HetSDN: Exploiting SDN for Intelligent Network Usage in Heterogeneous Wireless Networks

Kang Chen*#, Ryan Izard†, Hongxin Hu†, Kuang-Ching Wang†, James Martin†, Juan Deng†
*Southern Illinois University, kchen@siu.edu
†Clemson University, {rizard, hongxih, kwang, jmarty, jdeng}@clemson.edu

***Abstract*—Mobile devices nowadays can find multiple wireless networks, such as WiFi, 4G/LTE and relay through devices. These networks have different characteristics in terms of coverage, data rate, and price. Meanwhile, mobile applications (and even different TCP/UDP connections) often have diverse and time-variant network needs. Thus, to better use all wireless network resources, it would be ideal to enable a TCP/UDP connection to 1) select the most appropriate network dynamically and 2) migrate between networks transparently. However, existing methods fail to provide both functions in a systematic and efficient way at the TCP/UDP connection level. In this paper, we adopt Software-Defined Networking (SDN) to realize such a feature. We use the features of SDN to realize intelligent network selection that is adaptive to time-variant application needs, network availability, and scheduling commands. To support transparent migration, an intelligent home agent (HA) is designed with the SDN to anchor packets from the mobile device. It can intelligently determine which wireless network a TCP/UDP connection is running over. Finally, our implementation demonstrates the effectiveness and efficiency of the proposed system.**

I. Introduction

The demand for network connectivity from mobile devices is growing rapidly. By mobile devices, we refer to not only smartphones or laptops but also devices in emerging mobile networks such as connected vehicles [1] and Internet of Things [2]. To satisfy such a need, various wireless access techniques such as WiFi and 4G/LTE have been widely used. These networks have diverse characteristics in terms of coverage, data rate, and price and form a heterogeneous wireless network environment. Generally, we can assume that 4G/LTE networks offer high speed and high coverage but are expensive, while WiFi offers high speed network service at a low cost but has a limited coverage. In order to exploit all available wireless networks, mobile devices usually are equipped with multiple wireless interfaces.

Meanwhile, mobile applications often present diverse and time-variant network needs. For example, on connected vehicles, the collision avoidance application needs reliable network connection, while the backup of non-critical data can tolerant certain delay. An important Skype video interview would favor a faster network than that for video chat between friends. Such a diversity also exists at the TCP/UDP connection level since an application may have different TCP/UDP connections for different tasks. For example, a vehicle status monitoring application can accept an expensive (but more reliable) network for emergency messages (e.g., accidents) but would like to use the low cost network for regular status update.

#The work was started when at Clemson University.

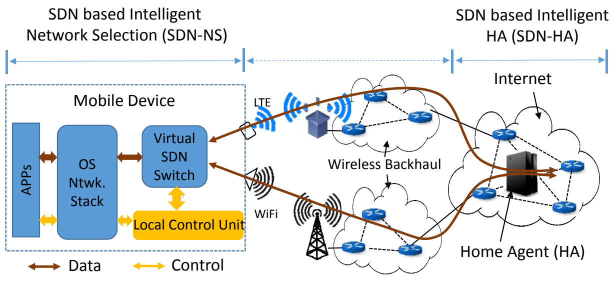


Fig. 1: HetSDN overview.

Furthermore, network availability may change due to user mobility or the scheduling from network operator. All those facts indicate that network usage (at the TCP/UDP level) should be adaptive to not only application needs but also network availability. To attain this goal, two features need to be supported at the TCP/UDP connection level: 1) select the most appropriate network dynamically and 2) migrate between networks transparently without being disconnected. We denote the two features as connection level intelligent network selection and connection level transparent mobility, respectively. The former helps a connection select the optimal network under diverse application needs and network availability, while the latter is needed to enforce the network selection timely (i.e., on the run) without being interrupted.

Therefore, in this paper, we propose HetSDN that utilizes SDN to systematically realize both intelligent network selection and transparent mobility at the TCP/UDP connection level in a heterogeneous network environment. Figure 1 shows an overview of HetSDN with two components: SDN based intelligent network selection (SDN-NS) and SDN based intelligent home agent (SDN-HA). The SDN-NS enables TCP/UDP connections on a mobile device to dynamically use the most appropriate wireless network based on application needs, local network status, and global network scheduling commands. The SDN-HA enables connections to migrate between wireless networks transparently and immediately without being disrupted, once a network switch decision has been made. Both components exploit unique features of SDN (i.e., flexible control and/or global view).

