Flow-level Load Balancing of HTTP Traffic using OpenFlow

Anees Al-Najjar
School of ITEE, The University of Queensland, Brisbane, Australia

Siamak Layeghy
School of ITEE, The University of Queensland, Brisbane, Australia

Marius Portmann
School of ITEE, The University of Queensland, Brisbane, Australia

Jadwiga Indulska
School of ITEE, The University of Queensland, Brisbane, Australia

Abstract: In this paper, we explore the concept of flow-based load balancing of network traffic on multi-homed hosts. In contrast to existing approaches such as MultipathTCP, our approach is a client-side-only solution, and can therefore easily be deployed. We specifically explore flow-based load balancing for web and video traffic use cases. Experimental evaluations of our OpenFlow-based load balancer demonstrate the potential of flow-based load balancing.

Keywords: Software Defined Networking, Load Balancing, Multi-homed Devices, Web Traffic Optimisation, Video Streaming Traffic

1. Introduction

Computing devices are increasingly equipped with multiple network interfaces, e.g. LTE and WiFi in the case of smartphones. Efficiently using multiple network interfaces on such multi-homed hosts is a challenging problem. Approaches such as Multipath TCP (MPTCP) (Ford, Raiciu, Handley & Bonaventure,2015) allow load balancing of traffic across multiple links and paths on a per-packet granularity. The problem with MPTCP is that it requires both ends, i.e. client and server, of the end-to-end path to support the protocol. Despite the many years since the introduction of MPTCP, its deployment and use are minimal with a few notable exceptions, such as Apple's Siri, as stated in https://support.apple.com/en-au/HT201373.

In contrast, we consider a client-side only approach to load balancing across multiple network interfaces, which does not require any special support from the server. In this approach, load

http://doi.org/10.18080/ajtde.v6n4.166

balancing at the level of granularity of packets is not possible, due to the fact that TCP connections are bound to IP addresses and hence host interfaces. Thus, we do not consider approaches such as Mobile IP (Perkins, 2002), Host Identity Protocol (HIP) (Moskowitz, Nikander, Jokela & Henderson, 2010) or Site Multihoming by IPv6 Intermediation (Shim6) (Abley, Black & Gill, 2003) here, due to their limited adoption. Instead, we consider a practical, flow-based load balancing approach, where the level of granularity for distributing network traffic is network flows, e.g. TCP connections. We discussed the basic idea of this approach and its preliminary implementation using Software Defined Networking and OpenFlow in Al-Najjar, Layeghy & Portmann (2016). Our initial evaluations in Al-Najjar, Layeghy & Portmann (2016) showed the potential and practicality of this approach. However, it was limited in regard to the considered network traffic (download of identical, fixed size files) as well as the considered network links with static link capacity.

In this paper, we investigate the potential of flow-based load balancing on multi-homed hosts in a realistic setting. We specifically focus on Web and video traffic, due to their predominance and relevance for overall quality of user experience.

The potential of flow-based load balancing depends on the characteristics of the network traffic, e.g. the number, size distribution, and level of concurrence of flows. In the extreme case, we could have a web page that is downloaded via a single TCP connection. In our approach, this flow would be allocated to a single interface, and there would be no potential gain for load balancing and using the other available network interface and corresponding path.

It is therefore important to understand the characteristic of Web traffic in regards to network flows. We have performed extensive measurements and analysis of the web traffic for HTTP(s)/TCP connections, based on the Alexa top 100 web pages (Alexa, n.d.). Our analysis shows that typical websites require a large number of flows (typically TCP connections), which shows there is a potential for flow-based load balancing to improve the download performance and user experience.

We also investigated controlling the HTTP traffic in SDN-based multi-homed devices over the QUIC (Quick UDP Internet Connection) protocol. QUIC is a relatively new transport-layer protocol specifically designed for web traffic (Roskind, 2013). Like TCP, QUIC is also connection-oriented. QUIC carries about 7% of the global Internet traffic and 30% of Google traffic (Langley *et al.*, 2017), and is becoming increasingly relevant.

In addition to web traffic, this paper also considers controlling the flow of video traffic. Dynamic Adaptive Streaming over HTTP (DASH) (ISO, 2014) traffic running over the QUIC protocol will be considered in our use case. Because DASH traffic is considered as a single TCP

http://doi.org/10.18080/ajtde.v6n4.166

or UDP flow and that flow is only allocated to a single network interface, it is not as amenable to flow-based load balancing as is web traffic. However, we consider the scenario of having video streams as background traffic, and investigate how this impacts on the efficiency of our SDN-based traffic load balancing approach for web traffic in multi-homed devices.

Our experimental evaluation of flow-based load balancing is based on an implementation using an OpenFlow Software Switch, Open vSwitch (OVS), and the Ryu SDN controller. For our experiments, we consider the realistic and practical scenario of a dual-home host, with both an LTE and a WiFi interface. We performed extensive measurements where we established the simultaneous and co-located link capacity of LTE and WiFi interfaces at our university campus. We then used these realistic link capacity measurements for our experiments, using link emulation.

Our results show that flow-based load balancing can significantly reduce the page load time, for the realistic and practical traffic and link scenario that we considered. Somewhat surprisingly, it even outperforms MPTCP.

The rest of this paper is organised as follows. Section 2 gives a brief background on the concept of SDN and OpenFlow, MPTCP and QUIC. Section 3 explains the idea of flow-based load balancing as well as our implementation. In Section 4, we present our analysis of web traffic and its potential for flow-based load balancing. Sections 5 and 6 present our experimental evaluation of flow-based load balancing, for two different link capacity scenarios. Finally, Section 7 discusses related works, and Section 8 concludes the paper.

2. Background

2.1. OpenFlow

Since our flow-based load balancer is implemented in SDN using OpenFlow, we provide a brief introduction to the relevant key concepts.

In Software Defined Networks (SDN), a key idea is the separation of control and data plane. The SDN architecture with its three layers (infrastructure, control and application) is shown in Figure 1. The logically centralised SDN controller configures the forwarding behaviour of forwarding elements (SDN switches) via a *southbound interface*.

http://doi.org/10.18080/ajtde.v6n4.166

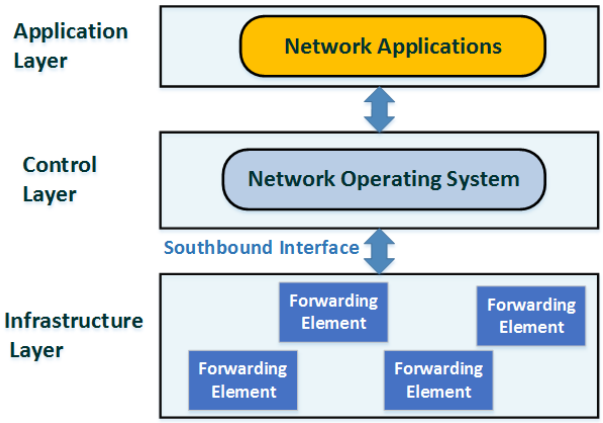

Figure 1. SDN Architecture

The OpenFlow protocol (McKeown *et al.*, 2008) proposed by Open Networking Foundation (available on https://www.opennetworking.org/) is the dominant SDN southbound interface. It allows the controller to install forwarding rules using a *match-action* paradigm. The rules can match on various L2-L4 header fields, including MAC and IP addresses, as well as the ingress port via which the packet was received.

