Experiences Deploying a Transparent Split TCP Middlebox and the Implications for NFV

Franck Le, Erich Nahum, Vasilis Pappas, Maroun Touma, Dinesh Verma
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
{fle|nahum|vpappas|maroun|dverma}@us.ibm.com

ABSTRACT

This paper summarizes our experiences deploying a transparent Split TCP middlebox for WiFi networks in Enterprise customer environments. Since Split TCP is nearly two decades old, we believed this would be a straightforward application of well-known technology. Reality, however, would teach us otherwise.

While we began our deployment in our own office with 3,000 users, we encountered several challenges in deploying this technology at customer sites. Each customer had different network architectures, security policies, and non-negotiable requirements. In particular, modifying the network architecture was frequently impossible. Deployment challenges tended to fall into two related but distinct categories. First, making the box transparent to both clients and servers required extending the notion of transparency from beyond just layer 3 and layer 4 to include layer 2. Second, the interaction of our middlebox with other middleboxes resulted in unexpected behaviors.

Our deployments supported up to 15,000 simultaneous users and lasted up to 2 years. We offer up our experiences so that others need not repeat them. We discuss some implications of our experiences on deploying network functionality in virtual environments, or Network Function Virtualization (NFV). If NFV is to be successful in real environments, these challenges will need to be overcome.

1. A TRANSPARENT SPLIT TCP PROXY

Wireless networks such as WiFi, Cellular, and BlueTooth have become ubiquitous. Yet frustrations remain with wireless network performance. Fading, attenuation, poor connectivity, and range limitations all contribute to poor wireless experience.

HotMiddlebox'15, August 17-21 2015, London, United Kingdom

 ISBN 978-1-4503-3540-9/15/08…$15.00

DOI: http://dx.doi.org/10.1145/2785989.2785991

Split TCP: One solution to this problem is Split TCP [1, 6, 25]. The concept is simple: Split a TCP connection between a fixed server and wireless client into two separate connections at the middlebox – one connection between the server and the middlebox, and one between the middlebox and the client. This allows packet loss recovery to occur independently over either leg, minimizing unnecessary congestion window reductions and improving TCP performance.

Routing TCP Streams: Of course, this requires that the middlebox be able to intercept all packets of a flow in both directions. Many network functions (e.g., firewall, intrusion detection system, deep packet inspection, split TCP proxy) have the requirement that they observe and even intercept full TCP streams. This can be achieved in three ways: First, clients can be explicitly configured to direct traffic to the middlebox (e.g., SOCKS, HTTP Proxy). Second, the middlebox can be placed in such a way as to force all traffic through it (at a "choke point"). Finally, the network can be configured to forward, route, or tunnel packets to the middlebox. Each approach has advantages and disadvantages.

The first approach was not acceptable, in neither our own IT environment nor any customer environment we encountered. Managing and administering every client to use the middlebox can be difficult, tedious, and frequently impossible. The second approach presents single point of failure problems, as we discuss in Section 2.2. The third approach requires modifying the routing infrastructure, which at many customers is considered untouchable.

Transparency: This brings us to the issue of *network transparency*. Transparency has two sides: transparency for the client, which does not realize it is traversing a middlebox, and transparency for the server, which sees requests as if originating from the client IP address. Client transparency eliminates the need for explicit configuration on the client. While server transparency might not be desired (e.g., using a NAT box to hide client IPs to the external world), it also eases deployment for many reasons. For example, a NAT box

can complicate the processing by subsequent middleboxes [21], including the identification of compromised clients by intrusion detection systems, the application of traffic shaping to different clients, and the analysis and characterization of user behaviors. Also, many middleboxes take advantage of network transparency (e.g., firewalls, IDS) for both functionality and security reasons. For example, a middlebox may not wish to expose its own IP address to reduce the risk of attacks.

For these reasons, we wished to utilize full network transparency. Linux introduced a mechanism for this in the 2.6 kernel called *transparent proxying* [14] (an earlier version had been in version 2.4 but had been removed.) Transparent proxying allows the middlebox to receive and send traffic using *any* IP address, thus masquerading as the server to the client, and as the client to the server. This support was originally intended for transparent Web proxies such as Squid [24], but we realized it could be used for Split TCP as well, making it much more viable for deployment. Transparency also includes the TCP layer, namely, the middlebox uses the client's source TCP port when connecting to the server. However, transparency does not extend to the MAC layer, as we discuss in Sections 2.6 and 2.7.

