A Distributed Operating System Network Stack and Device Driver for Multicores

Saif Ansary[†], Antonio Barbalace[ϕ], Ho-Ren Chuang, Thomas Lazor, Binoy Ravindran
ECE, Virginia Tech
{*bmsaif, antoniob, horenc, tlazor, binoy*}*@vt.edu*

***Abstract*—With the advances in network speeds a single processor cannot cope anymore with the growing number of data streams from a single network card. Multicore processors come at a rescue but traditional SMP OSes, which integrate the software network stack, scale only to a certain extent, limiting an application's ability to serve more connections while increasing the number of cores. On the other hand, kernel bypass solutions seem to scale better, but limit resource flexibility and control. We propose attacking these problems with a distributed OS design, using multiple network stacks (one per kernel) and relying on multi-queue hardware and hardware flow steering. This creates a single-socket abstraction among kernels while minimizing inter-core communication. We introduce our design, consisting of a distributed network stack, a distributed device driver, and a load-balancing algorithm. We compare our prototype, NetPopcorn, with Linux, Affinity Accept, FastSocket. NetPopcorn accepts between 5 to 8 times more connections and reduces the tail latency compared to these competitors. We also compare NetPopcorn with mTCP and observe that for high core counts, mTCP accepts only 18% more connections yet with higher tail latency than NetPopcorn.**

I. Introduction

Commodity and data center network connectivity is becoming increasingly fast. Ethernet is by far the most common network technology, and network interface card (NIC) manufacturers are already shipping 200-GbE adapter cards. A single CPU core cannot handle the high bandwidth traffic generated by such NICs, but multicore processors do.

Even if CPU frequency is stagnating for physical reasons, the number of cores per machine is increasing. On a multicore CPU, a parallel software network stack allows for efficient handling of Ethernet traffic. Together, these two trends necessitate a rethinking of the design of current SMP operating systems (OSes), their network stacks, and drivers.

Linux Network Stack Scalability. The software network stack implemented in traditional SMP OSes, such as Linux, does not scale well as the number of available cores in the platform increases. This is notably true for short lived connections [1] that depend on the setup of shared control structures in the OS. Experiments using Linux 3.2.14 and 10GbE show that the Apache webserver hardly handles more connections when the number of CPU cores goes beyond 8, even if the network bandwidth and the CPU are not fully used. Figure 1 compares the case of vanilla Linux (Linux), Linux with dynamic interrupt re-routing (irqbalance), and Linux with software receive flow steering [2] (flowsteering). Each of these solutions shows minimal improvements.

The problem is not new, and it has been attributed to the semantics and the implementation (based on the shared memory paradigm) of the TCP/IP stack [1], [3], [4]. This is important because it affects an entire class of applications that provide Internet services on a single IP address and port, including webservers and key/value stores.

Complex Inter-Subsystems Interactions. Traditional OSes are monoliths where subsystems create a net of complex interactions. In Linux, for example, the network stack mainly interacts with the device and filesystem layers. Hence, the scalability problem is not fully imputable to the network stack itself. This is captured by Figures 2 and 3. The first breaks down the execution of all Apache processes in a system when varying the number of connections. Up to 65% of the execution time is spent in the kernel spinning on various locks. Figure 3 breaks down the cost associated with "accept (krn)" (Figure 2) and shows the main sources of overhead to be the network and the filesystem subsystems.

Unfortunately, in mature monolithic kernels such as Linux, reimplementing locking for a single kernel subsystem will not guarantee scalability of the subsystem itself [1]. Thus, this work is based on Popcorn Linux [5], which already implements different scalable OS services.

Effective Resource Sharing. For latency-critical network applications, the common practice today is to keep the kernel in the control-path while removing it from the data-path (known as "kernel bypass"). This avoids the possible non-scalability and higher latencies introduced by multiple software layers. Although this solution is shown to work well (see Figure 4), removing the kernel from the data path (i.e., placing the network stack in user-space) prevents any application other than the one associated with a network flow from accessing data. This limits hardware resource shareability, monitoring, network filtering, etc. Additionally, user-space solutions embody lightweight network stacks, such as lwip [6], which are feature-poor when compared to feature-rich UNIX-like kernel network stacks.