OpenFlow supports different types of actions. The *output* action allows the switch to forward packets via a specific port. OpenFlow also supports a *set-field* action which allows rewriting of packet header fields. This is typically used for functions such as Network Address Translation (NAT).

The interaction between the SDN controller and switches occurs via OpenFlow messages. A switch can encapsulate and send a data packet to the controller via an OpenFlow *Packet-in* message. The controller can send a packet to the switch via a *Packet-out*, with instructions (a set of actions) on how to handle the packet. The controller also can install forwarding or flow rules on switches via OpenFlow *Flow-Mod* messages.

The OpenFlow protocol provides messages that allow querying statistics from switches in regard to links, ports and flows. *Port Stats* is one of these message groups. The controller requests statistics of active ports by sending a *PortStatsRequest* message. The switch replies with a *PortStatsReply* message, carrying a set of statistics related to each port, such as the cumulative number of sent and received packets and bytes, as well as the number of packets that have been dropped or had errors.

Flow Stats is another type of OpenFlow probing message type. It allows collecting statistics of the active flow entries (forwarding rules) in the switch. The controller requests this information via sending a *FlowStatsRequest* message, upon which the switch replies with a *FlowStatsReply* message. The message contains information related to each installed rule, for instance table_id, priority, number of bytes/packets that matched the rule, the active duration of the flow, and the match/action fields.

http://doi.org/10.18080/ajtde.v6n4.166

Our flow-based load balancer, which will be discussed in detail in this paper, was implemented using these basic OpenFlow primitives.

2.2. MPTCP

Multi-Path TCP (MPTCP) is one of the current approaches for sending traffic across multiple network interfaces and paths on multi-homed hosts (Ford *et al.*, 2015). We briefly explain MPTCP, since we will use it as a benchmark against our proposed approach. However, this is a somewhat unfair comparison, since MPTCP requires support on both ends of the communication path, which is a key reason for the very slow and minimal adoption of MPTCP. In contrast, our proposal is a client-side only solution, which makes deployment very easy.

MPTCP adds a layer between the Application and Transport layers in the TCP/IP protocol stack, as shown in Figure 2. It creates multiple TCP subflows that can be sent via multiple different network paths. As mentioned, MPTCP requires support from both connection sides (the client and the server). If the server does not support MPTCP, the protocol will fall back to basic TCP.

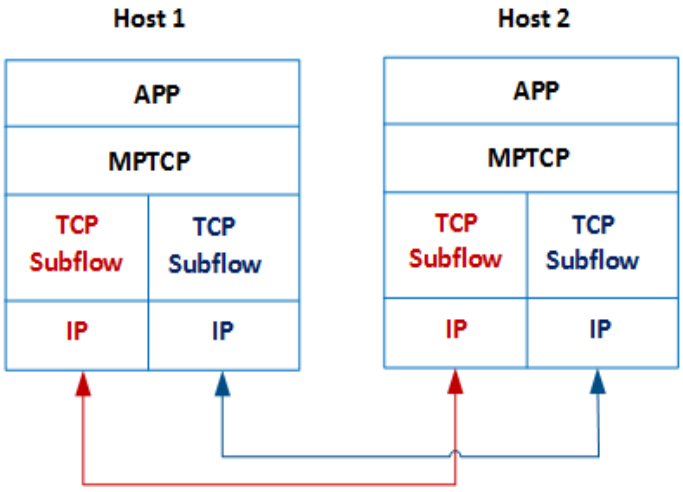

Figure 2. MPTCP Protocol

To establish an MPTCP connection, a host uses the normal TCP handshaking packets represented by SYN, SYN/ACK, and ACK with an additional option. This **MP_CAPABLE** option allows checking if both ends support MPTCP and, if not, the connection falls back to a normal TCP connection. In the case where MPTCP is supported by both client and server, a 64-bit authentication key is generated and exchanged. The keys are required in the next stages for creating and authenticating TCP subflows. Once both ends confirm supporting MPTCP, and authentication keys have been exchanged, a new TCP subflow can be initiated. Each MPTCP subflow also uses the same TCP handshaking packets with an MP_JOIN option. The option contains a number of flags and the address ID of the corresponding host.

http://doi.org/10.18080/ajtde.v6n4.166

MPTCP allocates network traffic among multiple network interfaces at the level of granularity of TCP segments. This is in contrast to our approach, where the level of granularity is limited to flows. As a result, one would expect MPTCP to outperform our flow-based approach. Based on our experiments, this is not the case. This can be explained by limitations of MPTCP that have been identified in previous work (Chen *et al.*, 2013).

2.3. QUIC Protocol

The QUIC protocol has been proposed by Google in order to overcome some of the limitations of TCP, specifically when used in conjunction with HTTP traffic (Langley *et al.*, 2017).

QUIC runs on top of UDP, making it easy to be deployed and updated. Figure 3 shows the architecture of HTTP2 over QUIC compared with HTTP2 over a TCP connection (Langley *et al.*, 2017; Cui *et al.*, 2017).

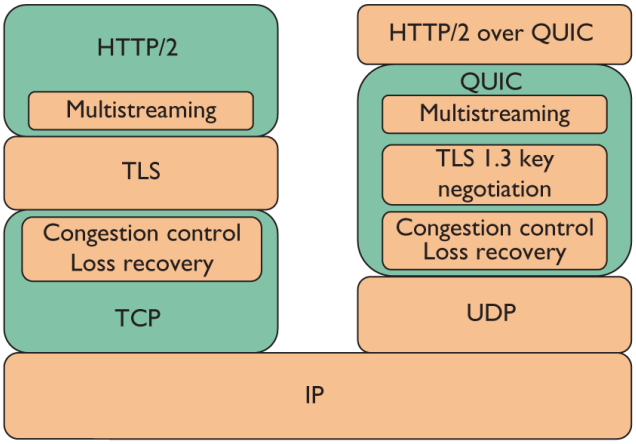

Figure 3. HTTP2 over QUIC vs HTTP2 over TCP (Cui *et al.*, 2017).

The QUIC protocol not just supports multi-stream multiplexing for HTTP traffic, like HTTP/2 over TCP, but also overcomes data delivery issues related with this type of multi-streaming. The HTTP/2 over TCP protocol multiplexes the data units related to a certain server into multiple streams carried via one connection. Delivering those streams is done in a sequential manner and, when loss happens, this stream will block the others, causing "head-of-line blocking". In contrast, the QUIC packets consist of multiple frames. Each frame encompasses stream frames resulting from multiplexing data units. If loss happens in a stream frame, the other frames will not be affected by that loss. This type of concurrent delivery can mitigate the aforementioned problem with TCP. QUIC also supports security, such as provided via TLS in HTTP. The simpler and more efficient connection establishment of QUIC, in contrast to TCP/TLS, is shown in Figure 4 (Cui *et al.*, 2017).We will consider the QUIC protocol in the experimental evaluation of our flow-based load balancing approach.

http://doi.org/10.18080/ajtde.v6n4.166

3. Flow-based Load Balancing

In this section, we briefly discuss the architecture of our flow-based load balancing system, and its implementation using OpenFlow. The overall idea is that, for each new flow (e.g. TCP or QUIC/UDP connection) initiated by the client, the SDN controller will decide to which network interface it will be allocated. Once a flow is allocated to an interface, all the corresponding packets will be sent via that interface. Changing the interface mid-flow is very difficult, and requires approaches such as Mobile IP, Shim6, HIP, etc. that are avoided in this work for the sake of simplicity and ease of deployment.

