HELDA
HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

UNIVERSITY OF HELSINKI

https://helda.helsinki.fi

An SDN Perspective on Multi-connectivity and Seamless Flow Migration

Hätönen, Seppo

2020

Hätönen , S , Huque , T I U , Rao , A , Jourjon , G , Gramoli , V & Tarkoma , S 2020 , ' An SDN Perspective on Multi-connectivity and Seamless Flow Migration ' , IEEE Networking Letters , vol. 2 , no. 1 , pp. 19-22 . https://doi.org/10.1109/LNET.2019.2951831

http://hdl.handle.net/10138/348166
https://doi.org/10.1109/LNET.2019.2951831

acceptedVersion

Downloaded from Helda, University of Helsinki institutional repository.

This is an electronic reprint of the original article.

This reprint may differ from the original in pagination and typographic detail.

Please cite the original version.

An SDN Perspective on Multi-connectivity and Seamless Flow Migration

Seppo Hätönen*, Tanvir Ishtaique ul Huque†‡, Ashwin Rao*, Guillaume Jourjon†, Vincent Gramoli§, Sasu Tarkoma*
*University of Helsinki, †Data61-CSIRO, ‡University of New South Wales, §University of Sydney

***Abstract*—Devices capable of multi-connectivity currently use static rules for selecting the set of interfaces to use. Such rules are limited in scope and can be counter-productive. We posit that SDN techniques can address this inefficiency. We present an approach that enables an SDN controller to manage the flows traversing the Ethernet, Wi-Fi, and LTE links in our laptop and also migrate the flows from one link to another. Our solution opens avenues that enable end-user device to negotiate with the network controllers when taking its control plane decisions.**

***Index Terms*—Network Management, SDN, Multi-connectivity.**

I. Introduction

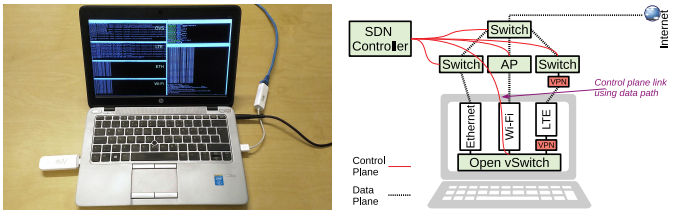
Identifying the set of communication interfaces to use is a key decision taken by devices capable of multi-connectivity. Multi-connectivity is not new and laptops are capable of connecting to the Internet via a combination of interfaces such as Wi-Fi, cellular, or Ethernet. However, selecting the set of interfaces to use is largely governed by static rules. For example, our phones typically prefer Wi-Fi over cellular connectivity regardless of the quality of the Wi-Fi network. Bahl *et al.* [1] detail this inefficiency while highlighting the benefits of using multiple interfaces in a collaborative manner.

Software Defined Networking (SDN) brings the capability of taking dynamic decisions based on the instantaneous network state. This has motivated researchers to leverage SDN techniques for taking control plane decisions in multi-connectivity scenarios. However, these works either focus on managing the Access Points (APs) and base stations [2], [3], identifying the interface to use locally on the device without communicating with the network controllers [4], [5], or are focused solely on enforcing security policies [6].

SDN can also be leveraged for managing the network interface of an end-user devices. Creating a software switch on the end-user device and using its virtual interface decouples the applications from the underlying network interfaces [4], [1], [7]. This enables the device to coordinate with the network controllers to identify which interfaces to use, allowing a more optimal interface selection. Furthermore, decoupling interfaces from applications enables seamless vertical handovers.

In this paper we exemplify our approach that allows a network controller to select the communication interfaces used by end-user devices. Our key contributions are as follows.

