

vFlood: Opportunistic Flooding to Improve TCP Transmit Performance in Virtualized Clouds

Sahan Gamage(Student author), Ardalan Kangarlou(Student author), Ramana Rao Kompella and Dongyan Xu
Purdue University
{sgamage,ardalan,kompella,dxu}@cs.purdue.edu

Virtualization is a key technology that is powering cloud computing platforms such as Amazon EC2. Virtual machine (VM) consolidation, where multiple VMs share a physical host, has seen rapid adoption in the industry with increasingly large number of VMs per machine and even per CPU core. By breaking away from the traditional model of hosting applications in physical machines, virtualization enables the “slicing” of each physical machine in a cloud infrastructure into multiple virtual machines (VMs), each individually hosting a server, service instance, or application component.

Meanwhile, many cloud applications tend to be *networking intensive*. The reason lies in the wide adoption of distributed computing techniques such as MapReduce for large-scale data processing and analysis. In addition, many scalable online services (*e.g.*, e-commerce, Web 2.0) hosted in the cloud are often organized as multi-tier services with server load balancers redirecting clients to web frontend servers, which in turn interact with backend servers (*e.g.*, database, authentication servers). These applications involve communication across multiple servers in the datacenter, and thus the application-level performance is directly dependent on the network performance.

Unfortunately, as our recent studies [1] suggest, these two trends are directly at odds with each other. Our key observation is that the VM consolidation (i.e sharing of CPU by multiple VMs) negatively impacts the VMs’ time sensitive I/O tasks such as TCP, which in turn affects the overall performance of many cloud applications. More specifically, with multiple VMs sharing the same CPU, the latency experienced by each VM to obtain its CPU time slice increases. In this work, we primarily focus on TCP connections in a VM that are transmit intensive, i.e., the VM transmits a lot of data to a client. For such connections, the acknowledgements corresponding to the data packets may arrive from the receiver quite fast, but because of the extra scheduling latency, the time it takes for the TCP sender to observe the acknowledgement increases significantly causing the congestion window to grow slowly over time.

To address this problem, we propose a novel technique in which the VM TCP stack opportunistically floods as many packets as possible to the driver domain during its CPU time slice. By flooding many packets, one can trivially improve the TCP connection progress significantly, but will also lead to congestion collapse in the network. Thus, in vFlood, the driver domain implements

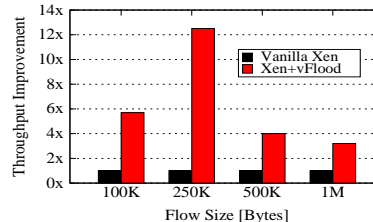


Figure 1: Throughput improvement for small flows

the TCP congestion control function on behalf of the sender VM and thus, as far as the network is concerned, the connections are functionally equivalent to any other standards-compliant TCP implementation, but will make much faster progress compared to the native TCP sender. Such opportunistic flooding effectively masks the sender VMs CPU access latency from the RTT of TCP packets, thereby improving the end-to-end throughput of the connection significantly. vFlood design has three main components. 1) A component inside a VM that advises VM TCP stack to flood as many packets as possible. 2) A component inside the hypervisor or the driver domain, which sends out packets to the external network in a controlled fashion and 3) A communication channel between VM component and the hypervisor component.

Our prototype implementation of vFlood on Xen consist of two Linux loadable modules, one inside domain-0 and another inside VM. While our prototype implementation is based on Xen, this approach is generally applicable to other virtual machine monitors (VMMs) where the actual network drivers reside in a driver domain or at the VMM layer.

Our evaluations show that vFlood is very effective for small flows. Figure 1 shows the TCP throughput improvements for transfers of varying flow sizes with and without vFlood. In this experiment, 2 non-idle VMs (40% CPU load) share the same CPU core with sending VM. These measurements show that vFlood yields from 3.2X to 12.5X performance improvement. Also, our experiments show that vFlood yields non-trivial improvements for larger transfers (23% to 31% for 100MB transfers). Our ongoing work focuses on investigating the applicability of vFlood to different cloud applications.

References

- [1] A. Kangarlou, S. Gamage, R. R. Kompella, and D. Xu. vS-noop: Improving tcp throughput in virtualized environments via acknowledgement offload. *ACM/IEEE SC'10*, 2010.