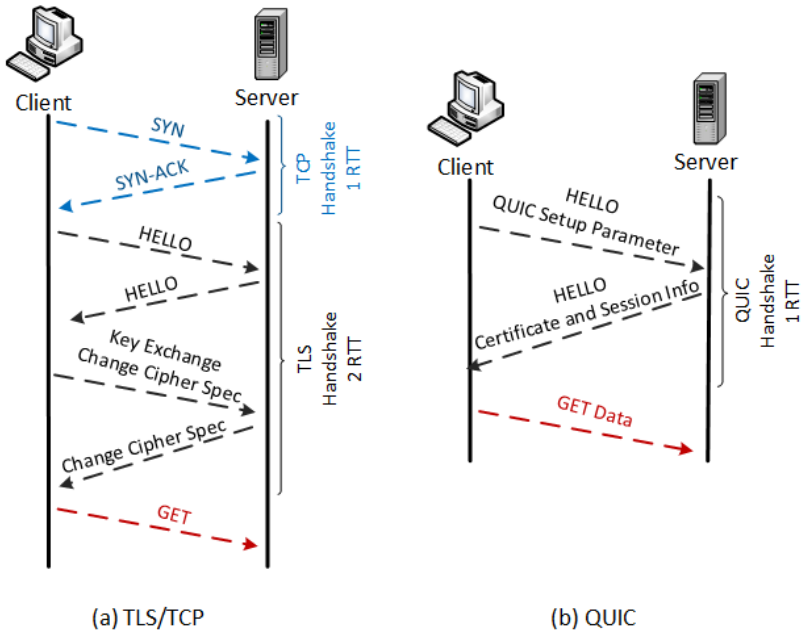

Figure 4. Handshaking of HTTP2 over TCP and QUIC protocols

The architecture of our system is shown in Figure 5. While we consider the scenario of two network interfaces, the approach works for any number of interfaces. The OpenFlow switch is bound to the two physical network interfaces, *eth0* and *eth1*. To provide it with the ability to switch network traffic across those interfaces in a way that is transparent to the application, we need to add a layer of indirection. We do this by adding a virtual interface pair (*veth0* and *veth1*). All application traffic is sent to *veth1*, via configuring the routing table. The OpenFlow switch can then control the forwarding of traffic from the application (entering the switch via *veth0*), via OpenFlow forwarding rules. In our implementation, these rules are installed by the SDN controller, which runs locally on the host.

http://doi.org/10.18080/ajtde.v6n4.166

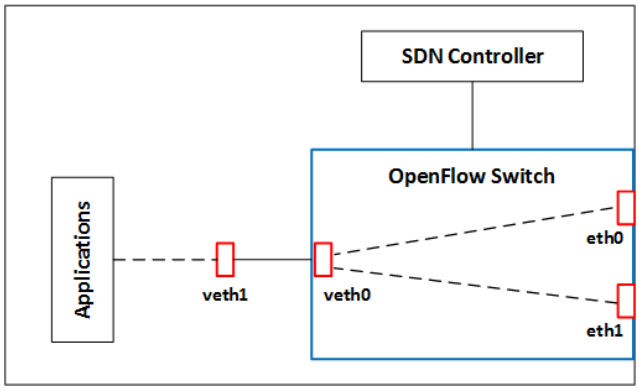

Figure 5. System Architecture

To enable transparent switching across different interfaces, we need to perform Network Address Translation (NAT), as well as ARP handling, discussed in more detail in Al-Najjar, Layeghy & Portmann (2016).

In our implementation, we used Open vSwitch (available on http://open-vswitch.org/download/) version 2.4 as our switch, and Ryu (available on http://osrg.github.io/ryu/) as our SDN controller.

3.1. Detecting and Controlling Flows

Web traffic can be transmitted over TCP or QUIC/UDP. This section discusses how new flows are detected and allocated to a particular network interface.

In the case of TCP, new flows are detected as follows. When the first packet of a new flow (i.e. TCP SYN packet) arrives at the OpenFlow switch, it will not match an existing forwarding rule, and hence it is forwarded to the controller via an OpenFlow *Packet-In* message. At this point, the controller can check that this is indeed the first packet of a new TCP connection, i.e. that the SYN flag is set. In OpenFlow version 1.3, which is used for our implementation, matching cannot be made on TCP flags, so this check can only be done at the controller. From OpenFlow version 1.5, matching on TCP flags is supported, and this can be done at the switch.

At this point, the controller decides which interface to allocate this flow to, based on the particular load balancing algorithm that is used, which will be discussed in the following section. The same basic approach is used to detect new QUIC/UDP flows, but with the additional filtering for UDP destination port 443, which is the port number allocated for QUIC servers.

Once the decision of allocating the flow (TCP or QUIC) to the specific network interface has been made, the controller installs a corresponding forwarding rule on the switch, which then sends all the packets belonging to this flow via the chosen interface, and performs the corresponding address rewriting operations. The OpenFlow match fields consist of the 5-tuple

http://doi.org/10.18080/ajtde.v6n4.166

of IP source and destination address, source and destination port number, as well as type of transport layer protocol.

3.2. Load Balancing Algorithm

To allocate network flows across multiple network interfaces, we use a *Weighted Round Robin (WRR)* load balancing algorithm, which allocates the number of flows to interfaces in proportion to their respective link capacity. To estimate the capacity of the different links in the context of SDN and OpenFlow, we utilise an active probing methodology that we have introduced in one of our previous works (Al-Najjar *et al.*, 2016). Unfortunately, this allocation can only be based on the number of flows, and does not consider the size of different flows. This is due to the fact that the flow allocation decision needs to be made when the first packet of a flow, e.g. a TCP SYN packet, is seen by the controller. Future work could potentially consider flow size estimation, to further improve the efficiency of the algorithm. However, as we will see, our flow based Weighted Round Robin algorithm considering the number of flows performs very well, due to the relatively large number of flows and their reasonably well-behaved size distribution, as discussed in the following section.

4. Web Traffic Flow Analysis

Since our load balancing approach is limited to the granularity of flows, its potential for performance improvement depends on the characteristics of the traffic in regard to flow availability and distribution. As mentioned before, in the extreme case of an application using a single large flow, flow-based load balancing cannot provide any benefit.

Since our focus is on web traffic, we performed an experimental analysis of typical websites with regard to their flow characteristics. Our methodology and results are discussed in the following.

For our analysis, we considered the top 100 Alexa websites. We downloaded the content of each website (main page) via a Python script using the Selenium WebDriver API (described on https://github.com/SeleniumHQ/selenium/), using HTTP/1.1. We disabled cookies as well as caching. All the traffic was captured as a pcap file, and the Tshark tool (Combs, 2012) (version 1.12.1) was used to analyse the data.

As a first result, Figure 6 shows the distribution of the number of flows for the 100 websites. We see a relatively long-tailed distribution, with a significant number of websites using more than 30 flows.

http://doi.org/10.18080/ajtde.v6n4.166

Based on our analysis, news sites tend to have a particularly large number of flows. Examples include *msn.com*, *theguardian.com*, *sohu.com*, and *sina.com*, with 151, 169, 207, and 281 flows, respectively. The average number of flows is around 42.