In short, there exists a large scalability gap between user-space and kernel-space network stacks. It follows then that there is an immediate need for a software network stack that provides OS-grade resource control, low-latency, and scalability – what this work seeks to provide.

NetPopcorn. We present NetPopcorn, a new network

[†] The author is now with Cisco Systems Inc.
[ϕ] The author is now with Euler, Huawei German Research Center.

stack and device driver based on the Popcorn replicated-kernel OS [5]. NetPopcorn focuses on the scalability of short-lived connections management. Specifically, NetPopcorn enables POSIX network applications to fully exploit increasingly higher bandwidths while maintaining low latencies, thus removing the scaling and high-latencies limitations imposed by current state-of-the-art OS designs. It also does so while avoiding the less-flexible hardware resource allocation enforced by kernel bypass designs. NetPopcorn demonstrates that constructing the OS with distributed system principles enables network serving applications to serve up to 8 times more connections than Linux and up to 5 times more connections than previous work [3], while additionally providing better tail latency and being no more than 18% slower than user-space network stacks.

II. Design

NetPopcorn targets modern multicore computers with multi-queue network cards. In order to scale the number of connections that the OS can simultaneously handle, NetPopcorn implements the following design principles: (1) reduce the usage of shared memory among cores (or group of cores); (2) minimize the communication among cores (or group of cores); (3) exploit hardware to reduce synchronization when needed; (4) support existing UNIX/Linux applications. NetPopcorn is implemented atop Popcorn Linux [5] to exploit the scalable replicated-kernel OS design – specifically, the distributed filesystem service. It also introduces the Snap Bean device driver model, which shares a single (multi-queue) device among different kernel instances and applies the work-sharing [7] design principle to hardware-defined entities (flow groups) to load balance network packet processing among kernels.

Snap Bean Device Driver. A replicated-kernel OS assigns each device to (and allows access by) a single kernel instance at a time, which in turn proxies device access to all other kernels (Figure 6). In [5] the authors show that proxying is efficient; however, the messaging needed to forward network packets through software to each of the other kernels can overload the proxy kernel, which becomes a bottleneck.

The Snap Bean device driver exploits the hardware packet filtering capabilities available in multi-queue NICS to remove software messaging overhead. It also extends the usual driver interface to expose the multiple queues and their configuration to the upper software layers. Common practice in SMP OS device drivers is to equally distribute queues between CPUs in a system. Snap Bean recreates a similar assignment in a replicated-kernel OS. It attaches to a NIC device driver and allows multiple kernel instances in a replicated-kernel OS to share the hardware queue pairs (Figure 7). As opposed to the proxy driver (Figure 6), each kernel receives packets from the hardware (black arrow). In Figure 7, the orange dotted lines show that a different kernel memory maps and manages each queue pair.

Opportunistic Load Balancer. We observed that work-sharing algorithms [7] introduced in de-centralized scheduling can be applied in the context of multi-queue NICs. Unlike a scheduling queue, a receive queue is not filled in by the software but by the hardware. While in de-centralized scheduling a single task moves between queues, here a group of connections that hash to the same filter value (flow group) should be reassigned in total between hardware queues.

We designed a load balancer that redistributes flow groups to queues when there is an imbalance between queues' load. Although a uniform partition of flow groups to queues works well when the incoming packets are homogeneously distributed among flow groups, a non-homogeneous distribution, creates performance degradation of user-space network services. As queues are statically partitioned to kernels, this implies that certain kernels will be overloaded while others will sit idle. Instead, the load balancer dynamically reassigns flow groups to queues (kernels) to keep the expected QoS.

In a multiple-kernel OS, each kernel possesses a separate network stack; this makes moving a flow group challenging because it can be composed of multiple active connections. In fact, migrating active connections in a replicated-kernel OS is very expensive: part of the kernel state, including the network stack, should migrate. Moreover, with live connections, the software (via the messaging layer) may need to move different packets until the hardware changes the destination queue of packets. To avoid this problem we introduce a heuristic that avoids messaging: we move a flow group between kernels only when there are no established connections through it. Within a single OS, POSIX dictates that a single ip/port server calling `accept()` *"extracts the first connection request on the queue of pending connections"* – our load balancer guarantees this.