In the following, Section II introduces related work. Section III presents the design of HetSDN. Section IV evaluates HetSDN, while Section V concludes the paper.

II. Related Work

A. Exploiting Multiple Wireless Networks

Exploiting multiple concurrently available wireless networks has already been widely studied [3]–[10]. The first group of works focus on augmenting network usage (e.g., bandwidth) by exploiting multiple wireless networks at the same time [3]–[5]. Multipath TCP (MPTCP) [3] designs a sub-layer above the TCP layer to distribute the traffic of a TCP stream over multiple paths (on different networks) for bandwidth aggregation. The WiRover system [4] designs a controller to handle packets from different wireless networks so that all networks can be exploited for data transfer. The work in [5] offloads delay-tolerant traffic to WiFi to augment the capacity of 3G network to time sensitive traffic.

The works in [6]–[10] further support transparent mobility at the TCP/UDP connection level in heterogeneous wireless networks. The work in [6] designs a control function for each network layer and a new control middleware to support core functionalities such as mobility, multi-homing and multi-path. ECCP [7] is an end-to-end connection control protocol that uses unique flow IDs to identify the flows between two end hosts. The authors in [8] use SDN on the mobile device to modify source/destination IPs so that a flow can be transparently migrated between wireless networks. The work of [9] proposes to either wait till the old flow ends and starts a new flow on the new network or design an agent to identify and resume migrated flows. In IFOM and IPMP [10], a device's traffic, though through different networks, is gathered at a home agent (HA) that binds its current addresses to its permanent address. Then, a connection can switch between different networks transparently.

B. SDN in Wireless Networks

The work in [11] uses SDN to manage the packet forwarding from mobile devices to the gateway (over different wireless networks) for transparent vertical handover. SoftRAN [12] designs a centralized controller to abstract all base stations in a local area as a virtual big base station, thus realizing more efficient radio resource usage and interference control. meSDN [13] extends the control of SDN to mobile devices. Then, the wireless connection to APs can be better utilized for functions such as WLAN virtualization, application-awareness, E2E QoS and network troubleshooting. The work in [14] summarizes challenges and benefits of adopting SDN to manage the cellular network, as well as SDN extensions needed to enable software-defined cellular networks.

In the work of [15], the concept of SDN is extended to wireless personal area networks. It uses SDN to define various rules dynamically to handle packets between different body area devices. MobileFlow [16] introduces an SDN based framework for mobile networks that decouples the data forwarding and control. It also introduces how applications can be realized in such a framework. Odin [17] and OpenSDWN [18] exploit SDN to improve wireless access performance and service differentiation in enterprise and home WiFi networks.

III. System Design

In this section, we present the detail of each technical component. We define the TCP/UDP connection that a packet belongs to as the packet's **host connection**. We use OpenFlow [19] as the SDN protocol for illustration.

A. Network Needs Description

We represent a TCP/UDP connection's network needs as a profile describing the requirement on various properties. For illustration purpose, we identify two properties (i.e., data rate and cost) in this paper. The data rate property has two requirements: high and low, which means that the connection needs a high data rate or can accept a low data rate, respectively. The cost property also has two requirements: expensive and economical, which means that the device owner is willing to use an expensive network or only wants an economical network for the connection, respectively. Consequently, there are 4 profiles (Pr_a, Pr_b, Pr_c, Pr_d) as shown in Table I.

TABLE I: Network Needs Profile Example.

Network Needs Profile		Data Rate	
		High	Low
Cost	Expensive	Pr_a	Pr_b
	Economical	Pr_c	Pr_d

TABLE II: Network Needs Profile Value.

Profile \ Network	WiFi	RelayNet	LTE
Pr_a	1	0	1
Pr_b	1	1	1
Pr_c	1	0	0
Pr_d	1	1	0

Each profile is further mapped to a binary value to show networks that fall into its category. Each bit of the binary value represents a network, and a bit is set to 1 when the network belongs to the profile and 0 otherwise. We place networks in the decreasing order of the preferability. For example, suppose the preferability follows: WiFi $>$ RelayNet $>$ LTE, where RelayNet is the network bridged through nearby devices. Then, the three bits of the binary value of a network needs profile represent WiFi, RelayNet, and LTE from left to right.

In this paper, the more **economical** a network is, the more preferable it is to the device user. Following this definition, the binary value of each network needs profile in Table I can be determined as shown in Table II. With such a design, applications determine the network needs profile of their TCP/UDP connections, which will be used in subsequent intelligent network selection.