History: From 2011 to 2014, we developed, deployed, and evaluated a Split TCP middlebox. The middlebox was deployed at roughly 10 sites, ranging from hundreds to tens of thousands of users, and lasting up to 2 years. After developing the middlebox implementation in the lab on a dedicated testbed, we believed we were ready for general deployment. However, we had not anticipated nor considered some of the more subtle interactions that occurred in deploying a transparent middlebox across multiple environments. In this paper, we describe some of those interactions and, where appropriate, discuss their implications in the context of moving network functionality into the cloud, frequently called Network Function Virtualization (NFV) [11]. In essence, these issues are all related, centering on deploying a *transparent* middlebox and how it interacts with *other* middleboxes.

Contributions: Middleboxes are commonly deployed in operational networks (e.g., [23]), and one might argue that network administrators and middlebox vendors must have therefore encountered the issues we describe. However, although some of the problems may have been previously reported, the information is widely scattered across online forums. More importantly, solutions and hacks are often suggested without fully explaining the reasons why they solve the problems (e.g., [22]). As such, it is difficult to reason about their limitations, and even understand if the solutions actually work. For example, commercial middleboxes have recognized issues (e.g., preservation of VLAN tags), and claim to have integrated features to address them, but do not describe the solutions [2, 4]. In contrast, we describe the issues in detail, present the solutions we implemented, and explain the limitations of our approaches. By exposing the issues and the associated challenges, we hope that the networking community will be able to develop more general solutions to facilitate the deployment of middleboxes and virtual network functions.

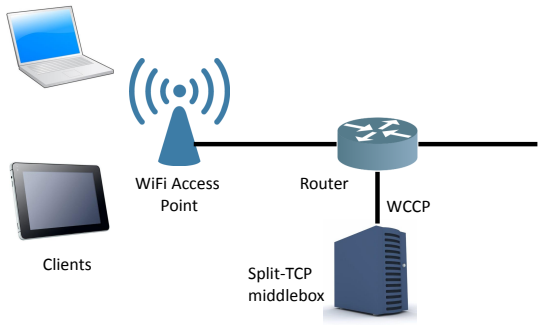

Figure 1: Sideband Mode.

2. DEPLOYMENT EXPERIENCES

In this section we structure our discussion as follows. For each issue we encountered, we describe the problem, how we overcame it, and, where appropriate, potential implications for deploying in the cloud.

2.1 Transparency and Network Topology

Problem: When we initially designed the Split TCP middlebox, it was deployed at IBM Research connected to a router as depicted in Figure 1, based on the requirements of our own IT staff. The router utilized Cisco's Web Cache Communication Protocol (WCCP) V2 [10]. to forward a subset of packets (in this case, from subnets corresponding to WiFi APs) to the middlebox. WCCP was originally developed for routers to redirect clients' requests to a local Web cache, but works equally well for a transparent split TCP proxy.

However, many customer network designs relied solely on layer 2 networks, with the router managed by their service provider and inaccessible to both us and the customer. Thus, the initial deployment solution was not feasible.

Solution: To handle layer 2 environments, we extended the Split TCP middlebox to be deployed in an *inband mode* as illustrated in Figure 2, and began to think of the deployment model in Figure 1 as *sideband mode*. A Linux bridge is created at the middlebox and both network cards are added to it. With all traffic traversing the bridge, `ebtables` allows packets to be intercepted.

Implications: While this particular problem was not difficult to solve, it was the beginning of a series of related problems having to do with *network transparency*. Transparent network functions offer unique challenges to NFV, since transparency must be defined in terms of each networking layer.

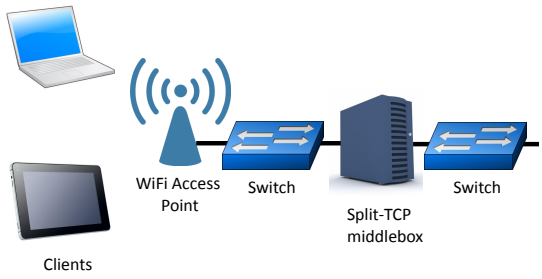

Figure 2: Inband Mode.

2.2 Failover

Problem: A central customer requirement was that the middlebox would fail gracefully. Particularly for a performance enhancing proxy such as Split TCP, which processes all data packets and even terminates TCP connections, a failure essentially "blackholes" the network. Customers required "fail-to-wire", where a failure results in the system returning to the configuration without the middlebox.