Overall, these results are encouraging for the potential of flow-based load balancing.

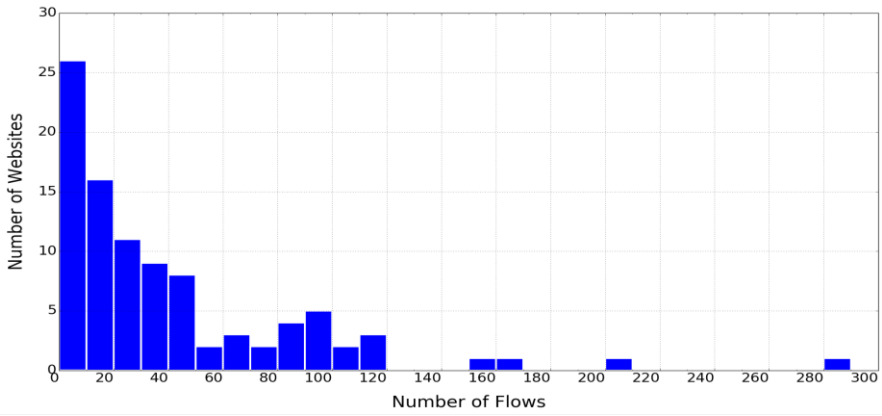

Figure 6. Alexa Top 100 Websites Flows Histogram

We also considered the size of the flows, and Figure 7 shows the distribution of flow sizes in kilobytes, using a log scale on the y-axis. We can see that, while the majority of flows are a few hundred KB or less, there are a small number of outliers, with the largest flow size close to 5MB.

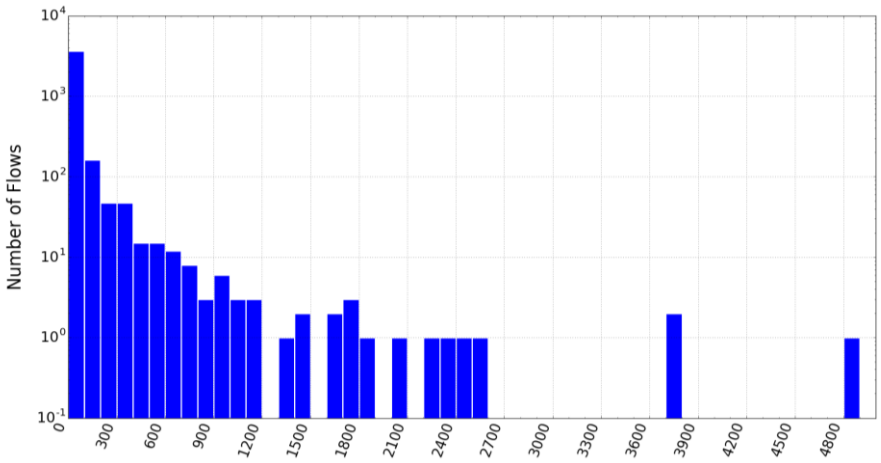

Figure 7. Alexa Flows Sizes

In summary, the distribution of flow numbers and sizes per website indicates that flow-based load balancing has the potential to deliver a performance gain, i.e. achieve a reduced page load time. We will further explore this via experiments in the following sections.

5. Load Balancing Experiment — Static Link Capacity

To evaluate the potential of flow-based load balancing for the web browsing use case, we initially performed an experiment using a scenario with a static link capacity.

http://doi.org/10.18080/ajtde.v6n4.166

Figure 8 shows the topology of our test-bed. The end-host is dual-homed and is connected to two gateways, *GW1* and *GW2*, that are connected to a physical gateway (GW) which provides connectivity to the Internet and provides access to the Alexa top 100 websites. The nodes were implemented as virtual machines (with Ubuntu Linux version 3.13.0-24 OS) and the whole topology was emulated using GNS3, a network emulation software available on https://www.gns3.com/. This will make sure that the last hop link presents the bottleneck in the end-to-end path, and should allow our load balancing approach to perform well.

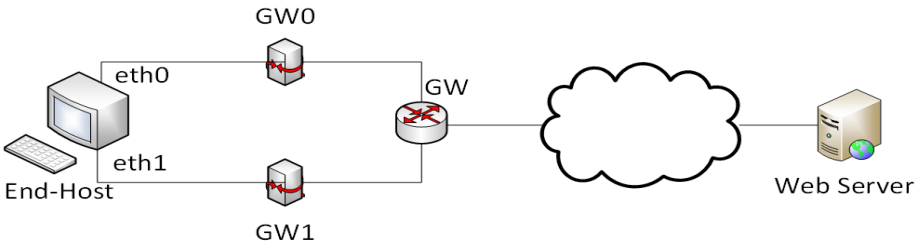

Figure 8. The Proposed Load Balancing Topology

As a performance metric, we use the page load time (PLT) (Wang & Jain, 2012), i.e. the time from when the first HTTP GET Request is sent, until the page is completely loaded. We again used the Selenium Webdriver API, along with Chromium (v58.0.3029.110), to measure the PLT for all the Alexa top 100 websites.

The static link capacity scenario is evaluated with HTTP traffic over TCP and QUIC/UDP.

5.1 Web Traffic over TCP

In this experiment, we measured the page load time (PLT) for each of the Alexa top 100 webpages 10 times, and took the average as our performance metric. We used the weighted round robin (WRR) load balancing algorithm, as discussed above, to allocate flows to the two interfaces considered in our experimental scenario. As a reference, we also measured the PLT for the single-interface case as well.

Figure 9 shows the cumulative density function (CDF) of the PLT parameter for all 100 websites. The figure clearly shows the advantage of the flow-based load balancing method. For example, in the single interface case, 50% of all page downloads are completed in under 12 seconds. In contrast, using both interfaces via flow-based load balancing, 50% of all downloads are completed in under 7.5 seconds. Overall, using both interfaces via flow-based load balancing achieves a reduction of the average page load time by almost 37%. This is a respectable improvement, considering the theoretical maximum is a reduction of 50%, and that we are working with a very coarse grained level of granularity, i.e. flows rather than packets.

http://doi.org/10.18080/ajtde.v6n4.166

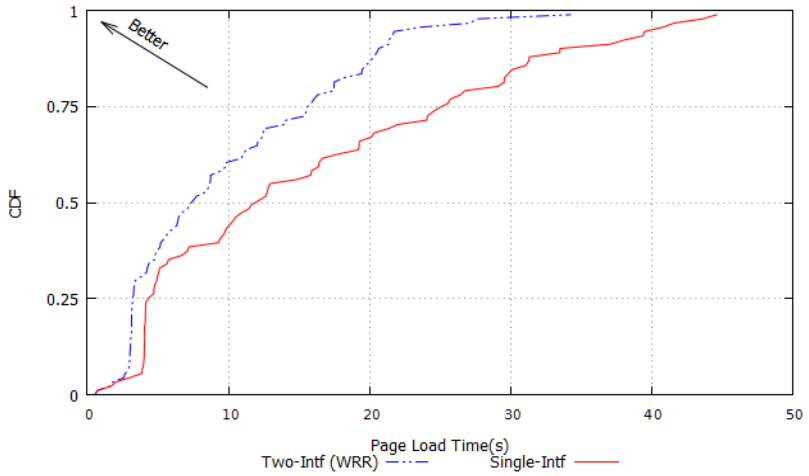

Figure 9. CDF of PLT for Static Link Capacity Scenario

We wanted to compare our flow-based load balancing approach with MPTCP, even though this is a somewhat unfair comparison. We expect MPTCP to perform significantly better, since it is able to perform load balancing on a packet-by-packet basis. On the flip side, it requires both ends to the communication path to be upgraded to support the mechanism. In contrast, our approach is a purely client-side approach, and therefore easy to deploy.