B. SDN based Intelligent Network Selection

The SDN based intelligent network selection function (SDN-NS) enables intelligent network selection for TCP/UDP connections. It needs to solve two challenges. First, it cannot change the behavior of applications except for specifying network needs. Second, it can respond to the changes on network needs, network status, and scheduling commands quickly. We exploit SDN to solve both challenges.

To solve the first challenge, we exploit the concept of virtual switch, i.e., Open vSwitch (OVS), to offer a virtual interface for applications, as shown in Figure 2. Actual network interfaces are associated with the virtual switch to enable applications to forward and receive packets through actual networks. Application only see and use the virtual interface

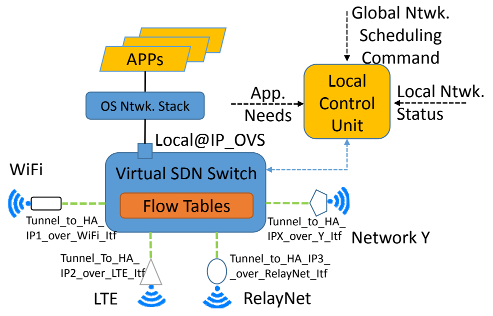


Fig. 2: SDN-NS inside a mobile device.

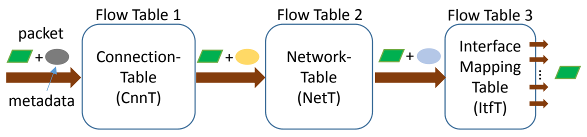


Fig. 3: Pipeline processing with three flow tables.

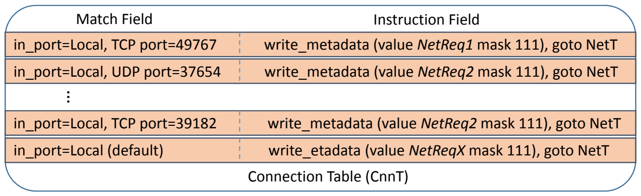
for transferring its data as if using a normal network interface. The detail of the OVS is introduced in Section III-B1.

We use the pipeline processing in the data plane in OVS to solve the second challenge, as illustrated in Figure 3. Each incoming packet from applications is associated with a meta-data to go through three tables. The connection table (CnnT) modifies the packet's metadata to reflect its host connection's network needs. Then, the network table (NetT) modifies the metadata to further select available and allowed networks from those after the connection table. Finally, the interface mapping table (ItfT) maps the packet to the most preferable interface among those after the network table. Such a design makes our design robust and easy for maintenance. The change on network needs, available/allowed networks, or interface mapping only needs to be updated to the corresponding table while keeping others unchanged. The details on the three flow tables are introduced in Sections III-B2 to III-B5.

SDN-NS also includes a local control unit that works directly on the OVS. It is responsible for flow entry update inside the mobile device based on input information.

1) Virtual Switch: An OVS is installed inside the mobile device to realize major functions of SDN-NS, as shown in Figure 2. With SDN-NS, all applications only use the virtual interface offered by the OVS. This offers a good isolation between applications and actual wireless networks. However, it leads to a challenge on how to forward application packets through the selected network. This is because all application packets take the IP of the virtual interface (i.e., a local IP) as the source IP and thus cannot go through the actual wireless network directly. We solve this problem by building tunnels.

Specifically, we associate each wireless interface with a tunnel port on OVS, as shown in Figure 2. The other end point of the tunnel is an interface on the device's home agent (HA). Note that all packets from a device are aggregated at the HA for transparent mobility, as introduced later in Section III-C. Thus, packets arriving at the port will be forwarded through the associated interface as tunnel packets towards the HA. The details about tunnel construction are introduced in Section III-C2.



Match Field	Instruction Field
in_port=Local, TCP port=49767	write_metadata (value *NetReq1* mask 111), goto NetT
in_port=Local, UDP port=37654	write_metadata (value *NetReq2* mask 111), goto NetT
⋮	
in_port=Local, TCP port=39182	write_metadata (value *NetReq2* mask 111), goto NetT
in_port=Local (default)	write_etadata (value *NetReqX* mask 111), goto NetT

Connection Table (CnnT)

Fig. 4: Connection table example.

2) Packet Metadata: The metadata associated with a packet reflects usable networks for the packet in the pipeline processing. Thus, we follow the design of the binary value of a network needs profile (in Section III-A) to determine its structure. It uses a bit to represent one network, and 1 means the network is usable and 0 not. In this paper, we set metadata to 3 bits (WiFi, RelayNet, and LTE).