1) Our approach is agnostic to the underlying network interfaces and we exemplify it by migrating the network flows of a laptop with Ethernet, Wi-Fi, and LTE interfaces. We consider a single network which provides a single IP address to our laptop. This assumption is in line with the network of the Department of Computer Science at the University of Helsinki, where each managed device has a single IP address across all interfaces.
2) We present an algorithm for migrating flows which builds on the work of Huque *et al.* [8]. Our enhancement ensures that the rules in the switches common to the path prior to flow migration and after the flow migration are left untouched while also ensuring per-packet consistency.
3) We observe that the flow migration times when using an implementation of our approach is typically less than 100 ms when the links are not saturated (see §III). Furthermore, in our setup, the flow migrations are seamless and do not cause existing TCP connections to reconnect.



(a) Laptop with 3 interfaces. (b) Network topology.

Fig. 1. **Multi-connectivity Setup.** *The Ethernet, Wi-Fi, and LTE interfaces of the laptop are plugged to an SDN switch running on the laptop.*

Our solution enables end user devices to leverage on SDN techniques for selecting the interfaces it uses and implementing network-triggered handovers. As discussed in §IV, the key challenges in using an SDN switch on an end-host include (i) bootstrapping the networking connectivity, and (ii) setting up the control channel between the switch and the controller.

II. Solution Description

Our solution leverages SDN for programmable link selection and flow migration. Figure 1(a) presents our prototype laptop which has three interfaces: Ethernet, Wi-Fi, and LTE. As shown in Figure 1(b), all three interfaces offer connectivity via a single network. We make this assumption because carriers are increasingly offering Wi-Fi and LTE connectivity. All three interfaces are connected to an SDN switch running on the laptop. For simplicity, we assume that the SDN controller managing the switches in the network also manages the switch on the laptop. Note that the switches in the network use an out-of-band control channel for communicating with the controller except the switch in the laptop which uses an in-band control channel. Further details of the prototype, including the reason

Algorithm 1: The set of rules to add and delete

Data: P'': the new path, and P': the current path
Result: α: set of rules to add, and δ: set of rules to delete

$S' = \{s(r) \mid r \in P'\}$
$S'' = \{s(r) \mid r \in P''\}$
$\theta' = \{\langle s(r), m(r), a(r)\rangle \mid r \in P', s(r) \in S' \bigcap S''\}$
$\theta'' = \{\langle s(r), m(r), a(r)\rangle \mid r \in P'', s(r) \in S' \bigcap S''\}$
$\alpha = \{r \mid r \in P'', s(r) \in S'' \setminus S'\}$
$\delta = \{r \mid r \in P', s(r) \in S' \setminus S''\}$
foreach $j \in S'' \bigcap S'$ **do**
    $r_\alpha = r \mid r \in P'', s(r) = j$
    $r_\beta = r \mid r \in P', s(r) = j$
    if $\langle s(r_\alpha), m(r_\alpha), a(r_\alpha)\rangle \notin \theta' \bigcap \theta''$ **then**
        $\pi(r_\alpha) \Leftarrow \pi(r_\beta) + 1$
        $\alpha = \alpha \bigcup r_\alpha$
        $\delta = \delta \bigcup r_\beta$
    end
end

for using the VPN tunnels for the LTE interface are discussed in §III. We also plan to extend this solution by running a local instance of an SDN controller, and this is discussed in §IV.

We now present our algorithm that our SDN controller uses for migrating the flows from one interface to another. Let S denote the set of SDN-capable forwarding elements in the network; we henceforth refer to forwarding elements such as switches and routers as switches. The rules managing all the flows in the network is $R = \bigcup_{j \in S} R_j$ where R_j is the set of rules on switch $j \in S$. We represent the path traversed by a flow of packets in a given direction by an *ordered sequence* of rules serving that flow. $P = r_1, r_2, \ldots, r_n$ where $r_j \in R$ and $1 \leq j \leq n$. We consider flows to be unidirectional, and bidirectional flows are considered two independent flows. Let P' and P'' respectively denote the current path traversed by the flow and the new path after the flow migration.