Solution: As discussed in Section 2.1, there are two available approaches: inband and sideband. The inband base, while conceptually simpler, is actually more difficult to manage from a failure perspective. This is because there is no "routing around the problem" when the middlebox fails. Instead, we used a hardware device installed on a PCI slot triggered by a software heartbeat from the middlebox. If the middlebox does not provide the heartbeat after some period of time, the hardware physically converts the Ethernet ports so that traffic simply passes through the middlebox. This works for both hardware and software failures.

The sideband case, while at first glance more complex, was actually easier to solve with WCCP. A router using the WCCP protocol communicates with a middlebox via periodic UDP heartbeat messages. If the middlebox does not respond within a configurable number of seconds, the router stops forwarding packets and instead performs default routing.

Implications: In the effort to virtualize network functionality, failover must be considered and planned for. VMs will eventually fail and need to be handled gracefully. Our deployments never required more than one machine, but of course, VNFs may handle much more significant load and require larger resources of CPU, memory, etc. Failover in NFV thus has *two* aspects: First, a single instance of a service may fail, while others continue, and thus software coordinating the scaling (e.g., load balancer) will need to redirect traffic to active instances. Second, the service *as a whole* may fail and require that some action may be taken, depending on the service. For example, policy might dictate that if a firewall fails, the network gets disconnected, whereas for a Web cache, the cache is simply bypassed.

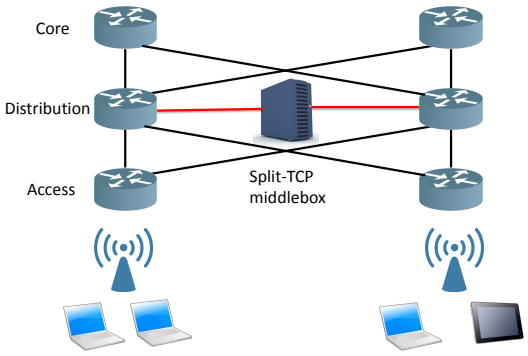

Figure 3: Asymmetric Routing.

We believe that this requirement may even become more critical as network functions are migrated to the cloud. This is because intermediate ISPs may experience failures, and cloud service provider infrastructures may require unplanned maintenance, increasing the risks for network functions to be unreachable.

One study [23] suggests DNS as a possibility to redirect traffic to network functions on the cloud. However, updating DNS entries can be slow, and such approach may therefore not be adequate to address network functions' failures. Indeed, early approaches to Web server load balancing and availability examined using DNS, [18], but found it ineffective at short-term time scales.

Two problems encountered above also make deploying NFV more difficult. In the inband case, each hardware vendor had its own API for managing and triggering failover. This is, of course, a headache for developers, who must write custom code for each vendor. Far better would be to have a standard API, supported by vendors, for managing failover. Similarly, in the sideband case, WCCP v2 is a Cisco-developed protocol. Although made available to competitors such as Blue Coat and Riverbed, an IETF standardized failover protocol that can be utilized by all network functions, virtualized or not, would be preferable.

2.3 Asymmetric Routing

Problem: Enterprise networks are frequently provisioned to be fault-tolerant and robust to failures by providing multiple paths in their networks. Many networks follow the industry standard Cisco Enterprise Network Architecture [8], with each access router being connected to two distribution routers, as shown in Figure 3. Since both paths are active, asymmetric routing can occur where upstream packets traverse one distribution router and downstream packets may traverse another. In particular, in our enterprise, we observed data packets traversing one path and ACK packets traversing the other. This was a problem as the Split TCP middlebox could not see the full TCP stream. The middlebox could not be placed closer to the APs, since the APs used Cisco's integrated controller and traffic between the controllers and the APs is encrypted [7].

Solution: We leveraged the WCCP protocol described in Section 2.2 to handle routing asymmetry. Both distribution routers were configured to forward all WiFi traffic to the middlebox, ensuring the full TCP stream traverse the middlebox.

Implications: Many network functions require full TCP streams, and routing asymmetry can break this. NFV will need to preserve this property in the cloud. While SDN can help with this issue, wide-area cases will be more difficult, since the NFV provider will not have control over all traversed networks.