We adopt the metadata register in OpenFlow as the packet metadata. As the OVS processes packets sequentially, the metadata actually is only associated with the packet under processing at a time. It supports the below bitwise update

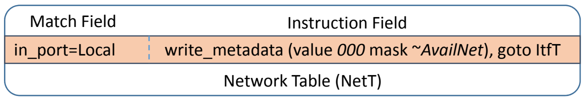
$$mdata\_new = mdata \;\&\; (\sim mask) \mid value \;\&\; mask \quad (1)$$

where mdata represents the metadata. It means that the bits of mdata indicated in $mask$ are set to $value$.

3) Connection Table (CnnT): The connection table (CnnT) is built to write each TCP/UDP connection's network needs into the metadata of its packets. Figure 4 illustrates the connection table in which each row represents the flow entry for one TCP/UDP connection. For the metadata update operation in each flow entry, the value is the binary value of the connection's network needs profile (denoted by *NetReq*) and the mask is set to 111. The update operation follows Equation (1) to get $mdata = NetReq$, i.e., setting the metadata to the connection's network needs. As shown in the figure, we also design a default flow entry for connections that use the default network needs.

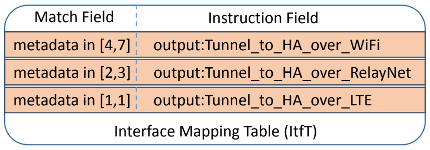
When an application is about to launch a TCP/UDP connection, it first notifies the local control unit the TCP/UDP port to be used and the connection's network needs profile. The control unit then inserts a flow entry for the connection in the CnnT table as described above. If one such entry already exists, it will be overwritten. Applications that cannot notify the local control unit can also work in HetSDN by using the default flow entry in the connection table.

4) Network Table (NetT): The CnnT table modifies the metadata to reflect networks that can satisfy the needs of the packet's host connection. Then, the network table (NetT) further modifies the metadata to reflect networks that are available (from local network status) and are allowed to use (from global scheduling commands) among those indicated in the first step. The NetT table has only one entry, as shown in Figure 5. In the instruction field, the value is 000 and the mask is the negative of available and allowed wireless networks (denoted ∼*AvailNet*). Then, by applying to Equation (1), we get $mdata\_new = NetReq \& AvailNet$. Note that the initial metadata value in this step is the *NetReq* resulted from the CnnT table. Consequently, the metadata is set to a binary value that reflects networks that are available and allowed to use and can satisfy the host connection's needs.



Match Field	Instruction Field
in_port=Local	write_metadata (value *000* mask *~AvailNet*), goto ItfT
Network Table (NetT)	

Fig. 5: Network table example.



Match Field	Instruction Field
metadata in [4,7]	output:Tunnel_to_HA_over_WiFi
metadata in [2,3]	output:Tunnel_to_HA_over_RelayNet
metadata in [1,1]	output:Tunnel_to_HA_over_LTE
Interface Mapping Table (ItfT)	

Fig. 6: Interface mapping table example.

5) Interface Mapping Table (ItfT): Finally, the interface mapping table (ItfT) forwards the packet to the most preferable network interface from those selected after the first two tables. Recall that the more significant a bit is, the more preferable the network is. Thus, the ItfT table should forward the packet to the network represented by the most significant 1 in its metadata (i.e., the most left 1). For example, the metadata 101 (i.e., both WiFi and LTE are usable) means that the packet should be forwarded to WiFi, which is represented by the most left 1. We thereby design the ItfT table as shown in Figure 6. In the figure, packets with metadata in range [4,7] are forwarded to the WiFi interface. This because in this case, the most significant bit of the metadata (i.e., WiFi) is always 1. Other flow entries follow the same rationale.

It is possible that the metadata of a packet is 000 after the first two tables, i.e., no usable network. In this case, the packet is forwarded to the local control unit, which notifies the corresponding application for further handling. How such an exception is handled is out of the topic of this paper.

6) Handling Reply Packets: We also design one flow entry for each wireless interface to forward received packets to applications through the local port of the OVS bridge. Finally, applications just send and receive packets through the OVS interface. This ensures the compatibility of HetSDN.

C. SDN based Intelligent Home Agent

Figure 7 shows the overall structure of SDN-HA. It includes a virtual SDN switch and a NAT. They work together to support transparent TCP/UDP migration, which is introduced in Section III-C1. Mobile devices reach the HA through tunnels, which is introduced in Section III-C2. The details about how to automatically know which wireless network should be used to forward reply packets to the mobile device are described in Section III-C3.