A straw-man approach for flow migration would be to install the rules present in P'' followed by removing the rules in P'. This can be implemented using the technique of Huque *et al.* [8] which builds on the two-phase commit [9] and the rule reverse update technique [10]. If the length of P' and P'' are n and m respectively, then this approach requires $n+m$ operations. However, this approach is sub-optimal because the controller ends up updating the rules on the switches common to the current path and the new path. As shown in Figure 2, the switches common to both paths are typically in the core of the network and it is desirable to minimize changing their rules. Therefore, our goal is to ensure that *the rules not affected by flow migration are left untouched by the SDN controller.*

We assume that each rule is described by four fields: i) the switch id, ii) the rule priority, iii) the match field, and iv) the action field; for a given rule r, we assume these four fields are given by the functions $s(r)$, $\pi(r)$, $m(r)$, and $a(r)$ respectively. As shown in Algorithm 1, we assume S' and S'' respectively denote the set of switches on the current and new path. We use the switches in $S' \bigcap S''$, and the match and action fields of the rules in P' and P'' for creating two sets of 3-tuples θ' and θ''. Note that a rule is already serving the flow if its switch id, match field, and action field is in $\theta' \bigcap \theta''$. Therefore the SDN controller does not need to install these rules. As shown in Algorithm 1, the rules on switches in $S'' \setminus S'$ can be added with their default priority, and the rules in $S' \setminus S''$ can be deleted. Furthermore, the priority of the rules in P'' whose switch ids are in $S' \bigcap S''$ but whose switch ids, match, and action fields are not in $\theta' \bigcap \theta''$ should be higher than the priority of the corresponding rules in P'. This gives us α and δ, the set of rules to add to R and delete from R respectively. We add and delete the rules using the technique of Huque *et al.* [8], which in-turn uses *probe packets* for ensuring per-packet consistency and blackhole-freedom during the flow migration.

By not modifying the rules whose switch ids, match field, and action field are in $\theta' \bigcap \theta''$, Algorithm 1 ensures that the SDN controller does not modify the rules which are not affected by flow migration.

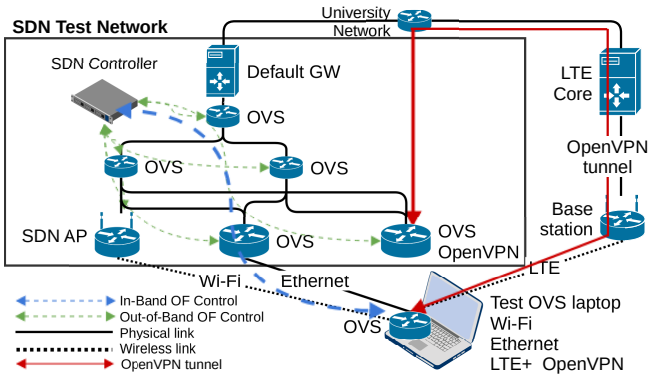


Fig. 2. **Testbed.** *Our test network contains interlinked SDN switches, an SDN capable AP, and an LTE connection via a VPN tunnel.*

III. Evaluation

We implemented our algorithm in an SDN controller written using the Ryu framework [11], and used it to migrate flows between the three interfaces of our laptop. We now detail our evaluation testbed, the test scenarios, and our results.

A. Testbed Description

As shown in Figure 2, the laptop's LTE, Wi-Fi, and Ethernet interfaces are plugged to an Open vSwitch (OVS) bridge running on the laptop. While plugging the Ethernet interface to the OVS bridge is straightforward, the Wi-Fi and LTE interface required the following steps. We assigned the OVS bridge the same MAC address as that of the Wi-Fi interface because Wi-Fi APs typically drop packets that have a different destination MAC address than any of the associated clients. LTE modems, such as the USB Huawei E3372h LTE dongle shown in Figure 1(a), typically expose a Point-to-Point Protocol (PPP) interface which cannot be directly plugged to an OVS bridge [12]. We therefore use an Open VPN client which exposes a TAP interface [13], which was then plugged to the OVS bridge. This VPN client connects to a VPN server running on a device connected to an SDN switch.

