Overview

OpenNetVM (ONVM) is a highly efficient network packet processing framework that provides simple abstraction for developing and running network functions. The ONVM platform provides load balancing, flexible packet flow management, individualized service abstractions, and basic software defined networking (SDN) capabilities. The platform already offers best-in-class performance: it has the ability to maintain 40 Gbps speeds while routing packets through dynamically created chains of network functions (NFs). OpenNetVM also lowers the barriers to deploying production network functions in software because it can run on inexpensive commodity hardware. However, this idea can be improved upon: we aim to make it possibly for multiple instances of ONVM to process packets in tandem, with seamless inter-routing, making it possible to run software-based network services at production scale.

Subtopic: Network Function Virtualization
Key Words: networking, software defined networking, packet processing, the cloud

Intellectual Merit

This Small Business Innovation Research Phase I project will push the limits of the performance possible by commodity servers. OpenNetVM is already a highly optimized platform, and any extensions will need to maintain existing performance standards. One technical challenge will be implementing an intelligent algorithm to discover the optimal node to route a packet that can not be processed locally. The system will need to scale to an arbitrary number of instances of ONVM that can collaborate to process packets in the most efficient way possible, especially if there are multiple possible destinations for a packet. The system needs to keep management overhead minimal and ensure the NF critical path is as fast as possible, because excessive network traffic and computation overhead will degrade the capabilities of the “production” network.

In order to reach these goals, the existing platform needs to be modified slightly to support distributed capabilities. A new abstraction layer needs to be added, so the system can process packets going to and from either local or remote instances of ONVM. This abstraction layer should be generic enough that the two critical paths share as much code as possible and remain highly optimized. Then, a routing protocol that meets the above considerations needs to be designed and implemented. This will necessitate the inclusion of a distributed datastore so that the nodes can share data; the cost of additional overhead will be a major performance consideration, since the platform can not afford constantly doing slow data lookups.

Broader/Commercial Impact

This research will drive costs for enterprises building production-scale network services. OpenNetVM can run on cheap commodity hardware, reducing the need to order specialized appliances. For example, using a specially crafted NF running in OpenNetVM, a company can analyze all packets flowing into and out of their network and transparently filter and quarantine malicious traffic without performance degradation. Since the NF is running software, the

company does not need to purchase dedicated Intrusion Detection System (IDS) hardware, and malware signatures can be rapidly iterated. Additionally, running all NFs on standard hardware allows users to more efficiently allocate their hardware capacity, all while spending less capital on servers. This project can revolutionize the NFV space, bringing high performance scalable networking applications to enterprise consumers at a significantly lower cost.

Elevator Pitch

OpenNetVM is is intended to benefit enterprise customers. These companies run large-scale, high-performance networking applications. These network applications tend to require expensive specialized hardware that can only be purchased from one of few suppliers. For example, a large enterprise typically has an Intrusion Detection System (IDS) protecting the entry point to their network. A popular website or an entry point to the corporate VPN may saturate a 10 Gigabit connection with incoming requests. All of this traffic needs to be screened for malicious traffic. Furthermore, the enterprise may also require a firewall, DDOS detection system, and dynamic routing (like Software Defined Networking); all of these features require different expensive and specialized hardware. So, when building an enterprise-quality network, the infrastructure costs will add up and can quickly become burdensome.

OpenNetVM will allow large enterprises to run their high-performance network applications at a fraction of the cost. OpenNetVM can run on commodity hardware which vastly reduces the amount of capital required to spin up a corporate network. This platform will not sacrifice performance either; packets can be processed at 10 Gigabit speeds, the same speeds as the more expensive dedicated hardware. By utilizing the flexibility of software, OpenNetVM also integrates dynamic network topology. Packets can be routed via a “service chain,” which means a packet flow can be processed by multiple network functions (NFs). This functionality allows the functionality of what would previously require multiple dedicated appliances to be consolidated into a smaller number of commodity servers. Furthermore, as system requirements change over time, the topology of the network function interconnections can be dynamically modified. For example, if the company’s website is experiencing additional load, more instances of a load balancer can be started to accommodate the additional traffic and spread the incoming requests appropriately.

