIB processing wherein an adversary injects Interest packets with long non-matching names with garbage tokens.

To address this, we propose a *2-stage LPM algorithm* using *virtual* FIB entries. With this scheme, lookup first starts from a certain short (most-likely-match, M) prefix and continue to shorter prefixes or restarts from a longer (but shorter than the longest, MD) prefix. Every FIB entry with M prefixes maintains the maximum depth (MD) of prefixes that start with the same prefix in the FIB; a *virtual* entry is created if needed. At 1st stage, lookup starts from M and continue to shorter prefixes. If no match is found at 1st stage, there is no LPM. If a match is found and $MD > M$, lookup starts 2nd stage from MD and continues until it finds a match. This algorithm not only reduces the average FIB lookup time by finding a matching prefix earlier, but also provides the bounded worst-case FIB lookup time (i.e. M lookups for garbage-token names) and hence improves the NDN router security. Putting this all together, our Interest forwarding code is optimized so that it performs name parsing, SipHash computation, and 1st stage of FIB lookup simultaneously via software prefetching.

For PIT and CS, 2 more techniques are devised. First, PIT and CS lookups are combined into one HT lookup because they are adjacent and use the same full-name key. A bit flag is used to distinguish whether an entry belong to CS or PIT. Besides saving one HT lookup, it is quite effective for Data forwarding because it reduces the frequency of insertion and deletion on CS and PIT dramatically.

Second, we maintain a separate partitioned PIT (i.e. CS/PIT) for each core (or thread). Our experiments show that sharing PIT entries among threads significantly impedes multi-core parallelization speedup because (1) a read or write lock on the shared entry can block threads due to access serialization, and (2) a memory write on the shared entry causes invalidation of copies in other per-core caches due to a coherence mechanism (e.g. MESI) and therefore lowers D$ utilization dramatically. PIT partitioning enables each core to exclusively access its own PIT without any locking or data aliasing among cores. For this to work, Interest and Data packets with identical names must be looked up in the same PIT instance. This is achieved by distributing packets to cores based on names. In our implementation, we use full-name hashes to distribute packets to the forwarding threads.

Figure 1 shows how NDN packets are processed in ASR9000 with an ISM. The router is connected externally to the interfaces on regular 10GE transport line cards (LC1-2). A high-speed switching fabric carries packets from a transport LC to the ISM. When an NDN packet arrives at a fabric interface on the ISM, it is received by a *Dispatcher* process (D0-3) attached to that interface. For performance, we bypass the Linux networking stack by using a user-space driver which directly DMAs packets between the *Dispatcher* and the NIC. The *Dispatcher* identifies the offset of the name in the packet and computes the hash of the full name, which is used to identify which *App* process (A0-A7) will process the packet (in order to support the partitioned PIT). The full-name hash is passed in with the packet buffer so that the *App* should not recompute it. The *App* performs all the NDN forwarding operations – CS/PIT lookup, FIB lookup – and then returns the packet to the *Dispatcher* along with output interface information. Lastly, the *Dispatcher* forwards the returned packet out to the outgoing interface through the switching fabric. By varying the number of *App* processes, our architecture can correctly balance the work performed at each step for maximum utilization.

3. PERFORMANCE & DEMONSTRATION

Figure 2 shows the performance of our forwarding engine measured on the ISM with an NDN forwarding simulator; it runs the identical forwarding code but reads packets from the files instead of the NICs. We have simulated 13 million Interest packets generated from 16 IRCache URL traces varying FIB and PIT sizes from 1M to 64M entries. It achieves near-linear speedup when increasing the number of threads running on different CPU cores. The throughput is only minimally affected by different sizes of FIB and PIT. By using 8 *App* processes as in Figure 1, our forwarding engine can process NDN Interest packets at 7.15 MPPS. Assuming the average NDN packet size is 500 bytes with small Interest but large Data packets, the throughput of our NDN forwarding is well above 4.81 MPPS required for 20Gbps NDN traffic.

In the demonstration, we showcase our NDN router implementation on the ASR9000 and demonstrate that it can forward real NDN traffic at 20Gbps or higher with an artificial traffic generator that produces various mixes of Interest and Data packets derived from traces.

4. REFERENCES

[1] J.-P. Aumasson and D. J. Bernstein. Siphash: a fast short-input PRF. Cryptology ePrint Archive, Report 2012/351, 2012. `http://eprint.iacr.org/`.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *CoNEXT '09*, 2009.

[3] W. So, A. Narayanan, D. Oran, and Y. Wang. Toward fast NDN software forwarding lookup engine based on hash tables. In *ANCS '12*. ACM, 2012.