1) Supporting Transparent TCP/UDP Connection Mobility: To enable TCP/UDP connections to migrate transparently, HetSDN requires all application traffic be anchored at the SDN-HA and then translated by the NAT as if originated from the HA. Therefore, when a migration happens, the corresponding host is not aware of such a change. For example, when a TCP/UDP connection migrates from WiFi to LTE, the corresponding host cannot see a change on received packets since all packets still take the IP of the NAT as the source IP. Thus, the connection is not affected after the change, which means that the migration is transparent.

2) Tunnels between SDN-HA and Mobile Device: As mentioned in Section III-B1, a mobile device creates tunnels

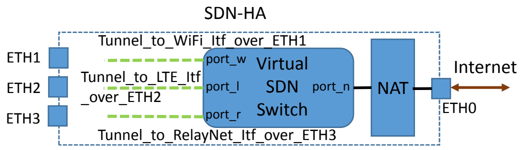

SDN-HA
Tunnel_to_WiFi_Itf_over_ETH1
ETH1
ETH2
ETH3
Tunnel_to_LTE_Itf _over_ETH2
port_w
port_l
port_r
Virtual SDN Switch
port_n
NAT
Internet
ETH0
Tunnel_to_RelayNet_Itf_over_ETH3

Fig. 7: SDN based intelligent HA (SDN-HA).

over different wireless interfaces to its HA to forward packets to the HA. Thus, corresponding tunnels are built on the virtual SDN switch inside the SDN-HA to 1) decapsulate received tunnel packets and 2) send reply packets back through the wireless network its host connection currently uses. Figure 7 shows the tunnel configuration on the SDN-HA.

While the HA can easily have multiple public IPs, the wireless interface on a mobile device may have a public IP or a private IP (i.e., inside a NAT). In the former case, the tunnel is built directly. In the latter case, the private IP address cannot be used as the remote end of a tunnel. We solve this problem by setting the remote end as the NAT of the mobile device and rely on the NAT to relay tunnel packets. Our experiment shows that the NAT of a LTE network can rely tunnel packets to the mobile device correctly.

3) SDN based Intelligent Packet Relay: For packets arriving at the SDN switch of the SDN-HA, we define incoming packets as packets sent out by the mobile device, and reply packets as those destined to the mobile device. As shown in Figure 7, incoming packets just need to be forwarded to the NAT (i.e., to $port\_n$). However, the challenge is how to forward reply packets back through the wireless network that is currently used by the host TCP/UDP connection.

We solve this challenge by extracting necessary information from incoming packets to build flow entries to guide reply packets. Figure 8 shows the process of our method. Suppose a packet of a connection arrives at port $port\_w$ with source IP ip_ovs and source TCP port tpx (step 1). Then, if there is a matched flow entry, the packet is forwarded accordingly (step 2.1). This means that flow entries for this connection (both incoming and reply packets) have already been built. Otherwise, we need to build flow entries for this connection's incoming and reply packets. Specifically, due to a miss, the packet is forwarded to the HA controller (step 2.2). The controller first removes all flow entries for the connection, if exist, and then inserts two new flow entries (steps 3.1 and 3.2), one for incoming packets and one for reply packets. The two flow entries guide subsequent incoming packets to port $port\_n$ (to NAT) and reply packets to port $port\_w$ (to reach mobile device through the tunnel over WiFi), respectively (4.1 and 4.2). The flow entry for reply packets matches with packets destined to ip_ovs at TCP port tpx and forwards matched packets to $port\_w$, as shown in Figure 9.

We can see that in the above design, the SDN-HA infers the information needed to forward reply packets through the correct wireless network automatically, without the need of explicit report from either mobile devices or APs. Such a design also does not compromise the transparent connection migration. When a migration happens, incoming packets enter the SDN switch through another port. This will make previous flow entries fail to match with the incoming packets (since

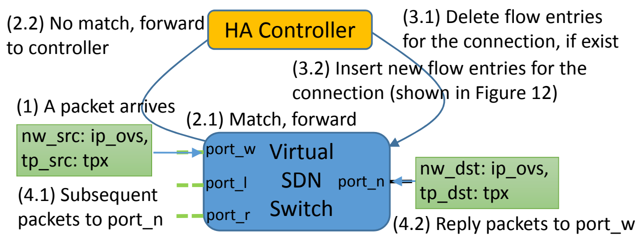
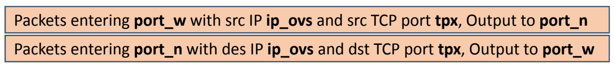