We assign the laptop a single IP address, which is given to the OVS, *i.e.* the physical interfaces or the VPN TAP interface do not have their own IP addresses. The LTE modem has its

own IP address, but this address is only used to create the VPN tunnel. This approach is similar to HetSDN [7], however we only use tunnels over LTE, we do not have a dedicated Home Agent, and we use a single SDN controller for both the network and the OVS on the laptop.

We use Linksys WRT3200ACM APs [14] with OVS version 2.9.2. The Ethernet operates at 1 Gbps link with sub millisecond latency. The Wi-Fi speed is up to 100 Mbps, however the latency varies from few milliseconds under no load to hundreds of milliseconds when the link is saturated; this is largely due to the bufferbloat problem [15]. The LTE link provides up to 40 Mbps bandwidth with a latency of 30 ms.

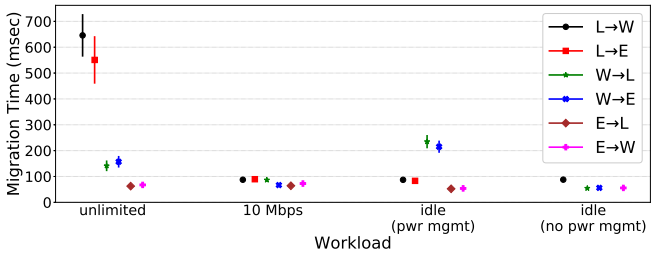
B. Test Metrics and Workloads

The objective of our evaluation is to highlight the impact of i) the different characteristics of LTE, Wi-Fi, and Ethernet links, ii) the buffers in the network, and iii) the in-band control channel because the OpenFlow messages and the data traffic from the laptop share the same interface.

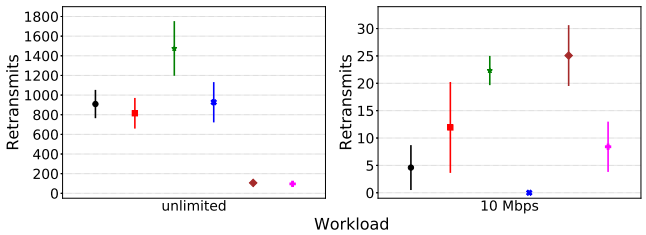
We use the flow migration time and the number of TCP retransmissions to evaluate the performance of our approach. Migrating a TCP flow from one interface to another therefore does not incur TCP reconnects. This implies that there are always packets in flight during the flow migration. The controller therefore sends probe packets in the current path after the rules for the new path have been installed on the switches, and before the rules for the current path are removed [8]. We consider a migration to be complete only after the *probe packets* have been received at their destination. The arrival of probe packets at the destination implies that the packets of the flow which were in flight when the migration started are no longer in flight; they were either received at the destination or lost in flight during the migration. Our approach can also cause TCP retransmissions because packets taking the new path might arrive before packets taking the current path.

We evaluate the performance of our solution with *iperf3* [16] under three different TCP traffic workloads: i) *unlimited* with no bandwidth restrictions, ii) *10 Mbps*, where *iperf3* throttled the rate of the TCP flows to 10 Mbps, and iii) *idle*, where the interface used by the device was changed when *iperf3* was not running. To meet our objectives, we only considered downlink traffic, *i.e.* traffic sent from the default Internet gateway (GW) to our laptop. For each workload, and for each of the 6 combinations of flow migrations between Ethernet (E), LTE (L), and WiFi (W) links, we run 30 iterations. For each 20 second iteration, we first start the flow on one interface and then after 10 seconds we migrate it to the other interface. This allows the TCP flow to stabilize before migration.

The *idle* workload gives the baseline performance on the time required by the SDN controllers to update the flow rules. Note that the lack of traffic triggers power management in the Wi-Fi interface and we therefore conducted this test twice: once when the Wi-Fi power management was enabled (the default), and once when it was turned off. The LTE modem does not have power management options available, so we could not perform similar tests for it. The *10 Mbps* workload



(a) Mean time to migrate a flow.



(b) Number of TCP retransmissions during flow migration.