Unfortunately, none of the Alexa top 100 websites that we considered supported MPTCP. The only website that we were able to find that supports MPTCP was, somewhat ironically, mptcp.org (Paasch, *et al.*, 2013). For this measurement, we used the Linux kernel implementation of MPTCP (v.090), with the default parameter settings, as in (Paasch *et al.*, 2013).

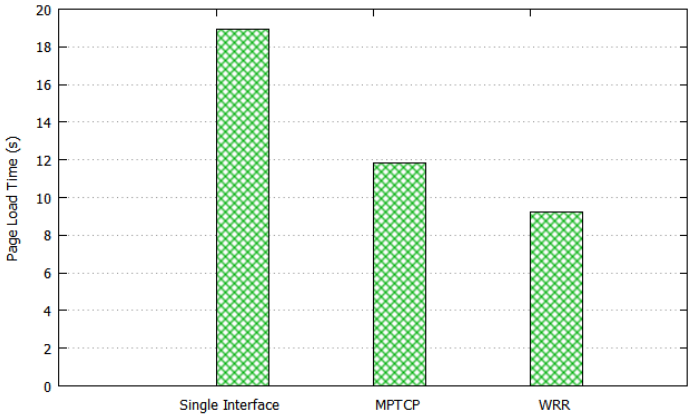

Figure 10. Mean PLT for Web Traffic over TCP (Static Link Capacity Scenario)

Figure 10 shows the page load time of mptcp.org, for three different cases: single-interface, MPTCP and flow-based weighted round robin (WRR) load balancing. Compared with the single-interface case, MPTCP reduces the page load time by 37%. Surprisingly, flow-based load balancing (WRR) clearly outperforms MPTCP and achieves a PLT reduction of 51%. Our investigations showed that MPTCP achieves a very uneven allocation of traffic across the two equal-capacity paths, with 1.3 MB of traffic sent across *etho* (see Figure 8) and only 130KB

http://doi.org/10.18080/ajtde.v6n4.166

sent across *eth1*. Another potential reason for MTPCP's relatively poor performance is its limitations in dealing with small flows, as reported in (Nikravesh *et al.*, 2016).

5.2 Web Traffic over QUIC/UDP

As previously mentioned, QUIC is a protocol developed by Google and is hence supported mostly by Google products (e.g. Chrome and Chromium browsers), as well as Google services (Google search engine and YouTube servers). In order to run QUIC, both communication end-points, i.e. the client and the server, need to support the protocol. In our experiments, we activated QUIC by enabling the "-enable-quic" option on the Chromium browser, using the Selenium API. The evaluation was done via two scenarios, with only web (HTTP) traffic, and another one with simultaneous web and video traffic.5.2.1 Web Traffic Only.

This scenario is about evaluating the control and load balancing of web traffic over the QUIC protocol. Given the limited support of QUIC on web servers, we used the YouTube main page. We loaded the page 10 times, and recorded the average page load time (PLT). We compared the results of our WRR-based load balancing approach with the scenario with a single interface only.

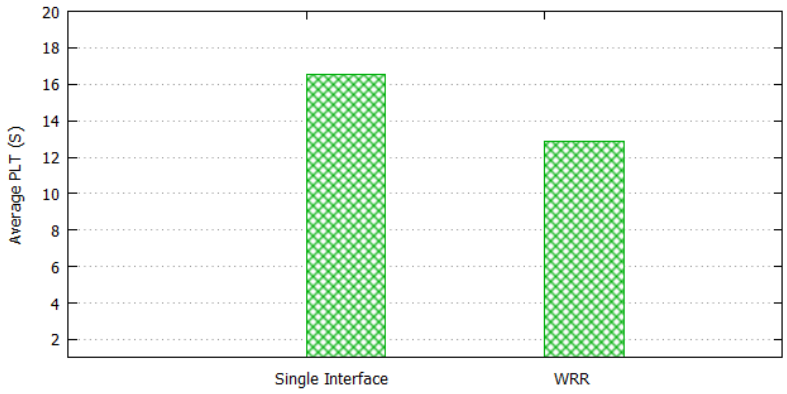

Figure 11. Mean PLT for Web Traffic over QUIC/UDP (Static Link Capacity Scenario)

Figure 11 shows the results. We can see that our WRR-based algorithm decreases the average page load time by around 30% compared to the benchmark scenario with a single interface only. While the benefits of our SDN-based load balancing approach are not quite as big as in the case of MPTCP, this experiment shows that it can still achieve a significant improvement when using the QUIC/UDP protocol.

5.2.2 Simultaneous Web and Video Traffic

Recently, multi-homed devices have allowed users to utilise multiple applications simultaneously. For instance, gadgets with decent operating systems, such as Android, offer a feature of having multi-window usage to their users. It is common to surf a website via a window while streaming a video through another window. Therefore, we adopt that scenario

http://doi.org/10.18080/ajtde.v6n4.166

to evaluate different application traffic types using our proposed system. The traffic to be evaluated is not only short-lived flows (such as webpage traffic), but also long-lived flows (e.g. DASH video traffic).

In this scenario, we consider the simultaneous flow of web and video traffic. This is an increasingly realistic and common scenario, with recent versions of Android supporting a multi-window feature, which allows users to watch a video in one browser window, while browsing a range of web pages in another window. To consider this scenario in our experiments, we used two Chromium browser windows. In the first one, we loaded the landing pages of the Alexa top 100 web sites and measured the page load time (PLT). In the other browser window, we continuously streamed a short video loaded from YouTube using the DASH (Dynamic Adaptive Streaming over HTTP) protocol, running over QUIC. The Big Buck Bunny video (available on https://www.youtube.com/watch?v=03-17GUAfNU) used in the experiment is 3 minutes long and encoded at a rate of 1 Mbps.

Our analysis using Wireshark showed the video traffic is transmitted as a single QUIC/UDP flow, as expected. This does not allow any load balancing of the video traffic across multiple interfaces. Instead, we can consider the video stream as background traffic for the simultaneously occurring web flows. Our experiments aimed to investigate the interaction between the two types of flows, and the overall performance of our flow-based traffic control and load balancing approach. Since DASH uses adaptive video encoding depending on the available bandwidth (Huang *et al.*, 2012 Akhshabi, Begen & Dovrolis, 2011), we also monitored the transmission rate of the video streams, by regularly polling the SDN switch via OpenFlow Flow Stats messages. The measured video transmission rate, or throughput, can be used as an indicator of the quality of the video, as viewed by the user. Figure 12 shows the average page load time (PLT) for the top 100 Alexa webpages for our weighted round robin (WRR) based load balancing approach, as well as the single interface scenario as a reference. We can see that, even with the video traffic in the background competing with the web traffic, our load balancing approach achieves a reduction in PLT of around 22% compared to the case where we only use a single interface.