(2.2) No match, forward to controller
HA Controller
(3.1) Delete flow entries for the connection, if exist
(3.2) Insert new flow entries for the connection (shown in Figure 12)
(1) A packet arrives
(2.1) Match, forward
nw_src: ip_ovs, tp_src: tpx
port_w
Virtual SDN Switch
port_l
port_r
port_n
nw_dst: ip_ovs, tp_dst: tpx
(4.1) Subsequent packets to port_n
(4.2) Reply packets to port_w

Fig. 8: Illustration of intelligent packet relay.



Packets entering **port_w** with src IP **ip_ovs** and src TCP port **tpx**, Output to **port_n**

Packets entering **port_n** with des IP **ip_ovs** and dst TCP port **tpx**, Output to **port_w**

Fig. 9: Inserted flow entries for a new connection.

we also match on the ingress port as shown in Figure 9). Consequently, steps 1 to 3 will be executed to remove old flow entries for the connection and insert new ones.

IV. Performance Evaluation

We implemented a simple HetSDN system on our campus to demonstrate its effectiveness and efficiency. The deployment is not complex and does not need to change any existing infrastructures. Figure 10 illustrates the overall structure of our deployment. We used a laptop with Ubuntu 14.04 as the SDN-HA in the test, which is located in our campus network. Mobile devices are the same type of laptops. Open vSwitch is used as the virtual SDN switch and Floodlight [20] is used as the SDN controller. On the mobile device, static flow pusher API of Floodlight is used to update flow entries in the three tables (Figure 3). On the SDN-HA, the intelligent packet relay function (Section III-C3) is realized as a module of the Floodlight with less than 500 lines of code, and the NAT is simply realized by postrouting IP forwarding.

In this test, we used three wireless networks: WiFi, RelayNet, and LTE. Their preferability follows: WiFi > RelayNet > LTE. There are commercial LTE and university WiFi signals on our campus. The RelayNet is simulated by the campus WiFi network by 1) limiting its coverage to our lab; 2) limiting the data rate to 6 Mbps; and 3) adding additional 20ms delay. Those limitations reflect the relaying through another device. The tunnel between the SDN-HA and the WiFi/RelayNet interface is built directly since both are in our campus network. The tunnel between the SDN-HA and the LTE interface is relayed by the NAT. We found that the LTE NAT can correctly relay GRE tunnel packets in our experiments.

A. Supporting the Design Goal

We evaluate how HetSDN supports the design goal: intelligent network selection and transparent mobility at the TCP/UDP connection level. We developed an example application for test. It echoes to a remote server (send a packet, wait for a reply, and send again) through UDP/TCP. Whenever a new connection is started or its network needs change, it talks to the local control unit to update its flow entries in the CnnT table, which requires only a few lines of code.

We use NetReq and AvailNet to represent the network needs of a connection and available networks, respectively. Thus, NetReq=5 (101) means that WiFi and LTE satisfy the connection's network needs, while AvailNet=6 (110) means

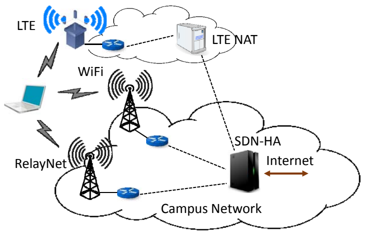


Fig. 10: HetSDN deployment on our campus.

that only WiFi and RelayNet are available. In each test, we change the connection's network needs and available networks intensively through software signalling to show HetSDN's ability. Such a scheme actually is more challenging for mobile devices since the network availability changes more frequently. An echo is regarded as failed if no reply is received after 2 seconds. The round trip time (RTT) of an echo is recorded to identify which network it actually uses.

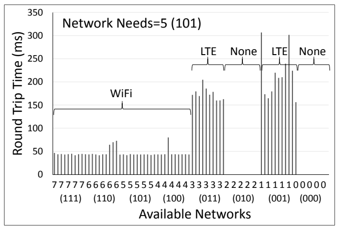
1) Change Available Networks: We first fix NetReq to 5 (i.e., only WiFi and LTE can satisfy the needs) and 6 (i.e., only WiFi and RelayNet can satisfy the needs) and change AvailNet from 7 to 0 (i.e., "all available" to "none is available"). The AvailNet decreases 1 after every 10 echoes. The RTTs of each echo using the UDP and TCP are shown in Figures 11(a) and 11(b) and Figures 12(a) and 12(b), respectively.