Fig. 3. **Evaluation results.** *We consider all 6 combinations of flow migrations between Ethernet (E), LTE (L), and WiFi (W) links. The common legend is shown in the top figure and L→W implies migrating a flow from LTE to Wi-Fi. The error bars represent the 95% confidence interval across 30 iterations.*

emulates migrating a flow used for streaming high quality videos. The *unlimited* workload emulates conditions when a flow is migrated from a saturated link to an unsaturated link. This workload highlights the impact of network buffers on the flow migration time and the TCP retransmissions.

C. Results

In Figure 3(a), we observe that the migration time of a flow depends on the workload and the interfaces used. Across all workloads, migrating flows from Ethernet incurred a smaller migration time compared to migrating flows from LTE. This is expected because the LTE link between the device and the SDN switch to which it is connected incurred a latency of around 30 ms even when there was no traffic.

For the *unlimited workload*, we observe that the migration times from LTE to other interfaces is much higher than other combinations, up to 800 ms. Similarly, migrating from Wi-Fi takes around 160ms, while from Ethernet is always below 100 ms. The higher migration times are explained by the bufferbloat problem. In LTE, we observe Round-Trip-Times (RTT) increasing from the baseline of 30 ms up to 400 ms over the first 10 seconds of the test, indicating large network buffers. When migration is triggered, the controller sends flow modification message over the in-band control channel. This message and corresponding reply use one RTT, and when the traffic begins to use the new path, a probe message is sent over the previous path, taking one RTT. In total, the migration uses two RTTs of the original link, *i.e.* up to 800 ms. We observe a similar behaviour with Wi-Fi, but the effect is smaller.

In contrast, we observed migration times of less than 100 ms for the *10 Mbps* workload for all combinations of LTE, Wi-Fi, and Ethernet interfaces. As the workload is less than any of the link capabilities, the traffic is not buffered.

For the *idle* workload, we observe that the migration time from Wi-Fi when Wi-Fi power management is enabled (de-

fault) is thrice the migration time when it is disabled. This is because the migration process requires waking up the Wi-Fi interface so that the controller can update the rules.

In Figure 3(b) we present the number of TCP retransmissions occurred when migrating a flow from one interface to another. For the *unlimited* workload, we observe that the buffer bloat problem results in a high number of TCP retransmissions when migrating from Wi-Fi and LTE links. For the *10 Mbps* workload we observe a significantly smaller number of TCP retransmissions compared to the *unlimited* workload.

These results highlight the impact of the in-band control channel when migrating flows across interfaces with completely different link characteristics. Our solution does not interrupt TCP sessions within single administrative domain.

IV. Discussion and Concluding Remarks

We now discuss our insights and some key additional challenges in leveraging SDN techniques for implementing seamless flow migrations in networks offering multi-connectivity.

1) *In-band control plane.* An in-band control plane causes the control plane messages to be queued up in large in-network buffers with the data plane traffic. The impact of these delays can be further aggravated by the packet losses in the wireless medium. This issue can be addressed by dedicating one interface for the control plane traffic. However, this requires multiple simultaneously connected interfaces, which is not possible at all times, and it also incurs other costs such as increased energy consumption.

2) *Network Loops.* As shown in Figure 1(b), an SDN switch on an end-user device for implementing multi-connectivity creates a loop in the network. This can cause flooding if the broadcast packets are not handled correctly. Switches in our network therefore send broadcast packets to our controller. For ARP queries, the controller sends them directly to the host whose address is being queried. Unfortunately this approach does not scale and there is a need for coming up with a more efficient solution that handles broadcast packets.

3) *Host deletion and disconnects.* Communication interfaces can enter power saving modes where the network interfaces stay up, but no traffic is flowing between the host and the network. In the absence of signaling between the host and the network controller, the controller cannot determine if a host has entered sleep mode, left the network, or disconnected.

4) *Wi-Fi Roaming.* Roaming requires the controller to support dynamic link disconnections and creation of new links. When the controller detects roaming, it must set the new path and delete the current path. However, one key difference between flow migration and roaming is that the current path may not be available when a device is roaming. Our solution can also complement solutions which allows a device to switch to cellular networks when roaming between Wi-Fi networks.

