VirtualStack: SDN-controlled Transparent Protocol Transitions at the Edge

Jens Heuschkel
TK / TU Darmstadt,
Darmstadt, Germany
heuschkel@tk.tu-darmstadt.de

Michael Stein
TK / TU Darmstadt,
Darmstadt, Germany
stein@tk.tu-darmstadt.de

Max Mühlhäuser
TK / TU Darmstadt,
Darmstadt, Germany
max@tk.tu-darmstadt.de

***Abstract*—Today, we observe ossification of the Internet in terms of used protocols. Many network protocols exist, each being designed for outstanding performance for specific applications or network conditions. Nevertheless, for compatibility reasons, most applications use a conventional TCP/IP stack. This results in suboptimal link utilization, causing high costs for service providers and bad user experience. To overcome this problem, we propose VirtualStack, a novel approach for network protocol virtualization. VirtualStack enables the use of special-tailored protocols and reacts to changing network conditions by conducting transitions between these protocols. In this paper, we demonstrate the concept of VirtualStack in an SDN scenario. Our implementation of VirtualStack is transparent for applications and the operating system. We use SDN techniques to control VirtualStack. We show the feasibility of building multiple stacks per network flow, switching between these stacks, and enabling operation of multiple coexisting network stacks to realize multipath operation.**

I. Introduction

The Internet of today suffers from ossification as result of "optimizing" application layer middleboxes (e.g., NATs, proxies, firewalls, etc.). Typically, middleboxes rely on higher-layer OSI protocols to do their job, and most of them expect TCP [3]. For example, as we observe for the migration from IPv4 to IPv6 and as shown by Fonseca et al. [5], the OSI layer 2 is already completely ossified. Also, most application developers are unaware of the network conditions their applications will be used in. Without respecting the network conditions, to ensure reliable operation, developers have to use the most compatible and resilient protocol they can get. As a direct consequence, most application developers use the TCP/IP stack to realize their application communication [9]. This ossification results in suboptimal link utilization, causing high costs for service providers and bad user experience [10].

Despite the dominance of the TCP/IP stack on the Internet, thousands of alternative network protocols exist. And there are good reasons for this diversity: Most network protocols are tailored for specific applications or network conditions. Under specific constraints, these protocols deliver better performance, link utilization, latency, reliability, or routing quality than a standard TCP/IP stack.

Obviously, it is unreasonable to use a special tailored protocol for a wide range of application scenarios. Additionally, networks are subject to rapid conditional changes, increasing the problem of choosing a specialized, high performance protocol at runtime.

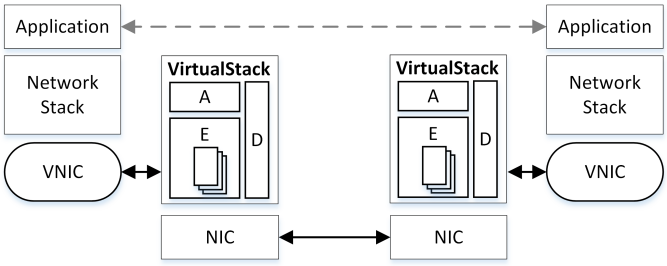

Fig. 1: Abstract Operation Overview: VirtualStack uses a virtual network interface (VNIC) to redirect traffic from applications.

To leverage the diversity of network protocols, a naive approach would be that developers integrate different protocols in their programs and provide the corresponding logic to choose the right set of protocols based on the current application requirements and network conditions. Since most application developers are no network experts, this approach is unlikely to work out in practice.

Even the IETF, which consists of conservative network experts in principle, argues in its draft "Transport Services (TAPS)" [4] that an adaptive mechanism for communication services is needed. But TAPS only considers the transport layer services and not the underlying layers, which handicaps cross-layer optimization.

In [7] and [8], we present VirtualStack (VS), a protocol virtualization and transition environment for the current and the future Internet. VS provides multiple virtualized network stacks for each recognized network flow. It is possible to switch between these stacks to enable an adaptive network protocol behavior. As shown in Figure 1, VS is transparent for the application because it uses a standard virtual network interface (VNIC) to redirect the traffic (like VPN software does) and provides an endpoint that handles the connection as expected by the application.

With the possibility to schedule packets on different stacks, VS allows for conducting very fast transitions between network protocolos. Also, VS realizes multipath communication by binding different stacks to different network devices.

Additionally, VS provides an SDN interface for providing and receiving monitoring data, receiving commands and for installing ECA rules [6] for autonomous optimizations. For network providers, this enables an end to end optimization of the complete network, including the core and the edge network and all layers of the OSI model.