From the four figures, we find that both the UDP connection and the TCP connection select the network based on available networks dynamically and migrate transparently. Firstly, in this test, packets are always forwarded through the most preferable one among available networks. For example, in Figures 11(a) and 12(a), when AvailNet is larger than 3, which means that WiFi exists anyway, the connection only chooses WiFi since it is the most preferable one. However, when AvailNet decreases to 3 (i.e., RelayNet and LTE are available), the connection immediately chooses the LTE since only LTE satisfies the requirement. Such a phenomenon can also be found in Figures 11(b) and 12(b).

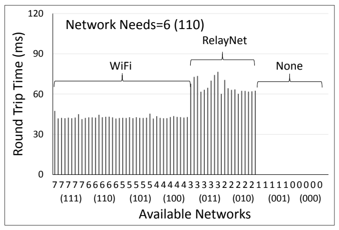
Secondly, we did not observe any errors during the test, and all echoes are continuous. This means that the connection is not aware of the network change but only shows different RTTs on its echoes. Thus, the migration is transparent without disconnecting the TCP/UDP connection.

2) Change Network Needs: We then fix AvailNet to 5 (i.e., WiFi and LTE are available) and 7 (i.e., all networks are available) and change NetReq from 7 to 1 (i.e., "all networks can satisfy the needs" to "only LTE can satisfy the needs"). The NetReq decreases 1 after every 10 echoes. The RTTs of each echo using the UDP and TCP are shown in Figures 11(c) and 11(d) and Figures 12(c) and 12(d), respectively.

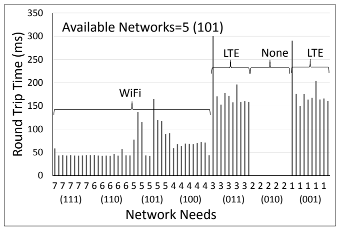
We find that the results in this test are consistent with the previous test. Firstly, both the UDP connection and the TCP connection migrate between networks based on its network needs dynamically and transparently. For example, as shown in Figures 11(d) and 12(d), when all networks are available (NetAvail=7), the connection always choose the most preferable one based on its network needs. When WiFi can satisfy the needs (i.e., NetReq>3), it always chooses the WiFi. Otherwise, if the RelayNet can satisfy the needs (NetReq=3 or 2), it



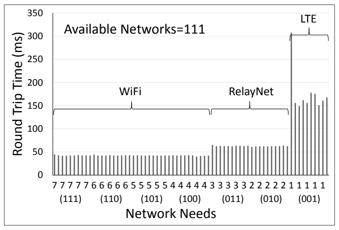
(a) NetReq=5 (101).



(b) NetReq=6 (110).

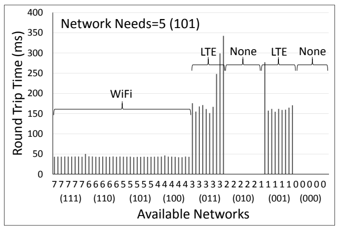


(c) AvailNet=5 (101).

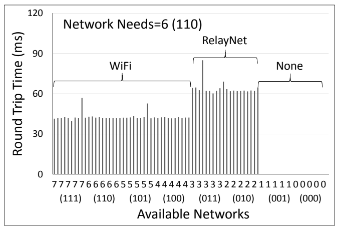


(d) AvailNet=7 (111).

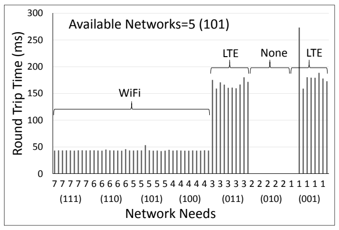
Fig. 11: RTT to the remote server with UDP.



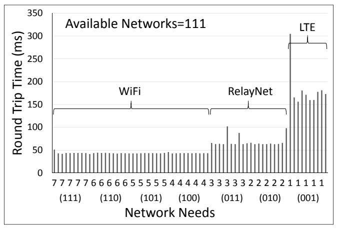
(a) NetReq=5 (101).



(b) NetReq=6 (110).



(c) AvailNet=5 (101).



(d) AvailNet=7 (111).

