\chapter{Introduction}

With the explosion of connected devices in today's ubiquitous computing environment, a whole new class of data-intensive applications have emerged which demand data processing at high-input rate. Such applications process unbounded volume of data arriving at a high rate. While having unbounded volume of data may have it's merits, it also means that applications have to evolve a different data-persistence strategy and be efficient in filtering out the signal from noise. Traditional query processing systems which issue a single non-persistent query on persistent data are unsuitable to live up to the demands of such applications. This led to the development and adoption of continuous query \cite{Chen} and Stream Processing \cite{chakravarthy} paradigms which in contrast apply persistent queries to streams of incoming data. Here a query is long-running and is evaluated continually against incoming data, and only the data which satisfies the conditions of the query is selected for further processing. Cugola et al. \cite{cugola2012processing} classify Complex Event Processing (CEP) as an important characteristic of the Stream Processing paradigm. Being able to detect patterns in streams of data and trigger necessary user or application defined events is a key requirement of stream processing engines. Complex Event Processing which evolved exactly for this purpose, offers solutions for real-time detection of patterns in data streams and triggering of appropriate actions based on the user or application logic.

\newline \newline

Because of the scale of modern day IT systems, large scale distributed CEP deployments like many other distributed applications are hosted on commodity servers. Although there is an increasing trend of taking advantage of Fog Cloud computing resources \cite{bonomi2012fog} for high velocity, high variety streaming applications, the interplay between the conventional cloud and Fog cloud is expected to increase as the demand for meaning and analysis increases; making Fog localization a supplement to a global cloud deployment.

In the midst of all this enters into fray a Software Defined approach to Networking [\cite{Jain}, \cite{casado}] which aims to separate the control plane of the underlying network from the data plane, thereby enabling rapid deployment of network services and paving way for deploying network services as virtual functions within the virtualized network stack [\cite{sherwood2009flowvisor},\cite{han2015network} ] and eventually to service function chaining \cite{halpern2015service}.

\newline \newline

Emergence of Network Functions Virtualization is symptomatic of the advances in multi-core commodity server architectures and high speed network cards. It allows network architects to leverage virtualization technologies to implement network functions and run it on commodity hardware instead of using dedicated network appliances. These virtual functions can be provisioned on-demand without the need for installing expensive equipment. Network Functions Virtualization and Software Defined Networking provides network operators with immense opportunities to monetize the underlying infrastructure. With user applications and network functions running within virtualized containers, a valid research question would be how much of the application context can the Network Functions be made aware of to extract greater performance of the application without disaffecting the network service. In this thesis, an attempt is made to ask the question in the context of Complex Event Processing. The latency-sensitive characteristic of a CEP application combined with high volume, high velocity, and low signal-to-noise ratio characteristics of the arriving data make it an optimal target for offloading certain application context onto the network.

\newline \newline

Buchmann et. al \cite{Hinze:2009:EAE:1619258.1619260} review various contributing technologies in the field of Event-Based systems. This thesis proposes a paradigm of viewing the underlying network as a contributing technology. Offloading application context onto the underlying network allows network architects to tune their network service to latency sensitive applications. It also provides better insights into the traffic characteristics of the application which may be used for designing better staged processing, load balancing and reducing the overall clog on the network I/O stack. Finally, it presents network operators with opportunities to explore further monetization of the network and virtualization infrastructure and pro-actively scrutinize revenue models which blur the line between application land and the network.

\section{Problem Statement}

To understand the research question, let us consider a single node deployment of a CEP engine within a Linux Kernel Virtual Machine (KVM) \cite{kivity2007kvm} guest. Linux KVM uses a per-guest userspace QEMU \cite{bellard2005qemu}, a fast machine emulator to run one Operating System on another. KVM-QEMU uses the virtio \cite{russell2008virtio} network interface specification, which provides a series of Linux drivers for various hypervisor implementations. It specifies a standard with a virtio front-end driver within the guest kernel, a virtio-net device which is the back-end driver within the per-guest QEMU process, and a transport mechanism in the form of vring or virtqueue between the two. When the path of a packet at the host Physical NIC to the CEP application inside a Virtual Machine is examined:

\begin{figure}[H]

\centering

\caption{Data path in virtualization environments}

\includegraphics[height=12cm]{Vswitch05.pdf}

\end{figure}

1. The packet at the physical device is handled by the NIC driver and lifted to the vSwitch. . The vSwitch switches the packet to the TAP device of the corresponding Virtual Machine. \newline \newline

2. The host injects an IRQ to notify the guest about the incoming packet. \textbf{(VM-ENTRY)} The guest schedules the QEMU userspace process within which resides the emulated virtio-net device.\textbf{(VM-EXIT)} The virtio-net device executes a \textit{read} system call (user-kernel-user transition) to receive packet from the TAP device and pushes into the virtqueue. \newline \newline

3. The virtio driver in the guest receives a callback \textbf{(VM-ENTRY)} once there is data in the virtqueue, and executes a \textit{get\_buf} to deliver the packet to the network stack of the guest. The packet is processed through the network stack of the guest and finally queued to the transport layer socket, ready to be processed by the CEP application running in user space of the guest. (guest kernel- guest user transition).\newline \newline