We also considered the throughput of video traffic streamed concurrently with loading the webpages. Figure 13 shows the achieved throughput of video traffic, which is an indicator of the video QoS experienced by the end user. The figure shows three results: the video throughput achieved if we only use a single interface; the throughput achieved when using our WRR-based load balancing approach; and the *Reference* case, where there is no web traffic and video traffic has exclusive access to the available link capacity.

http://doi.org/10.18080/ajtde.v6n4.166

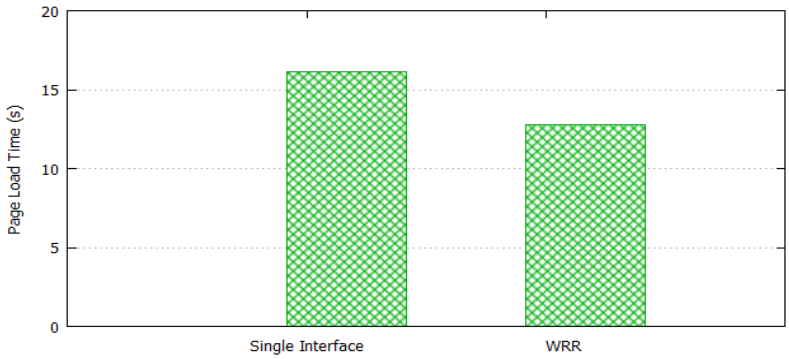

Figure 12. Mean PLT for Web and Video Traffic over QUIC/UDP (Static Link Capacity Scenario)

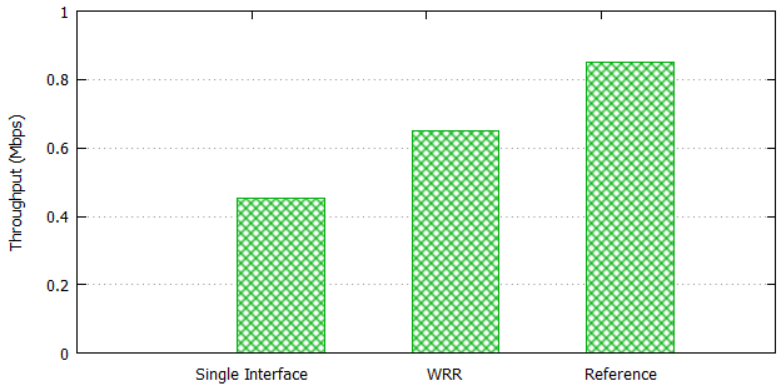

Figure 13. Throughput of Video Flows over QUIC/UDP (Static Link Capacity Scenario)

In summary, we can see that our load balancing mechanism strikes a good balance of handling competing web and video flows, and can achieve a significant reduction in page load time for web traffic, while increasing the video quality compared to the single interface case.

6. Load Balancing Experiment — Dynamic Link Capacity

In the previous section, we evaluated the concept of flow-based load balancing using a realistic traffic scenario of web-browsing. However, we considered the somewhat unrealistic scenario of static link capacities, which we used as a baseline case. In this section, we will consider a more realistic link bandwidth scenario. For this, we aim to use traffic traces from real wireless networks (WiFi and 4G) and then use these to emulate realistic links in our experiment.

While we were able to find a number of published papers and corresponding traffic traces for either WiFi or 3G/4G networks, such as in Netravali *et al.* (2015), we were not able to find any dataset which provides link bandwidth measurements for both WiFi and 3G/4G at the same time and location. However, this is exactly what we need, if we want to evaluate the potential of load balancing traffic across these types of networks.

To address this gap, we performed our own measurements. Our approach and the gathered data are discussed in the following section.

http://doi.org/10.18080/ajtde.v6n4.166

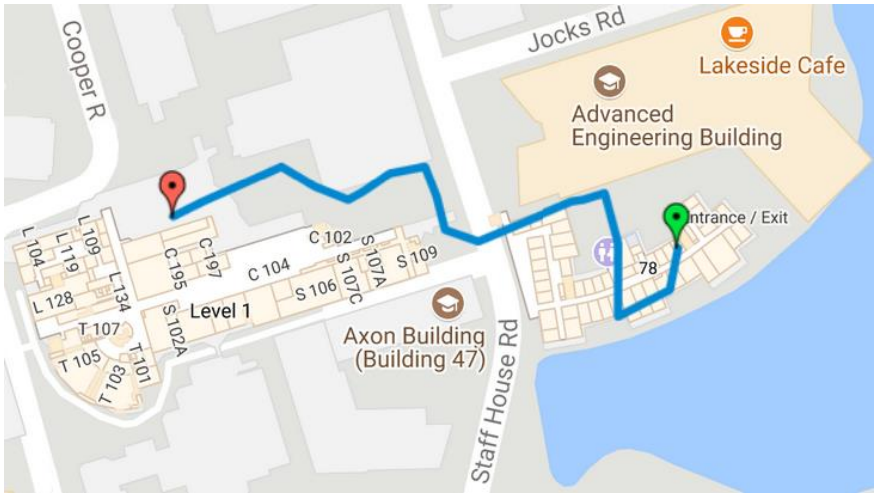

Figure 14. Bandwidth Measurement Path

6.1 WiFi and 4G/LTE Bandwidth Measurement

We performed our bandwidth measurements on the St Lucia campus of the University of Queensland (UQ). For this measurement, we walked across the campus while recording the link capacity of both the UQ WiFi network, as well as the Telstra 4G/LTE network, in 1 second intervals. The location of each measurement point was recorded using GPS. Figure 14 shows the path that was taken for our measurement. The path includes both indoor segments (starting inside building 78), as well as outdoor segments, giving a broad range of wireless link conditions. The duration of the measurement experiment is 400 seconds.

The bandwidth measurements were performed using iperf (available on https://iperf.fr/), with an iperf server running in our networking lab. located on campus. Given the high-speed campus network, it is safe to assume that our bandwidth measurement corresponds to the last-hop wireless link, since it is the path bottleneck.

For the experiment, we used two identical laptops (Dell Latitude E5470, Intel Core i5-2.3GHz, 8GB RAM, Ubuntu Linux 14.04), carried by the experimenter in a backpack. One laptop was equipped with a USB-based 4G/LTE modem (MF823). For the WiFi measurement, we used the laptop's built-in WiFi interface (Intel AC8260, 802.11a/g/n/ac).

For the iperf server, we used a Dell PowerEdge R320, Intel Xeon 2.2GHz, 32GB RAM, running the same version of Ubuntu Linux as on the laptops.

The measured bandwidth dataset is shown in Figure 15. We can see that, for the first 2 minutes of the measurements, network throughput is highly dynamic, with WiFi having a higher capacity up 160 Mbps, while LTE/4G has a capacity of well below 10 Mbps. This is as expected, since it corresponds to the indoor segment of the measurement path. For the rest of the measurement, taken outdoors, we see that 4G/LTE provides a relatively steady capacity of around 30 Mbps. In contrast, WiFi fluctuates highly and with mostly a lower capacity, and

http://doi.org/10.18080/ajtde.v6n4.166

with some sections that have no throughput at all. We will use this data set for link emulation in our flow-based load balancing experiments discussed below.