Fig. 12: RTT to the remote server with TCP.

only uses the RelayNet. If only the LTE can satisfy the needs (NetReq=1), it switches to the LTE. Such a phenomenon can also be found in Figures 11(c) and 12(c). Secondly, we find that the migration between different networks is transparent without incurring any errors or disconnection.

Furthermore, in both tests, the change of NetReq or AvailNet only needs to update relevant flow entries in the CnnT table or the NetT table without affecting others. This shows the high efficiency of the SDN-NS.

Combining all above results, we conclude that the proposed HetSDN system can enable a TCP/UDP connection to dynamically select the most suitable network and transparently migrate between networks based on its network needs, local network status, and global network scheduling commands.

V. Conclusion and Future Work

In this paper, we propose an SDN-based system, namely HetSDN, to enable intelligent network usage for mobile devices in a heterogeneous network environment. HetSDN consists of two components: SDN-NS and SDN-HA, both of which novelly exploit the flexible control over packet forwarding provided by SDN to fulfill their tasks. The SDN-NS uses a virtual SDN switch to dynamically guide application packets to the most suitable wireless interface. The SDN-HA anchors all traffic for the mobile device. It also uses SDN to intelligently guide reply packets back to the device through the network currently used by their host TCP/UDP connections. Finally, a mobile device can dynamically use the most suitable network and transparently migrate between different networks at the TCP/UDP connection level. The deployment on our campus demonstrates that HetSDN can efficiently realize the design goal. In the future, we plan to deploy HetSDN in a larger scale and investigate how to leverage the global controller for overall network access optimization.

References

[1] M. Faezipour, M. Nourani, A. Saeed, and S. Addepalli, "Progress and challenges in intelligent vehicle area networks," *Communications of the ACM*, vol. 55, no. 2, pp. 90–100, 2012.

[2] H. Kopetz, "Internet of things," in *Real-time systems*. Springer, 2011, pp. 307–323.

[3] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," *IETF RFC 6824*, 2013.

[4] J. Hare, L. Hartung, and S. Banerjee, "Beyond deployments and testbeds: experiences with public usage on vehicular wifi hotspots," in *Proc. of MobiSys*, 2012.

[5] aruna balasubramanian, ratul mahajan, and A. Venkataramani, "Augmenting mobile 3g with wifi," in *Proc. of MobiSys*, 2010.

[6] A. Singh, G. Ormazabal, H. Schulzrinne, Y. Zou, P. Thermos, and S. Addepalli, "Unified heterogeneous networking design," in *Proc. of IPTComm*, 2013.

[7] M. Arye, E. Nordstrom, R. Kiefer, J. Rexford, and M. J. Freedman, "A formally-verified migration protocol for mobile, multi-homed hosts," in *Proc. of ICNP*, 2012.

[8] 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," in *Proc. of CellNet*, 2012.

[9] A. Rahmati, C. Shepard, C. Tossell, A. Nicoara, L. Zhong, P. Kortum, and J. Singh, "Seamless TCP migration on smartphones without network support," *IEEE TMC*, 2014.

[10] C. Sankaran, "Data offloading techniques in 3GPP Rel-10 networks: A tutorial," *Communications Magazine, IEEE*, vol. 50, no. 6, pp. 46–53, 2012.

[11] R. Izard, A. Hodges, J. Liu, J. Martin, K.-C. Wang, and K. Xu, "An openflow testbed for the evaluation of vertical handover decision algorithms in heterogeneous wireless networks," in *Proc. of TRIDENTCOM*, 2014.

[12] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software defined radio access network," in *Proc. of HotSDN*, 2013.

[13] J. Lee, M. Uddin, J. Tourrilhes, S. Sen, S. Banerjee, M. Arndt, K.-H. Kim, and T. Nadeem, "meSDN: Mobile Extension of SDN," in *Proc. of MCS*, 2014.

[14] L. E. Li, Z. M. Mao, and J. Rexford, "Toward software-defined cellular networks," in *Proc. of EWSDN*, 2012.

[15] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling SDNs," in *Proc. of EWSDN*, 2012.

[16] K. Pentikousis, Y. Wang, and W. Hu, "MobileFlow: Toward software-defined mobile networks," *Communications Magazine, IEEE*, vol. 51, no. 7, 2013.

[17] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise wlans with odin," in *Proc. of HotSDN*, 2012.

[18] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann, "OpenSDWN: Programmatic control over home and enterprise WiFi," in *Proc. of SOSR*, 2015.

[19] "Openflow," https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[20] "Floodlight," http://www.projectfloodlight.org/floodlight/.