OpenNetVM is able to provide these revolutionary features by implementing Network Function Virtualization (NFV). For every “appliance” that an enterprise IT department would want on their network, they can abstract the functionality into software and run it on the OpenNetVM platform. OpenNetVM uses the high-performance DPDK networking library to interface directly with the hardware. Packets can be shared between NFs and processed without copying overhead – the platform works entirely with shared memory. Furthermore, the OpenNetVM platform will be able

to function in a distributed manner: packet flows can be seamlessly routed among multiple hosts to be processed in an optimal way. This means the system can scale seamlessly and automatically recover from failed hosts without loss in service. Automatic failover and horizontal scaling provide features that enterprise customers can rely on in production operations. Traditionally, these features were very expensive to implement, as they require specialized networking hardware. OpenNetVM can provide these advantages for free on inexpensive servers by harnessing the power and flexibility of software.

Commercial Opportunity

OpenNetVM is targeted towards enterprise consumers for use in their data centers. These companies often serve public facing services, internal applications, and other networking infrastructure from their own data centers. These services traditionally need specialized hardware to run and additional hardware to properly secure. Networking hardware is expensive to purchase and difficult to configure and maintain. OpenNetVM aims to alleviate these pain points by utilizing the flexibility of software to dynamically create performant and secure networks for enterprise applications.

OpenNetVM is an open platform - the software is open source and can run on commodity hardware. This is critical for growing a user base as there are few barriers for a potential customer to try the system. The first step in growing the platform is to get users, and an open platform that can be cheap, fast, and secure is a great value proposition. However, enterprise customers demand stability and reliability: we need to offer support, and this is our route to monetization. We will model ourselves after RedHat, a company that has had great success monetizing free software by providing support to enterprise customers.

Enterprise support contracts involve large dollar amounts, or enough to sustain our business. Our goal is for OpenNetVM to be easily evaluated (since it is a free and open platform) and then the customer is impressed at the savings due to the reduced hardware requirements, so they purchase an annual support plan to keep the system maintained. This approach will still end up being cheaper up front to the customer because networking hardware is incredibly expensive. Furthermore, OpenNetVM will provide immense future flexibility: the company easily add services to their network and dynamically scale as usage grows, all for only the cost of commodity hardware.

It is hard to overstate the value of quality support to an enterprise customer. Companies will pay for the privilege of reliability, especially when their revenue-drivers are on the line. We will provide support for outages and help troubleshoot failures and bring the system back online. We will also provide R&D support for a company evaluating our project and assist them while they

are exploring OpenNetVM. If they have any questions during this process, we'd be able to give them priority support in answering their questions.

Another means of revenue for OpenNetVM is prioritized feature development. If someone is interested in seeing a specific feature get developed, they can either pay to prioritize it or wait for it to get accomplished from the open-source roadmap. If someone is prioritizing a feature, they will pay an hourly "consulting" rate for development efforts related to implementing that feature. This is valuable for companies that have a slightly specialized use case for OpenNetVM. If they need a certain change made and can't wait (since it would affect their business operations), they can pay to have it written faster, generating supplemental revenue for the project.

There are few competitors in the NFV (Network Function Virtualization) space. The closest competitor is ClickOS (http://www.read.cs.ucla.edu/click/). ClickOS is a "modular router" that can dynamically route packets. Both products are already currently in the market in beta form as research projects, but ClickOS does not appear to support a "clustering" mode, limiting its scalability potential. OpenNetVM's clustering capabilities allow it to grow to infinite scale and process arbitrarily large packet streams, making it a better candidate for running enterprise services (which need to run at incredible scale).

Societal Impact

OpenNetVM is a platform for other people to build products, therefore, there is not a lot of direct societal impact. However, it is a platform other companies build upon to make the world a better place. The cloud has played a major part in making technology more accessible to the world, and wide availability of the OpenNetVM platform will continue this trend. Recently, the barrier to entry in the tech space has been drastically reduced by the commoditization of cloud service infrastructure (it is easier than ever to create an app or website and bring your idea to the world). OpenNetVM takes that concept and brings it to networking infrastructure. Ideally, other companies can build amazing services on the OpenNetVM platform and positively impact society.

There are not any real regulatory issues surrounding this project. It is an open platform, meaning that users can do as they wish. OpenNetVM is released under a BSD License, which allows the source code to be redistributed without repercussion (provided the license is maintained) and is provided "as is" without any explicit or implicit warranties. Therefore, users can do whatever they like with OpenNetVM, and that is perfectly fine. Ideally, the project would only be used for "good" purposes, but that can not be controlled under the terms of the project

license. We do not anticipate any malicious or unethical uses of the project, but seeing as it is an open platform, there is a very little amount of control we can exert on it.