III. Implementation

We implement NetPopcorn on top of Popcorn Linux (Linux 3.2.14) on x86 multicore hardware and Intel's 82599EN multi-queue NIC. The prototype targets TCP/IP, although we implement minimal support for other protocols.

The Snap Bean device driver is implemented as a Linux kernel module that register itself as a network device driver. Snap Bean is loaded with the information of which network device driver it should attach to (in this case *ixgbe*) on the kernel that initializes the 82599EN.

IV. Evaluation

Hardware. The experiments are conducted on a setup with 3 clients and 1 server interconnected by a dedicated 10Gb/s CISCO Nexus 5010 Ethernet switch, ensuring no external traffic to interfere with the experiments. Each client machine has a 4-GHz AMD FX 8350 8-core processor and 12 GB of RAM. The server machine has a dual socket Intel Xeon E5-2695v2 12-core processor (hyperthreading disabled) with 96 GB of RAM. We tested that 3 clients generate enough traffic to push the server to its limits.

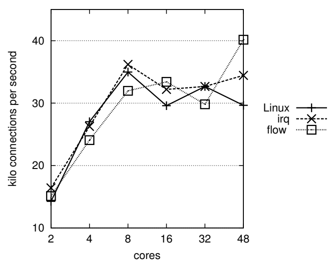

Figure 1. Apache webserver running on Linux with various network optimizations. Number of connections varying the number of CPU cores.

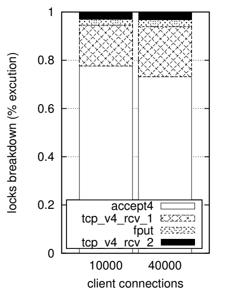

Figure 2. Apache processes locks breakdown in a system.

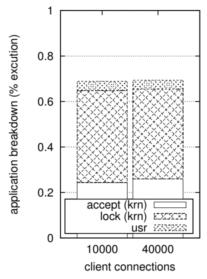

Figure 3. Apache's "accept4()" locks breakdown.

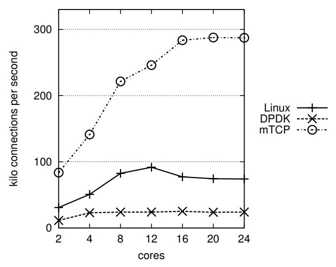

Figure 4. Lighttpd webserver running on Linux, DPDK, and mTCP. Number of connections varying the number of CPU cores.

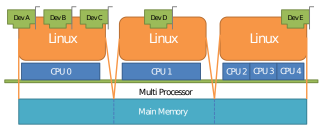

Figure 5. Popcorn static hardware partitioning.

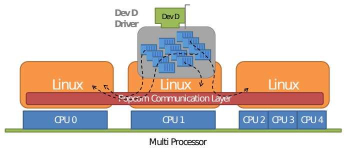

Figure 6. Proxy networking.

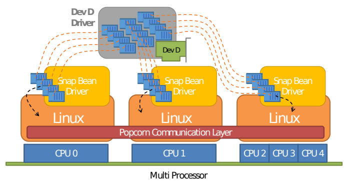

Figure 7. Snap Bean networking.

Software. We compare NetPopcorn with vanilla Linux, Linux Affinity Accept (affinity), mTCP [8] , FastSockets [1], and Popcorn Linux (proxy driver). However, we do not report Popcorn Linux's numbers as despite the 2-core setup (as in [5]), all results are worst than vanilla Linux. Vanilla Linux is version 3.2.14; Affinity Accept is based on Linux 2.6.24. NetPopcorn is built on top of Popcorn Linux for multicore platforms [5], [9], adding about $2k$ LoC to Popcorn.

We extensively compare NetPopcorn and competitors using two different web-serving applications: Apache and lighttpd. Apache starts multiple processes awaiting incoming connections on the same socket using `accept()` syscall. Lighttpd starts multiple processes as well, but each uses the `epoll()` syscall on the same listening socket. `accept()` sits at a lower layer in the system software than `epoll()`. We launched 10 webserver processes per core.