2.4 Tunneling

Problem: In *sideband* mode, packets must be redirected from the router to the middlebox. To achieve this, two main methods exist: WCCP (Layer-2) L2, and WCCP GRE. WCCP L2 overwrites the original destination MAC address of the IP packet with that of the middlebox, instead of that of the next L2 hop. This approach requires the middlebox to be accessible on a neighboring LAN of the router. In some configurations, we ran dedicated Ethernet cables from the router(s) to the middlebox so that we could use L2 redirection. The second method, WCCP GRE, forwards packets through a GRE tunnel between the router(s) and the middlebox. It allows packets to reach the middlebox even when it is multiple routers away. However, GRE requires a minimum of 8 bytes for the GRE header, which effectively reduces the available MTU of the packet. The tunnel is not visible to the client nor server at the TCP layer. Thus, connections would be established, but the moment a full-size MTU packet was transmitted (typically in a server HTTP response), the tunnel would drop it. A connection would die shortly after it had begun.

Solution: After some investigation, the cause was identified: The "don't fragment" (DF) flag in the IP header was set, causing the packet to be dropped at the tunneling router as the size exceeds the maximum size of the outgoing interface and the router can't fragment the packet. The router should instruct the server to send smaller packets by sending an ICMP type 3 packet (Destination Unreachable; Fragmentation Needed and DF set.) However, routers are often configured to not send ICMP destination unreachable messages, or firewalls may discard them [15]. To address the issue, we updated the MTU of the middlebox interface(s) to account for the GRE header.

Implications: Tunneling will be an important component of NFV to route packets through the cloud. Several encapsulation techniques (e.g., VXLAN [16], Overlay Transport Virtualization [9]) have recently been developed to extend layer 2 networks across data centers. However, as described above, tunneling can create issues with MTU which may ultimately disrupt connectivity. MTU problems are subtle to diagnose since they allow partial progress for a connection. Although we fixed the problem by adjusting the MTU of the middlebox interface(s), this solution is not general: In particular, the size of the outer headers may vary. For example, VXLAN can encapsulate both VLAN and untagged Ethernet frames, which have different lengths. GRE and IPsec also have different overheads, and IKE may decide to tunnel IPsec packets over UDP in the presence of NAT(s) [13]. As such, fixing the MTU of a middlebox interface may not be optimal for all the possible cases. More preferable would be a solution that dynamically detect MTU issues. One possibility could the IETF standardized Packetization Layer Path MTU Discovery (PLPMTUD) procedure [17]. It does not rely on ICMP messages, but probes with progressively larger packets at the transport layer. However, one study [15] has highlighted the challenge in determining whether a segment is lost due to congestion or MTU issues.

2.5 Network Address Translation

Problem: In one deployment setting, a customer required that a NAT be present between the wireless clients and the Split TCP middlebox. We observed a number of TCP connection attempt failures, where a client sent a TCP SYN, and the Split TCP middlebox properly received it (as observed by tcpdump). However, the middlebox neither sends a TCP SYN to the server, nor replies with a TCP SYN-ACK to the client. Instead, the client's request times out, and the TCP connection request fails. Online forums have reported similar behaviors [22].

Solution: Online forums reported that disabling TCP timestamps fixed the issue, but did not provide any explanation why this worked [22]. It turns out that the Linux TCP stack includes some mechanisms to prevent delayed segments from one connection from being accepted by a later connection re-using the same 4-tuple (source and destination ports, source and destination addresses). This is the purpose of the TIME-WAIT state in TCP [20]. For a given period of time, the Linux TCP stack compares the timestamp of the newly received packet with that of the last packet from the previous connection from the same IP address. If the newly received timestamp is smaller, the packet is silently discarded.

In the presence of a NAT, all hosts except one may consequently suffer connections failures, since the middlebox perceives them all as coming from a single IP address. Further details on the issue are described in [3], which also presents a more direct solution: Disable the behavior on the middlebox by setting `net.ipv4.tcp_tw_recycle` to 0.

Implications: As middleboxes are migrated to the cloud, the risks for unexpected interactions with other middleboxes and the resulting errors may grow. This is

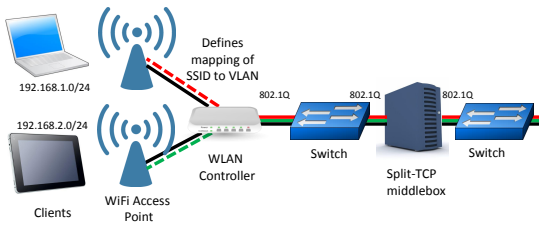

Figure 4: Middlebox in a VLAN Environment.

because the number of traversed middleboxes may increase, as cloud service providers often deploy different network functions including load balancers to alleviate network congestion, or firewalls to protect their infrastructure. In addition, different vendor implementations may present slightly different behaviors, making it difficult to predict or test potential failures.