Any group of people can be impacted by this project. A new startup would be wise to use this platform to host their services. They can leverage OpenNetVM's scaling feature and not have to worry about infrastructure (and instead focus on building the best application possible). The startup can also use security-focused NFs, like an IDS (Intrusion Detection System) or a Firewall to ensure the integrity of their application. And when this startup needs to change their configuration, they don't have to wait on any lengthy hardware orders, since they can simply reconfigure their network on the fly due to the flexibility of software-based networking. Finally, as the startup grows, they can leverage OpenNetVM's native load balancing capabilities to make sure they don't exhaust the capabilities of their hardware and continue providing valuable services to their customers, even in the face of heavy loads. All of this can be accomplished using only commodity hardware in the data center. Not being reliant on hardware orders allows the company to not have to worry about their networking infrastructure and focus on the parts that matter: building great services to make the world a better place.

Technical Discussion

There are numerous technical challenges that must be solved before bringing OpenNetVM to market. First and foremost, there needs to be some way to coordinate data between multiple running instances of the ONVM manager. This distributed datastore will contain information about the physical MAC addresses of running managers, which managers are running which services (and how many instances), and some stats for each NF. This data will allow ONVM to make intelligent decisions on how to route packets around the cluster of instances. This datastore needs to be automatically synchronized across the cluster of managers without consuming bandwidth needed by the packet processing critical path. The datastore also needs high availability and redundancy built-in. This is because the ONVM platform can not inherently trust user-created NFs so they may crash at any point without notice. Therefore, the distributed datastore needs to be failure-tolerant, as well, so that packets are not incorrectly routed to NFs no longer running. Finally, updates to the datastore need to be ordered and atomic. Like any problem solved using distributed data structures, ONVM needs to be able to gracefully handle contended writes (for example, if two instances of one service are started at the same time on two separate nodes, the overall count needs to be correctly updated).

The next technical challenge is how to exactly send packets from one instance to another. Once the OpenNetVM manager realizes that a packet is destined for a service not running locally, it must forward the packet to a host that is running the desired service. This means the packet must be encapsulated with another set of headers so that network infrastructure can properly

route it to the new destination. Additionally, the manager must also send along the manager's metadata (e.g. the destination service and current index in the service chain). This includes a new set of ethernet and IP headers. Next, on the receiving end, the manager must be able to discern these encapsulated packets (using an application layer protocol) from other regular incoming packets, decapsulate them, restore the ONVM metadata, and enqueue them to the proper NF.

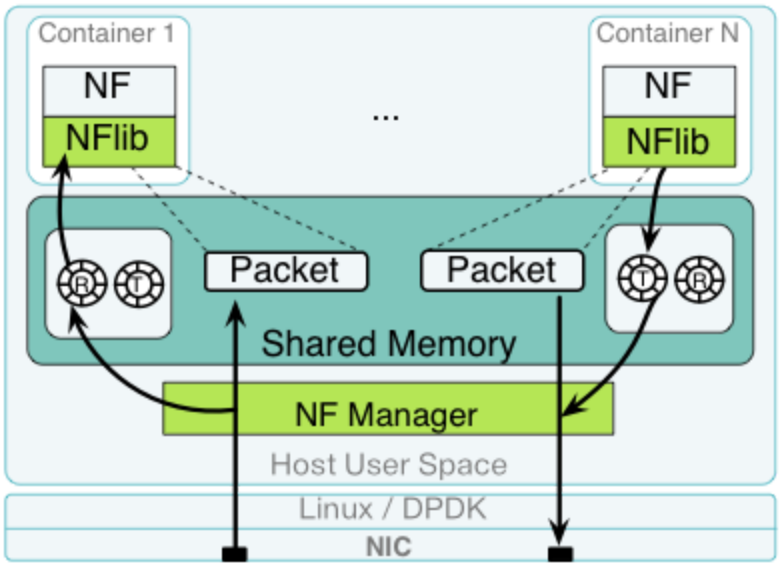
Finally, the biggest technical challenge will be automatically scaling NFs to account for changes in load. The manager will be able to recognize the situation when an NF can not handle the amount of packets it is being asked to process and then start another copy of the NF. OpenNetVM can already load-balance across many instances of a service, so the packets will then be automatically sent to the new instance. Additionally, the manager should also realize when multiple NFs have a capacity excess and signal some instances to shut down. This will free up resources on the server for a different, more needed, NF to be running instead. This will be difficult to implement for a number of reasons. Primarily, the existing NF library and architecture may need to be reconstructed in order to be "reentrant" (or safe to have multiple copies running at a given time). This may require a change in some of the interfaces used by OpenNetVM. Some NF implementations may not support auto scaling at all (due to how they are implemented) and need to be able to indicate that to the manager. Finally, if a single service is running on multiple hosts, the manager needs to use the statistics stored in the distributed datastore and make a decision about which hardware to start the new copy on. It may take into account the power of the hardware and remaining CPU cores available to process the packets.