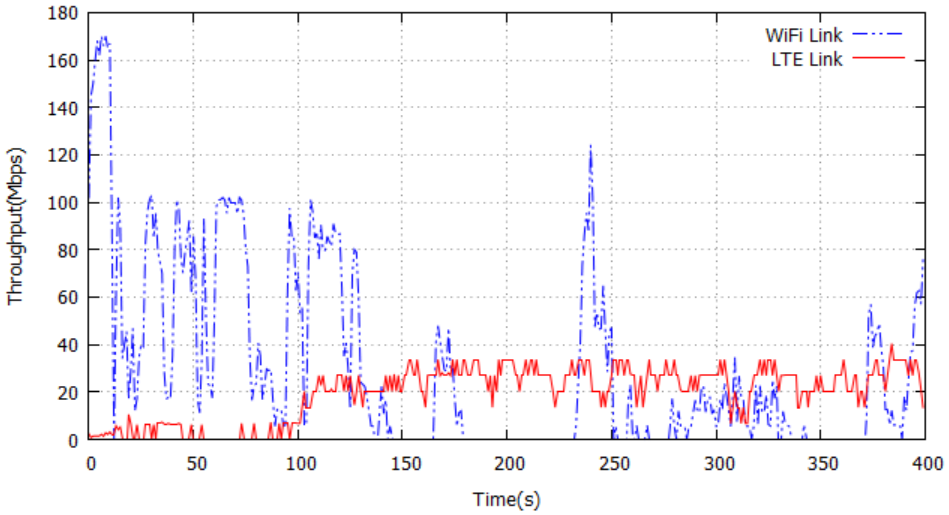

Figure 15. UQ Measured Bandwidth

6.2 Results

The testbed and scenario for this experiment are the same as discussed in Section 5 and shown in Figure 8. The only difference is that, instead of using a static link capacity for the two links (*eth0-GW0*, *eth1-GW1*), we now emulate the dynamic capacities of these links based on our measured data set (Figure 15). As before, we use the Linux *tc* tool for link emulation. Every second, *tc* is called with the corresponding link emulation parameter, i.e. bandwidth. In our scenario, link *eth0-GW0* corresponds to the WiFi link, and link *eth1-GW1* to the 4G/LTE link.

We again measure the page load time (PLT) for the Alexa top 100 websites.

In this experiment, we do this by continuously loading the same page for the entire 400 s duration of the experiments, and we record the average PLT for the period.

We considered three scenarios: using WiFi only; using 4G/LTE only; and using flow-based load balancing across both links. As in our initial experiment, we used a Weighted Round Robin (WRR) approach to load balancing. The difference in the dynamic case is that the weights are updated every second, based on the bandwidth data of the different links.

Figure 16 shows the CDF graph of the average page load time across all the 100 websites. The figure shows the results for the load balancing case (WRR) as well as for the two single-interface scenarios (LTE and WiFi). We can see that the load balancing (WRR) approach provides a significant reduction in page load time compared to both single-interface cases. For WRR, 50% of downloads are completed in under 3.9 s. The corresponding numbers for WiFi and LTE are 6.3 s and 4.8 s, respectively. Figure 17 further shows the mean PLT values for the three cases. We see that, for the single-link case, LTE achieves an average of 6.7 s, compared

http://doi.org/10.18080/ajtde.v6n4.166

to 8.6 s for WiFi. This is consistent with Figure 15, which shows that LTE has a consistently high bandwidth most of the time, compared to the more patchy performance of WiFi. Most importantly, we see that flow-based load balancing using simple weighted round robin (WRR) achieves a further reduction in PLT, with an average of 5.8 s. This represents an almost 33% reduction compared to WiFi, and a more than 13% improvement over LTE.

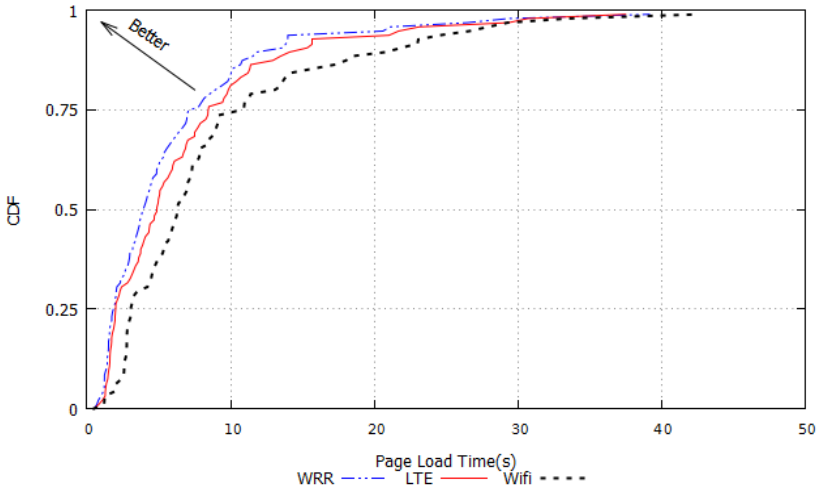

Figure 16. CDF of PLT for Web Traffic over TCP (Dynamic Link Capacity Scenario)

In summary, we have demonstrated that flow-based load balancing using simple weighted round robin has the potential to make efficient use of multiple network interfaces on end-hosts. Our experiments have shown this for the important use case of web traffic.

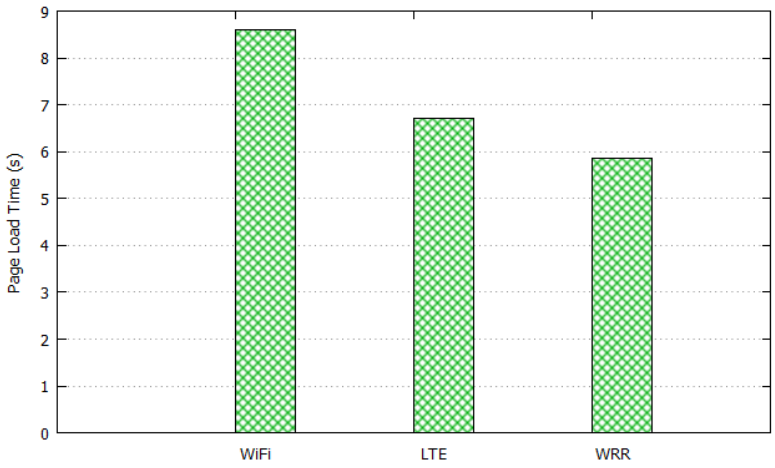

Figure 17. Mean PLT for Web Traffic over TCP (Dynamic Link Capacity Scenario)

7. Related Work

Probably the most well-known traditional approach to load balance traffic on multi-home hosts is MPTCP (Ford *et al.*, 2015). The protocol distributes TCP traffic over multiple network interfaces and end-to-end paths, and it can do this on a packet-by-packet basis. MPTCP requires deployment at both the client and server end, since it is not compatible with legacy TCP. As a result, MPTCP has achieved only limited adoption and deployment so far. Stream

http://doi.org/10.18080/ajtde.v6n4.166

Control Transmission Protocol (SCTP) (Stewart, 2007) is another transport layer protocol that supports multi-homing. Similar to MPTCP, SCTP requires support from both the client and server ends, and hence has found only very limited use. The key benefit of our flow-based load balancing approach is that it is a client-side only approach, which can easily be deployed. As a trade-off, the level of granularity is reduced (flow *vs* packet). Despite this, we demonstrated that our approach can outperform MPTCP for the web traffic use case.