4. The raw bytes within the packet are de-serialized into an Object representation by the CEP application and normal application processing ensues.\newline

A similar process is used - in reverse - to send a packet out. In many event processing systems, a multiple layers of processing is used where the events are directed from one engine to another, depending on the user application logic, either for consumption or for further processing. As it can seen there are multiple context switches, system calls and packet copies to deliver packets to the guest. Even after this point the packets have to go through the complex Linux networking stack \cite{beifuss2015study} to be delivered to the CEP engine. This is especially problematic because of the following characteristics of Event Processing applications:

\begin{itemize}

\item High Data Arrival Rates

\item Low Signal to Noise Ratio

\item Staged Processing

\end{itemize}

Although CEP engines implement their own version of flow control to cope with the arrival rate, it results in latent reaction to events. In addition event processing may also be a staged process with different stages handled in different virtual machines, adding extra burden on the I/O stack of the host. These inefficiencies cannot be handled within the CEP engine. An effort to use the underlying network services to relieve the burden on the CEP engine is worth considering, particularly now more so because of the emergence of Network Functions Virtualization and Software Defined Networks.

\section{Contribution}

The main contribution of the thesis is in providing an application-aware virtual switch that detects user-defined event patterns and applies user-defined actions. The main idea behind this implementation is that an earlier detection of events and subsequent application of appropriate actions before the packets traverse through complex path described in section 1.1 would reduce the point-to-point latency between source and sink of the data. The implementation also aims to aid in staged event processing by utilizing the Layer 3 routing capabilities of the virtual switch and enabling it with application context.

\newline \newline

Mekky et. al \cite{mekky2014application} provide an application-aware implementation of Open vSwitch, which is capable of content-based server selection and load balancing. But they rely on the controller heavily to adapt to packet flows, which is not ideal in a streaming scenario. Zhang wt. al \cite{zhang2014smartswitch} implement a mem-cache aware SmartSwitch prototype to cache data blocks near worker nodes. The switch interprets application data inside the packets and redirects them to the appropriate servers. However, the implementation is done on a custom virtual switch and currently offers only redirection and load balancing capabilities. \cite{hwang2015netvm} describe and implement a high-speed packet processing platform, NetVM, built using Intel's DPDK library. Within this implementation is an hypervisor based switch that is capable of steering traffic by learning the application context. However, NetVM comes packaged as a middlebox solution to be used in the network functions pipeline.

\newline \newline

There are several publications in the area of high performance packet processing [\cite{gallenmüller2015comparison} , \cite{ning2013virtualization}] within cloud networks. They mostly aspire to zero-copy I/O frameworks and to reduce the number of context switches needed during packet I/O.

Vhost-net is an accelerator to the virtio framework, which provides vhost-net device in the kernel space which can DMA to virtio front-end in the guest. It aims to avoid the context-switch between the VM and QEMU during packet I/O. But this framework requires a modification to the host kernel itself to support the vhost-net device, since the device is not tied to KVM. DPDK \citep{scholz2014look}, netmap \cite{rizzo2012netmap}, and PF\_RING ZC \cite{kim2017study} offer shared-memory and modified drivers to by-pass the default network stack and allow the application to directly manipulate the packet buffers using custom API. However, this also has the disadvantage of monopolizing the NIC for a single application. VALE \cite{Rizzo:2012:VSE:2413176.2413185} is a virtual ethernet switch for high-speed inter-VM communication based on netmap API and batched processing. This is however highly tuned for inter-VM communication only and also is an L2 switch. While the aforesaid works try to improve packet I/O using different techniques, they do not specifically cater to a characteristic in the Event Processing domain, where majority of packets are discarded by the application after processing, or are forwarded to another machine. More will be discussed about the research areas which inspired and provided groundwork for the thesis in Chapter 3.

\newline \newline

The main priority of the thesis is to provide a production switch for offloading aspects of Event Processing applications onto the network. The vSwitch is not designed to be a complete replacement for an Event Processing engine, but rather as a network based complement to it. To this end, the following contributions have been made:

\begin{itemize}

\item A context-aware Open vSwitch implementation is implemented to detect event types and steer them to virtual machines and facilitate staged processing without having to context switch into an intermediate virtual machine.

\item The Open vSwitch is enabled to execute stateful logical operations on the data items of an event and filter based on application logic.

\item A Ryu controller based implementation is provided to enable applications to offload user logic on to the vSwitch using a http and JSON based northbound API.

\item Evaluation of the implementation in different deployment modes and a comparison to the performance of a source-tree based Open vSwitch deployment is presented.

\end{itemize}

\section{Outline}

The thesis document has been structured as follows. In Chapter 2, the different design and deployment paradigms, concepts, and technologies that provide a context to the thesis are explored with the intention to provide a brief introduction to the paradigms, protocols, libraries, development kits and deployment modes used for the design, implementation and evaluation of the thesis. Following that, in Chapter 3, a survey into the various publications and research projects which carry out inquiries in the same spirit of the thesis is laid out with an attempt to contrast and highlight the difference in contribution. In Chapter 4, the design and implementation of the vSwitch and Ryu based controller is detailed. In Chapter 5, a thorough evaluation of the implemented solution with a relevant discussion in presented. Chapter 6 includes a discussion on the conclusions and future work.