Automatic scaling of NFs is a substantial technical innovation. Companies pay large sums of money to cloud infrastructure providers (Amazon AWS, for example) that can automatically scale their services according to instantaneous demand. This is highly valuable for systems that are not under constant load (this is most user-facing systems). This innovation is also beneficial for the environment; energy is not wasted powering services at idle and existing capacity can be more efficiently allocated to the current needs of the system. It means that sysadmins do not have to procure extra expensive hardware that sits idle when unused and it means current hardware capacity is not being wasted.

The point of OpenNetVM is to provide a fast and scalable NFV platform that can run on commodity hardware. In this scenario, every bit of computation power is valuable (since commodity servers are relatively low-powered). Therefore, OpenNetVM must be incredibly efficient in the way it allocates resources to packet processing jobs (and any wasted power or unoptimized accesses can slow down the critical path to an unacceptable level). Increased coordination between running instances allows OpenNetVM to be smarter about how it allocates

its resources. This kind of intelligent resource utilization will allow enterprises to spend less capital when building out networking infrastructure and provide an easy avenue to increase capacity in the future. Currently, there are no systems that provide this kind of NFV scalability on the market: OpenNetVM would be the first.

The core technical innovation at play here is to determine when an NF is “overallocated” or “underallocated”. In OpenNetVM, each NF has an “RX Ring”: a circular buffer that transports packets from the manager to the NF. The manager enqueues packets into the ring, and the NF dequeues them, processes them, and sends them back to the manager via another ring (where the manager can enqueue them to another NF or out on the NIC). These buffers have a finite size (by default, they can hold 128 packets at a time). The manager can track the utilization percentage of each NF’s RX ring. This metric will provide insight into whether the NF has more packets enqueued than it can handle. In that scenario, the ring will be highly utilized and the manager may even be forced to drop packets if the ring is completely full. Consequently, this is a good opportunity to start another copy of the NF so that all the excess packets can be “load-balanced” to the new instance. On the other side, if the total RX ring utilization (across all instances of the service) is low, then it may be a good idea to stop one of the NFs to free up resources. This scenario would happen when a fewer number of instances can process the entire packet load destined for that service.



In OpenNetVM, an NF is “pinned” to a CPU core. This means only that thread is allowed to run, granting the packet processing loop exclusive access to that core. A context switch (running something else on that core) is a large overhead to incur, and will most likely result in dropped packets (since the RX ring will fill up while the NF’s process is not currently executing on the CPU). Therefore, the number of NFs a single machine can run is limited by the number of cores on that machine. This information (the number of remaining unused cores) will need to be shared in the datastore so that the manager can decide where (on which machine) to start new instances of a given NF. New NFs will need to be started on machines that have available CPU capacity and the algorithm should do its best to ensure an optimal allocation of NFs per machine. It may be more performant to spread all NFs across machines or to “fill” one machine

to capacity before bootstrapping new NFs on a second. Some experimentation will be required here to determine which strategy is best. Furthermore, A newly starting NF copy must also be on the same machine as a copy that is already running (so that the binary is already loaded into memory with the correct arguments supplied by the user). The autoscaling algorithm will have to take these factors into account when deciding when and where to start new NFs.

During Phase I of the research, the first step is to make it possible for packets to be routed to an external instance of the ONVM manager. When it finds a packet destined for a service not running locally, the manager needs to look up an appropriate destination (one that does have the desired NF running), encapsulate the packet, and send it to that machine over the network. Once this functionality has been completed and is working, we can perform benchmarks to ensure that performance is not sacrificed. Ideally, the system can still process packets at line speed (at least 10 gigabit/second), even when every packet must be sent remotely. The distributed routing algorithm is designed so more than one “hop” should never be required for a packet to be sent to a machine that is actually able to process it. Furthermore, packets are sent between instances using the same network interface as they come in, so 10 gigabit speeds should be maintained. Finally, datastore synchronization should be off the critical path to save CPU time and network bandwidth. Since the workload should consist predominantly of reads, there should be little performance overhead from excessive write locks and from blocking on data coming in on the network. However, these assumptions should all be verified once the initial work has been completed. If any of these fail to hold, then the overall approach to the problem may need to be reconsidered. If the performance benchmark results are encouraging, then we know OpenNetVM will be in a competitive position in the market. Enterprise customers really care about the features ONVM offers and will be heavily interested in using it if it can fufill some of their requirements cheaply and provide provisions for future scale increases, as well. Since OpenNetVM is a free and open source project, there really is no other aspect of commercial feasibility (other than possibly engaging in enterprise support agreements for entities using the software in production services).