5) *Network Bootstrap.* SDN switches use a dedicated out-of-band link to the SDN controller to exchange messages during network bootstrap. In contrast, the switch on the end-user device uses an in-band link to the controller, which does not exist during the network bootstrap. The switch therefore requires either a local controller to set basic rules, or preset low priority rules to establish the network connectivity. Furthermore, these rules should be comprehensive enough to ensure network connectivity when the device is unable to communicate with a network controller.

6) *Local Controller.* Our proof of concept only has a single SDN controller. While such a centralized controller can be implemented in a distributed manner, it might not be able to scale to manage all the user devices in the network. This calls for a more decentralized approach where each user device has its own local controller. Such a controller opens avenues for a more user-centric approach to be taken into the network management. We envision that this local controller will have the key role in taking decisions affecting the device's traffic flows and it will need to work with controllers of the networks offering connectivity to the device. Such a split topology of network controllers requires addressing a wide range of issues including trust, consistency, and discovery management.

To conclude, we exemplify how multi-connectivity can be realized through SDN, and our solution opens avenues towards addressing challenges brought forth by multi-connectivity.

References

[1] P. Bahl, A. Adya, J. Padhye, and A. Wolman, "Reconsidering Wireless Systems with Multiple Radios," *SIGCOMM CCR*, vol. 34, no. 5, 2004.

[2] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "OpenRoads: Empowering Research in Mobile Networks," *SIGCOMM CCR*, vol. 40, no. 1, Jan. 2010.

[3] M. Yan, C. J. Casey, P. Shome, A. Sprintson, and A. Sutton, "Ætherflow: Principled wireless support in SDN," in *23rd IEEE ICNP*, 2015.

[4] K.-K. Yap, T.-Y. Huang, M. Kobayashi, Y. Yiakoumis, N. McKeown, S. Katti, and G. Parulkar, "Making Use of All the Networks Around Us: A Case Study in Android," *SIGCOMM CCR*, vol. 42, no. 4, Sep. 2012.

[5] S. Deng, A. Sivaraman, and H. Balakrishnan, "All Your Network Are Belong to Us: A Transport Framework for Mobile Network Selection," in *HotMobile*. ACM, 2014.

[6] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards SDN-Defined Programmable BYOD (Bring Your Own Device) Security," in *NDSS*, 2016.

[7] Kang Chen, R. Izard, Hongxin Hu, K. Wang, J. Martin, and Juan Deng, "HetSDN: Exploiting SDN for intelligent network usage in heterogeneous wireless networks," in *IEEE/ACM IWQoS*, 2016.

[8] M. T. I. ul Huque, G. Jourjon, and V. Gramoli, "Garbage Collection of Forwarding Rules in Software Defined Networks," *IEEE Communications Magazine*, vol. 55, no. 6, 2017.

[9] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for Network Update," in *ACM SIGCOMM*, 2012.

[10] D. M. F. Mattos, O. C. M. B. Duarte, and G. Pujolle, "Reverse Update: A Consistent Policy Update Scheme for Software-Defined Networking," *IEEE Communications Letters*, vol. 20, no. 5, 2016.

[11] Ryu SDN Framework Community, "Ryu sdn framework." [Online]. Available: https://osrg.github.io/ryu/

[12] Open vSwitch, "Frequently asked questions." [Online]. Available: http://www.openvswitch.org/support/dist-docs-2.5/FAQ.md.txt

[13] F. Thiel, "Universal tun/tap device driver." [Online]. Available: https://www.kernel.org/doc/Documentation/networking/tuntap.txt

[14] S. Hätönen, P. Savolainen, A. Rao, H. Flinck, and S. Tarkoma, "SWIFT: bringing SDN-Based flow management to commodity Wi-Fi access points," in *IFIP Networking Conference*, May 2018.

[15] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Queue*, vol. 9, no. 11, Nov. 2011.

[16] ESnet / Lawrence Berkeley National Laboratory, "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool." [Online]. Available: https://iperf.fr/