2.6 Security Gateway

Problem: In another setting, customers utilized a security gateway to prevent packet spoofing. The gateway is both a DHCP server assigning IP addresses and a security appliance that verifies that arriving packets with a given source IP address have the same client hardware MAC address that was assigned by DHCP. If the verification fails, the gateway silently drops the packet.

In the presence of the Split TCP middlebox, packets reaching the security gateway have their source IP address preserved to that of the originating client. However, the sender MAC address is that of the middlebox. This is because the middlebox terminates the TCP connection from the client, and originates a new one with the server, masquerading as the client. As a result, the verification between the source IP address and the sender MAC address fails, and packets from the middlebox (e.g., TCP SYN) are dropped[1].

Solution: We developed a module on the middlebox to learn the mapping between the client IP and MAC addresses by listening to ARP messages. The module overwrites outgoing packets, replacing the source MAC address of the middlebox with the appropriate MAC address of the client. However, while the main functionality appears simple, the module has to ensure race conditions do not still cause connection failures, and clients do not experience abrupt disconnections if the daemon process dies.

Implications: This reinforces the issue from Section 2.5, that problems may increase from unexpected interactions between middleboxes as more of them are introduced. We could not reproduce the problem in our local lab, as the gateway was not available to us for testing.

[1]In one sense this is proper behavior, as the middlebox *is* spoofing the client.

2.7 VLAN

Problem: A number of customers rely on VLANs and required us to deploy the Split TCP middlebox on a trunk link. Figure 4 illustrates such a scenario: The VLAN controller defines the mapping of the WiFi SSID to VLAN. The middlebox is deployed in the *inline* mode (Section 2.1), and uses a Linux bridge and `ebtable` rules to intercept the traffic. However, outgoing packets from the middlebox lacked the proper VLAN tags and could not be properly forwarded to the corresponding gateway router(s). The middlebox therefore broke the connectivity of the wireless clients.

Solution: An additional module was developed and added to the middlebox to perform VLAN tagging of the outgoing packets. To properly tag the packets, the module must be configured with the VLAN IDs, the IP address ranges belonging to each VLAN, and the IP address of the default gateway router for each VLAN. With such information, given the source IP address of a packet, the middlebox can identify the proper VLAN tag to attach to the packet. However, this solution is limited in that it does not allow the same subnet (e.g., 192.168.1.0/24) to be used by multiple VLANs. This scenario did not occur in any of the environments we deployed the middlebox; but because of this restriction, the solution may not be applicable to all environments. Commercial middleboxes can preserve VLAN tags [2, 4]. However, their solutions are not described in detail and thus, their features and limitations (e.g., support for a same subnet across multiple VLANs) are unclear.

Implications: Virtual Extensible LAN (VXLAN) [16] enables clients, switches and virtualized network functions deployed on the cloud to belong to the same layer 2 network/VLAN. However, deploying a network function in a VLAN environment creates challenges to correctly tag outgoing packets. At the same time, this presents an *opportunity* for NFV for customers to simplify their networks by separating the functionality they require (e.g., security, traffic isolation) from how it is currently implemented (e.g., VLANs).

3. DISCUSSION

Certain types of network functions are sensitive to RTT in a way that others are not, particularly performance enhancing proxies (PEPs) [5] such as Split TCP. Thus, their *placement* in the network topology is fundamental to their success. Split TCP, for example, is most effective when it is adjacent to the wireless hop. A Web cache needs to be close to the client to effectively reduce response times.

For these types of latency-sensitive network functions, some have suggested cloud providers with a "CDN-like footprint" [23]. However, it is not clear how likely such cloud providers will be. More likely, we believe, is that these functions will be hosted in a local private cloud.

Similarly, introducing virtualization and wide-area delays will increase the *variability* of the service provided by the network function, due to issues such as network congestion, network failures or oversubscribed clouds.

Finally, network functions may be exposed to unusual threats from their co-tenants in a cloud. This may require further hardening of services, requiring more stringent or additional firewall protection, adding to the cost and delay in NFV.