We evaluate Apache and Lighttpd with Apachebench. A webserver is started on the server machine first; then the client machines coordinate, and each starts 10 instances of Apachebench in parallel. Each Apachebench instance is configured with concurrency level set to 1000, connections to 100000, and fetches a static file of 4 B (similar to [4]). Such a small file is used to stress the OS and study its scalability in terms of handled connections.

Connection Scalability. Figures 8, 9, and 10 respectively report Apache's average number of requests per second, number of failed connections per run, and time per request. Figures 11, 12, and 13 report the same values but for Lighttpd. Lighttpd evaluation includes FastSocket and mTCP. These numbers are collected after a warm-up phase.

Comparing Apache's and Lighttpd's graphs, the main take away is that NetPopcorn always serves more requests per second than Linux and Affinity Accept. Affinity Accept outperforms Linux (up to 30% more connections), but the difference between them is larger on `epoll()` than on `accept()`. The Linux network stack behaves better when the *event poll* layer handles concurrency; with `epoll()`, Linux can handle up to $90k$ connections per second on 8 CPUs. However, adding more CPUs does not give any additional benefit – in fact, the number of connections drops for Lighttpd while plateauing for Apache. *Affinity Accept* is affected by the same stagnating effect.

mTCP accepts many more connections than any other solution. However, while this difference may be relevant for a low number of cores, the number is no more than 18% greater than that of NetPopcorn (for 16, 20, and 24 cores). *FastSocket* handles a similar number of connections for low core counts but counter-scales after 12 (in NUMA node). Both mTCP and FastSocket response times do not scale with the number of cores (versus NetPopcorn).

NetPopcorn drastically reduces the number of failed connections from 8 CPUs onward. With Lighttpd the number of failed connections drops down on the order of the tens, while with Apache on the order of the hundreds. Linux, Affinity Accept, and FastSocket have 2-3 orders of magnitude more failed connections than NetPopcorn and mTCP.

The average total time per request from the client side (Apachebench) includes any software and network overheads (and thus the actual value is less important than the trend itself). NetPopcorn is the quickest at any core count showing that a replicated-kernel OS already provides benefits at low core counts. On 2 CPUs, NetPopcorn is up to 23% faster than Linux and 16% faster than Affinity Accept for Apache and Lighttpd, respectively.

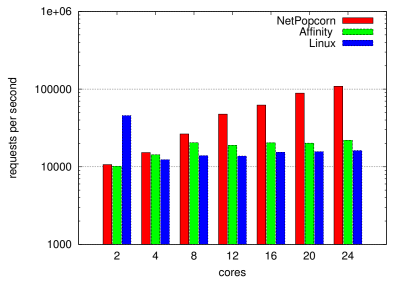

Figure 8. Apache webserver, number of requests per second.

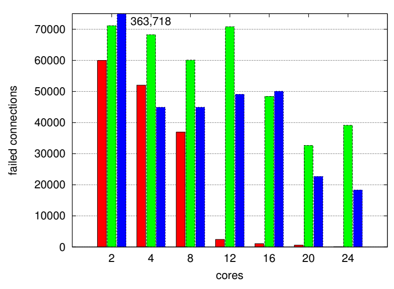

Figure 9. Apache webserver, number of failed connections.

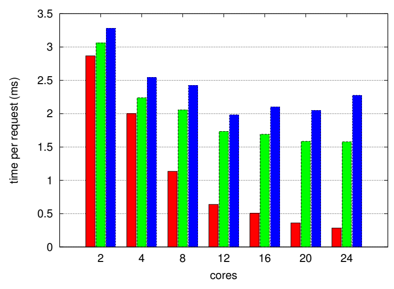

Figure 10. Apache webserver, average request time.

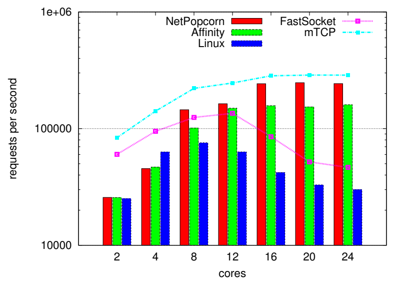

Figure 11. Lighttpd webserver, number of requests per second.