As stated above, the first technical milestone is to build functional distributed routing. This includes configuring the distributed datastore, creating the encapsulation/decapsulation module, and integrating this code into the existing OpenNetVM codebase. At this point, the datastore needs to support the following items: storing running instance of the manager as well as their physical MAC address and a list of which services are running on which manager instance as well as the number of copies. These items are used to make routing decisions. The encapsulation/decapsulation module also needs to be integrated into the existing packet processing code path, but should only be invoked when necessary. The success criteria for the first milestone is to be able to run some example programs on a small cluster and ensure that

packets are being routed to a secondary instance of ONVM when the desired service is not running on the first. This milestone should be completed by the end of November 2016, for the end of semester (36% completion) demo.

The next milestone is to refactor the existing NFLib to support reentrancy. Currently, some NF state is stored in variables global to the program - this technique only supports one copy of the process running at a given time. For example, the NF support library currently stores a struct of NF state (including an instance ID) as a global variable. If there are two threads running the same function, they will both refer to the same memory and the library will be unable to represent two NFs running with unique IDs. In a different approach, memory needs to be allocated for each running instance and should be passed around the function skeleton so that each incarnation of the function can run safely. Furthermore, the interface that all NFs are expected to implement may need to be changed to support this refactor. This may include changed function signatures or added callbacks into the NF lifecycle so that the manager can exert finer control over when NFs run. The interface will also need to support starting another copy of an NF's packet processing loop. This means that the current initialization function may need to be split up into initialization that only needs to be run once and initialization that needs to be run per instance. A function will also need to be added so that an NF can gracefully be shut down. This phase will not have any functional changes, but simply be an in-place refactor and will lay some preliminary groundwork for future milestones. No functional regression should occur as a result of these changes and this refactor should be applied to all existing NFs currently in the OpenNetVM repository. After completion, we will run a series of regression tests to ensure that all existing NFs remain in an functional state. The goal is for this milestone to be completed by the start of the Spring 2017 semester (by January 16, 2017).

The next milestone is to implement a means of passing control messages from the manager to an NF. The two processes already have a shared communication channel (the packet RX ring) which may be appropriate to use for this purpose. If that is the case, NFLib code will have to be modified to filter out control messages so that they are not improperly sent to the packet handler code. This should be a small consideration, because the volume of control messages will be significantly smaller than the volume of regular packets to be processed. This milestone will be successful when the manager can send "control messages" to NFs, which can then take action based on the control message, without affecting the actual NF processing. Additionally, all existing NFs in the repository will have to be tested to ensure that their functionality is not affected by this change. If some NFs interface with the ring buffers directly, they may need to be modified to ignore these control messages. This milestone does not depend on the previous one, so they can happen in parallel. This milestone will also be completed by January 16, 2017.

Finally, the last milestone is to implement NF auto-scaling. The previous two milestones will have laid all the requisite groundwork and will have made all the required changes to existing interfaces across the entire repository. To accomplish this milestone, statistics can be integrated into the datastore and a scaling module can be added to ONVM. These statistics will include the RX ring utilization percentage of each NF so the manager can determine when an NF is operating at capacity. Then, the scaling module will consume the statistics for local service and make a determination if any should be started or stopped. Success is twofold. First, a new NF should automatically start (and be seen on the stats display) when an existing one gets overloaded. This can be tested by implementing a “dummy” NF that gets stuck in an infinite loop and never actually processes any packets, simply letting the RX ring fill up until packets are dropped. Second, run an NF with excess instances that processes packets very quickly. One or more of those excess instances should be stopped until those that remain can process the incoming packets. This functionality will be completed by the time the final project is due, or 4/1/2017 (with some buffer time for testing final project cleanup).

Distributed OpenNetVM is a valuable contribution to the Network Function Virtualization space. It will allow NFV platforms to become smarter and more dynamic without sacrificing raw performance. The underlying technical innovations are complex enough to create new features that do not currently exist in the industry and will be able to make a significant impact on the market. The implementation plan outlined above will allow the team to ship a successful product and to meet all required deadlines.