4. CONCLUSIONS AND FUTURE WORK

This paper has focused on the *deployment* issues encountered as part of designing, deploying, and evaluating a transparent Split TCP middlebox for wireless networks. There are other issues we encountered but, due to space limitations, cannot include them here. Broadly, they fall under two categories:

Debugging. These issues centered on development decisions that aided or hindered debugging running systems, particularly under high load, or after extended deployments (i.e., months).

Measurement. While it was simple to measure performance improvements in a dedicated testbed, it was much more difficult to quantify to customers how well the system worked for *their* traffic and environment. In particular, *variability* introduced by using real workloads, networks and servers made this much more challenging. Constraints required by the customers also restricted our actions.

We are currently writing up our entire experience including the above in longer form and hope to publish that as a full paper.

Acknowledgements

We would like to thank Hani Jamjoom and the anonymous reviewers for their helpful comments. This research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement No. W911NF-06-3-0001. Vasilis Pappas is currently at Google. His contribution to this work was performed while at IBM.

5. REFERENCES

[1] A. Bakre and B. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *IEEE ICDCS*, 1995.

[2] Baracuda. VLAN deployments. https://techlib.barracuda.com/bwf/deplyvlan.

[3] V. Bernat. Coping with the TCP TIME-WAIT state on busy linux servers. http://vincent.bernat.im/en/blog/2014-tcp-time-wait-state-linux.html.

[4] Blue Coat. Technology primer: VLAN tagging. https://www.bluecoat.com/sites/default/files/documents/files/VLAN_Tagging.1.pdf.

[5] J. Border, M. Kojo, J. Griner, G. Montero, and Z. Shelby. Performance enhancing proxies intended to mitigate link-related degradations. *IETF RFC 3135*, 2001.

[6] K. Brown and S. Singh. M-TCP: TCP for mobile cellular networks. *ACM SIGCOMM CCR*, 27(5):19–43, 1997.

[7] Cisco Systems. Cisco 8500 series wireless controllers. http://www.cisco.com/c/en/us/products/wireless/8500-series-wireless-controllers/index.html.

[8] Cisco Systems. Enterprise network architecture v. 3. http://www.cisco.com/web/strategy/education/us_education/borderless_education.html.

[9] Cisco Systems. Overlay Transport Virtualization (OTV). http://www.cisco.com/c/en/us/solutions/data-center-virtualization/overlay-transport-virtualization-otv/index.html.

[10] Cisco Systems. WCCP version 2. http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipapp/configuration/xe-3sg/iap-xe-3sg-book/iap-wccp-v2.html.

[11] ETSI. Network functions virtualization: Architectural framework. http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf, 2013.

[12] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In *Proc. of ACM IMC*, 2011.

[13] T. Kivinen, B. Swander, A. Huttunen, and V. Volpe. Negotiation of NAT-Traversal in the IKE. *IETF RFC 3947*, 2005.

[14] Linux Kernel Sources. tproxy. http://www.kernel.org/doc/Documentation/networking/tproxy.txt.

[15] M. Luckie and B. Stasiewicz. Measuring path MTU discovery behaviour. In *Proc. of ACM IMC*, 2010.

[16] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. VXLAN: A framework for overlaying virtualized layer 2 networks over layer 3 networks. *IETF RFC 7348*, 2014.

[17] M. Mathis and J. Heffner. Packetization layer path MTU discovery. *IETF RFC 4821*, 2007.

[18] J. Mogul. Network behavior of a busy Web server and its clients. Technical Report 95/5, DEC WRL, Oct. 1995.

[19] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[20] J. Postel. Transmission control protocol. *IETF RFC 793*, 1981.

[21] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 2012 USENIX NSDI*.

[22] Serverfault. Why would a server not send a SYN/ACK packet in response to a SYN packet. http://serverfault.com/questions/235965/why-would-a-server-not-send-a-syn-ack-packet-in-response-to-a-syn-packet. Accessed: 2015-04-01.

[23] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012*.

[24] Squid Wiki. SquidFAQ interception proxy. http://wiki.squid-cache.org/SquidFaq/InterceptionProxy.

[25] R. Yavatkar and N. Bhagawat. Improving end-to-end performance of TCP over mobile internetworks. In *WMCSA*, pages 146–152. IEEE, 1994.