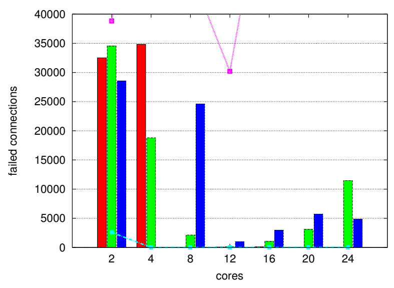

Figure 12. Lighttpd webserver, number of failed connections.

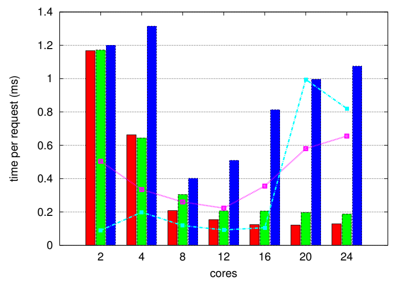

Figure 13. Lighttpd webserver, average request time.

Load Balancer Overhead. To highlight the benefits and overheads of having an active load balancer we re-run the Lighttpd experiments without the load balancer. The comparison between the results with and without the load balancer shows its benefits especially when the number of cores is 8. The number of connection serviced increases by $32k$ and the number of failed connections drops by $11k$ (although the time to handle a request increases). For higher number of CPUs, the load balancer improves the performance of the applications without any negative effect.

V. Conclusions

We introduce NetPopcorn, a distributed OS network stack and device driver based on a replicated-kernel OS. NetPopcorn exploits hardware features to reduce kernel-to-kernel communication and handles more network connections as the number of cores increases without increasing latency.

NetPopcorn shows that an OS fully developed with a distributed programming model can handle up to 5 times more connections, with 2 orders of magnitude fewer failed connections than Affinity Accept, Fast Socket, and Linux, which are all based on the shared memory programming model, and within only 18% of mTCP's maximum. We show the effectiveness of NetPopcorn's load balancer that emulates POSIX semantics. In an imbalanced scenario, NetPopcorn's tail latency is up to one order of magnitude lower than in Affinity Accept, Fast Socket and mTCP.

NetPopcorn is available at popcornlinux.org.

Acknowledgments

This work is supported in part by grants received by Virginia Tech including that from ONR (grants N00014-13-1-0317 and N00014-16-1-2711), AFOSR (grant FA9550-14-1-0163), and NAVSEA/NEEC (grants 3003279297 and N00174-16-C-0018).

References

[1] X. Lin, Y. Chen, X. Li, J. Mao, J. He, W. Xu, and Y. Shi, "Scalable kernel tcp design and implementation for short-lived connections," in *ASPLOS '16*, 2016.

[2] "Scaling in the linux networking stack," https://www.kernel.org/doc/Documentation/networking/scaling.txt, 2015.

[3] A. Pesterev, J. Strauss, N. Zeldovich, and R. T. Morris, "Improving network connection locality on multicore systems," in *EuroSys '12*.

[4] N. Z. Beckmann, C. Gruenwald III, C. R. Johnson, H. Kasture, F. Sironi, A. Agarwal, M. F. Kaashoek, and N. Zeldovich, "Pika: A network service for multikernel operating systems," *MIT CSAIL*.

[5] A. Barbalace, A. Murray, R. Lyerly, and B. Ravindran, "Popcorn: a replicated-kernel OS based on Linux," in *OLS*, 2014.

[6] A. Dunkels, "Minimal TCP/IP implementation with proxy support," 2001.

[7] L. Rudolph, M. Slivkin-Allalouf, and E. Upfal, "A simple load balancing scheme for task allocation in parallel Machs," in *SPAA III*.

[8] E. Jeong, S. Wood, M. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park, "mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems," in *NSDI 14*.

[9] D. Katz, A. Barbalace, S. Ansary, A. Ravichandran, and B. Ravindran, "Thread Migration in a Replicated-kernel OS," in *ICDCS XXXV*, 2015.