This paper demonstrates transitions between network mechanisms and coexistence of multiple network stacks for one network flow. Switching between TCP/IP and UDP/IP stacks, we show transitions on the transport layer (illustrated in Figure 2). Furthermore, we show transitions on the physical layer by switching between stacks that are bound to distinct network interfaces. This leads to four stack configurations, one for each protocol on each channel. The user may trigger the transitions by sending commands to the SDN interface of VS over a provided user interface.

II. Considered Transitions

As outlined above, this demonstration considers transitions on two layers: transitions on the transport layer and on the physical layer. We envision transitions on further layers, e.g., transitions between application-specific topology control algorithms [**?**] on top of the MAC layer. A transition is always a switch between two stacks with distinct configurations. For each target configuration, a stack is built first. The old stack configuration is held for switching back quickly if needed.

A. Transport Layer Transitions

Transport layer protocols have a big influence on the performance of the network connection. Many researchers are developing novel protocols for improving the network performance (e.g., DCTCP [1], QUIC [2], etc.). Most of these protocols are designed for a special purpose, where the performance is optimal. In this demonstration, we show transitions between the two standard transport protocols, UDP and TCP. One out of many possible reasons for conducting a transition from UDP to TCP could be that the packet loss exceeds the maximum packet loss the application can handle. On the other hand, a transition from TCP to UDP can be motivated by a cross layer optimization when there is already a reliability mechanism on the MAC layer. This demonstration focuses on these two specific transport layer protocols. However, the concept is applicable to other transport layer protocols and allows for always activating the best performing protocol out of the wide range of existing protocols for the current network conditions.

B. Physical Layer Transitions

As demonstration of a physical layer transition, we show a switch between two network interfaces. In our demonstration setup, this leads to a transition between the two Ethernet channels. In Linux operating systems, the network interface is tightly coupled to a configuration of the interface. It is also possible to define multiple configurations of one hardware interface as different accessible network interfaces. With this procedure, we conduct transitions between different hardware configurations.

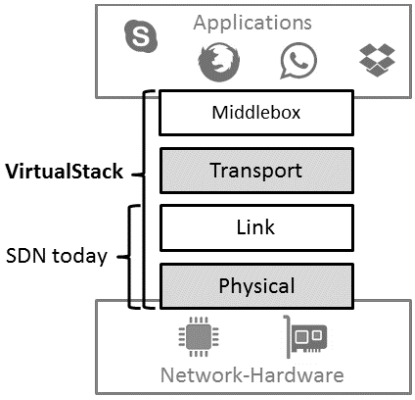

Fig. 2: Illustration of OSI layers with the scope of SDN today and VirtualStack. Boxes filled in grey are considered layers for the presented demonstration.

Recently, multipath connections have gained much attention [11]. VS is able to activate multiple stacks. By binding these stacks to different network interfaces, VS builds a multipath connection. This enables multiple transport protocols for a multipath connection because VS is able to choose a different protocol stack on every channel. This feature is also shown in this demonstration.

III. Demonstration Application

To demonstrate the VS concepts, we developed two special applications:

- The *visualization* (Figure 3) shows the incoming traffic reaching VS for each of the four stack configurations. Additionally, it shows a history of the throughput for each stack configuration.
- The *SDN User Interface* provides a southbound interface for VS and a command line interface for users to enable or disable specific stack configurations.

This course of the demonstration contains the following five main steps:

1) VS boots up, connects itself to the SDN User Interface and establishes the configured stacks: a UDP/IP and a TCP/IP stack for each of the two Ethernet channels. The initial configuration is the UDP/IP stack on channel 1 since this is the application configuration.
2) We show a transport layer transition through sending the commands for activating the TCP/IP stack and deactivating the UDP/IP stack on channel 1. VS will execute the commands immediately, and the visualization reflects that.
3) We show a physical layer transition through sending the commands for activating the TCP/IP stack on channel 2 and deactivation the TCP/IP stack on channel 1. Again, VS will execute the commands immediately, and the visualization reflects that.
4) We show the coexistence of multiple stacks to enable multipath connections with different transport layer protocols per channel. This is done by sending the command

to activate the UDP/IP stack on channel 1. VS will execute the commands immediately and the visualization reflects that.

5) We give the control to the audience to perform random commands and play around with the different stack configurations.

One full demonstration cycle can be shown within no more than 15 minutes.

IV. Demonstration Requirements

The demonstration is executed on at least one standard PC or laptop with a screen for displaying the stack visualization. VirtualStack runs on a Linux operating system (preferred Ubuntu 14.04). The visualization was developed for the Windows operating system (Windows 7 or newer). The technical equipment is to be placed on a table.

The two operating systems can be provided by virtual machines on the same computer. However, the demonstration is more convincing when the two components (VirtualStack and the visualization) are running on two separate machines and are connected with two separate physical links. In this case, a second machine (no screen required) with a Linux operating system (preferred Ubuntu 14.04) and two network cards for each machine are required (e.g., USB network cards if not enough internal cards are present). The complete setup is illustrated in Figure 4.