A number of papers have proposed to use the SDN paradigm and OpenFlow to load balance network traffic. These works have mainly focused on load balancing in the network infrastructure and the server side (R. Wang *et al.*, 2011; Handigol *et al.*, 2009), which is in contrast to our approach.

The authors in Yap *et al.* (2012) use OpenFlow to control the network traffic in multi-homed Android hosts. The approach discusses different network functionalities, such as network hand-off, dynamic interface selection, and interface aggregation. However, the work does not address the specific problem of load balancing. Another point of difference is that the implementation of these functionalities requires support from both ends of the network path.

Another technology that allows using multiple network interfaces on end-hosts is Apple's Wi-Fi Assist, described on https://support.apple.com/en-au/HT205296, which switches to the cellular connection in case of a poor WiFi connection. This approach essentially does a vertical hand-off between the two networks, and does not allow for dynamically load balancing traffic and using both interfaces simultaneously. This is in contrast to the approach discussed in this paper.

8. Conclusions

In this paper, we have explored the concept of flow-based load balancing of network traffic across multiple interfaces on multi-homed hosts. The key benefit of this approach, compared to alternative solutions such as MPTCP, is that it is a client-side-only solution. Our approach demonstrates the capability to efficiently control and load balance HTTP flows over both TCP and QUIC/UDP. Our evaluation specifically focuses on the important use cases of web traffic, as well as simultaneous web and video traffic. Our analysis of the Alexa top 100 websites in regards to their flow distribution showed the potential for the concept of flow-based load balancing. We experimentally evaluated the concept via our OpenFlow-based implementation, considering static and realistic dynamic link capacity scenarios. Our results showed a significant performance improvement in terms of reduction in page load time, as well as increased throughput and quality of video traffic.

http://doi.org/10.18080/ajtde.v6n4.166

References

Abley, J; Black, B; Gill, V. 2003. 'Goals for IPv6 Site-Multihoming Architectures'. RFC 3582. Retrieved from https://tools.ietf.org/html/rfc3582

Akhshabi, S; Begen, AC; Dovrolis, C. 2011. 'An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP'. *Proceedings of the second annual ACM conference on multimedia systems*, pp. 157-168.

Alexa. n.d. 'The top 500 sites on the web'. Retrieved from, http://www.alexa.com/topsites/global/. Accessed 25 Jan 2017.

Al-Najjar, A; Layeghy, S; Portmann, M. 2016. 'Pushing SDN to the end-host, network load balancing using Openflow'. *2016 IEEE international conference on pervasive computing and communication workshops (percom workshops)*, pp. 1-6.

Al-Najjar, A; Pakzad, F; Layeghy, S; Portmann, M. 2016. 'Link capacity estimation in SDN-based end-hosts'. *Signal processing and communication systems (icspcs), 2016 10th international conference on*, pp. 1-8.

Chen, Y-C; Lim, Y.-S; Gibbens, RJ; Nahum, EM; Khalili, R; Towsley, D. 2013. 'A measurement-based study of multipath TCP performance over wireless networks'. *Proceedings of the 2013 conference on internet measurement conference*, pp. 455-468.

Combs, G. 2012. 'Tshark Dump and analyze network traffic'. Retrieved from https://www.wireshark.org/docs/man-pages/tshark.html

Cui, Y; Li, T; Liu, C; Wang, X; Kiihlewind, M. 2017. 'Innovating transport with QUIC: Design approaches and research challenges'. *IEEE Internet Computing*, *21* (2), pp. 72-76.

Ford, A; Raiciu, C; Handley, MJ; Bonaventure, O. 2015, October 14). 'TCP Extensions for Multipath Operation with Multiple Addresses'. RFC 6824. RFC Editor. Retrieved from https://www.rfc-editor.org/rfc/rfc6824.txt

Handigol, N; Seetharaman, S; Flajslik, M; McKeown, N; Johari, R. 2009. 'Plug-n-serve: Load-balancing web traffic using Openflow'. *ACM SIG-COMM Demo, 4* (5), p. 6.

Huang, T-Y; Handigol, N; Heller, B; McKeown, N; Johari, R. 2012. 'Confused, timid, and unstable: picking a video streaming rate is hard'. *Proceedings of the 2012 internet measurement conference*, pp. 225-238.

ISO. 2014. 'Information technology — Dynamic adaptive streaming over HTTP (DASH). Part 1: Media presentation description and segment formats' ISO/IEC 23009-1:2014, May. Retrieved from https://www.iso.org/standard/65274.html

http://doi.org/10.18080/ajtde.v6n4.166

Langley, A; Riddoch, A; Wilk, A; Vicente, A; Krasic, C; Zhang, D; *et al.* 2017. 'The QUIC transport protocol: Design and internet-scale deployment'. *Proceedings of the conference of the ACM special interest group on data communication*. pp. 183-196.

McKeown, N; Anderson, T; Balakrishnan, H; Parulkar, G; Peterson, L; Rexford, J; ... Turner, J. 2008. 'Openflow: enabling innovation in campus networks'. *ACM SIGCOMM Computer Communication Review*, *38*(2), pp. 69-74. doi: 10.1145/1355734.1355746

Moskowitz, R; Nikander, P; Jokela, P; Henderson, T; Heer, T. 2010. 'Host identity protocol', RFC 5201-bis. Retrieved from https://tools.ietf.org/html/draft-moskowitz-hip-rfc5201-bis-01

Netravali, R; Sivaraman, A; Das, S; Goyal, A; Winstein, K; Mickens, J; Balakrishnan, H. 2015. 'Mahimahi: Accurate record-and-replay for HTTP'. *Usenix annual technical conference*, pp. 417-429.

Nikravesh, A; Guo, Y; Qian, F; Mao, Z. M; Sen, S.(2016. 'An in-depth understanding of multipath TCP on mobile devices: Measurement and system design'. *Proceedings of the 22nd annual international conference on mobile computing and networking*, pp. 189-201.

Paasch, C; Barre, S; *et al.* 2013. 'Multipath TCP in the Linux kernel'. Available from https://www.multipath-tcp.org/

Perkins, C. 2002. 'IP mobility support for IPv4'. RFC 3344.

Roskind, J. 2013. 'QUIC (Quick UDP Internet Connections): Multiplexed stream transport over UDP'. *IETF-88 TSV Area Presentation*. Retrieved from https://www.ietf.org/proceedings/88/slides/slides-88-tsvarea-10.pdf

Stewart, R. 2007. 'Stream control transmission protocol', RFC 4960. Retrieved from https://tools.ietf.org/html/rfc4960

Wang, R; Butnariu, D; Rexford, J. 2011. 'Openflow-based server load balancing gone wild'. *Hot-ICE'11: Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, p. 12.

Wang, Z; Jain, A. 2012. 'Navigation timing', W3C Recommendation 13 December 2012. Retrieved from http://www.w3.org/TR/2012/REC-navigation-timing-20121213/

Yap, K-K; Huang, T.-Y; Kobayashi, M; Yiakoumis, Y; McKeown, N; Katti, S; Parulkar, G. 2012. 'Making use of all the networks around us: a case study in android'. *Proceedings of the 2012 ACM SIGCOMM workshop on cellular networks: operations, challenges, and future design*, pp. 19-24.

http://doi.org/10.18080/ajtde.v6n4.166