In summary, the demonstration is placed on a table to be provided at the conference venue together with an external power supply. A external presentation screen is optional equipment, which should be provided at the conference for an optimal presentation. The rest of the equipment described above will be provided by the authors. After setting up the demonstration, we do not require an Internet connection.

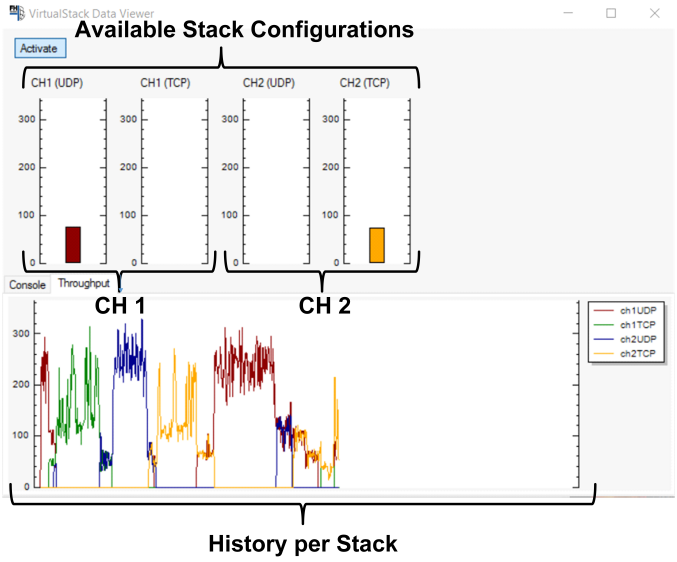

Fig. 3: Visualization Application: Shows the incoming traffic of the respective stacks. We have two available stacks per channel (UDP and TCP).

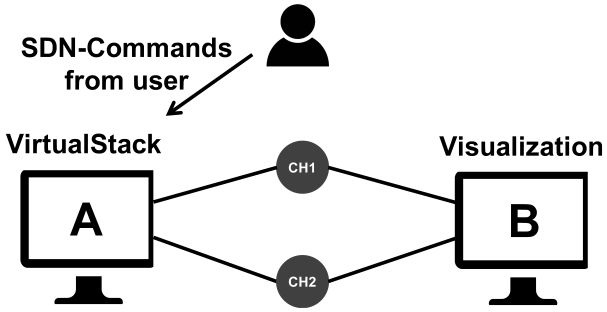

Fig. 4: Hardware Setup: VirtualStack is running on host A together with a packet generating application. The visualization is running on host B, showing the incoming traffic on the different channels with respective stack configurations. The user is able to send SDN commands to VirtualStack to change the behavior of VirtualStack.

Acknowledgment

This work has been funded by the German Research Foundation (DFG) as part of the projects A01 and B02 within the Collaborative Research Center (CRC) 1053 – MAKI.

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proceedings of ACM Special Interest Group on Data Communication*, 2010, pp. 63–74.

[2] G. Carlucci, L. De Cicco, and S. Mascolo, "HTTP over UDP: an experimental investigation of QUIC," in *Proceedings of ACM Symposium on Applied Computing*, 2015, pp. 609–614.

[3] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet, "Revealing middlebox interference with tracebox," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 1–8.

[4] G. Fairhurst, B. Trammell, and M. Kuehlewind, "Services provided by IETF transport protocols and congestion control mechanisms," Internet-draft draft-ietf-taps-transports-07, IETF, Tech. Rep., 2015.

[5] R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica, "IP options are not an option," University of California at Berkeley, Tech. Rep. UCB/EECS-2005-24, 2005.

[6] A. Frömmgen, R. Rehner, M. Lehn, and A. Buchmann, "Fossa: Using genetic programming to learn ECA rules for adaptive networking applications," in *Proceedings of 40th IEEE Local Computer Networks Conference*, 2015, pp. 197–200.

[7] J. Heuschkel, A. Frömmgen, J. Crowcroft, and M. Mühlhäuser, "VirtualStack: adaptive multipath support through protocol stack virtualization," in *Proceedings of the 11th International Network Conference*, 2016, pp. 1–6.

[8] J. Heuschkel, I. Schweizer, and M. Mühlhäuser, "VirtualStack: a framework for protocol stack virtualization at the edge," in *Proceedings of 40th IEEE Local Computer Networks Conference*, 2015, pp. 595–598.

[9] W. John and S. Tafvelin, "Analysis of internet backbone traffic and header anomalies observed," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007, pp. 111–116.

[10] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[11] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? Designing and implementing a deployable multipath TCP," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012, pp. 29–29.