Implementation of PHY Rate and A-MPDU Length Adaptation Algorithm on WiSHFUL Framework

Changmok Yang[†], Seongho Byeon[†], Peter Ruckebusch[‡], Spilios Giannoulis[‡], Ingrid Moerman[‡], and Sunghyun Choi[†]
[†]Department of ECE and INMC, Seoul National University, Seoul, Korea
[‡]Ghent University, IDLab, Gent, Belgium
{cmyang, shbyeon}@mwnl.snu.ac.kr, schoi@snu.ac.kr
{peter.ruckebusch, spilios.giannoulis, ingrid.moerman}@ugent.be

***Abstract*—Research on a new solution supporting low latency, high reliability, and scalability is required to deal with ever-increasing demand for wireless communication. However, it is often restricted due to the fact that setting up experiments needs a considerable amount of effort and cost. To alleviate such difficulties, the WiSHFUL project has been established, which proposes an architecture for flexible and unified control of the wireless systems and platforms. The WiSHFUL architecture enables exploiting existing control knobs in a unified manner by offering platform-independent programming interfaces on top of heterogeneous hardware platforms. To illustrate the strength of the WiSHFUL architecture, we implement the existing wireless local area network (WLAN) performance enhancement algorithm, called *STRALE*, which adapts PHY rates and frame aggregation length for performance enhancement in mobile environments, using unified programming interfaces (UPIs) on the open-source software platform of WiSHFUL. Accordingly, *STRALE* is extended to be available on all devices compatible with WiSHFUL, showing the convenience and benefit of using the WiSHFUL platform.**

I. Introduction

The ever-increasing demand for wireless communication pushes researchers to devise novel solutions that support low latency, high reliability, and scalability. Experimental evaluation of these solutions on testbed infrastructure is required to validate their performance. However, setting up such experiments requires a considerable amount of effort and cost. For example, while research efforts based on software defined radios (SDRs), such as USRP, WARP, and Sora, have been carried out recently [1–3], a considerable portion of spent time and cost is consumed for auxiliary work since each platform provides different interfaces.

To alleviate such difficulties, the WiSHFUL project has been established [4] that proposes an architecture for flexible and unified control of wireless systems and platforms.[1] The WiSHFUL architecture enables exploiting existing control knobs in a unified manner by offering platform-independent programming interfaces on top of heterogeneous hardware platforms. Ultimately, WiSHFUL aims to reduce the threshold for implementation and experimentation in view of stimulating wireless innovation, and increases the realism of experimentation.

[1]WiSHFUL stands for **Wi**reless **S**oftware and **H**ardware platform for **F**lexible and **U**nified radio and network contro**L**.

An open source implementation of the WiSHFUL architecture is available online [5]. The implementation offers unified programming interfaces (UPIs) for radio and network control [6] so that experiments could be staged and conducted easily based on different platforms. Since the UPIs are device-independent, it is possible to devise an algorithm for a specific set of devices and then reuse the same algorithm on a different set of devices. Moreover, due to the modular nature of the architecture, supporting a new type of device only requires to create a device module that implements the defined UPIs. WiSHFUL currently supports multiple widely used technologies (e.g., IEEE 802.11, IEEE 802.15.4, and LTE), operating systems (e.g., Linux, Windows, and Contiki) and radio platforms (e.g., Atheros, Broadcom, CC2520, and Xylink Zynq), as well as advanced reconfigurable radio systems (e.g., IRIS, GNURadio, WMP, and TAISC) [7–10].

To illustrate the convenience and benefit of the WiSHFUL architecture, we have implemented *STRALE* [11], that is proposed to enhance the performance of IEEE 802.11 wireless local area network (WLAN) in mobile environments, using UPIs. *STRALE* adapts the physical (PHY) rate and frame aggregation length depending on the degree of mobility, and it complies with the 802.11 medium access control (MAC) without PHY layer modification. Originally, it had been fully implemented in *ath9k* device driver [12] and, consequently, can only be used on Atheros WLAN network interface cards (NICs). By employing UPI functions to implement *STRALE* on top of WiSHFUL, however, it becomes available to be used on any device that is supported by WiSHFUL.

Porting *STRALE* to WiSHFUL only requires to add the necessary UPI functions for controlling PHY rates and frame aggregation, and to extend the device module for *ath9k* with the new UPI functions. For this purpose, *STRALE* is refactored by separating the adaptation part from the device driver and implementing it within the WiSHFUL device module, where the parameters between the device module and the device driver are exchanged via *debugfs* [13]. To investigate the impact of the delay caused by this redesign and *debugfs*, we also evaluate the performance of *STRALE* newly ported to WiSHFUL by using the original implementation as baseline and comparing between the two. The result shows that there is only a 3.1% difference in throughput, which is acceptable

given the increased flexibility.

The rest of the paper is organized as follows. Section II provides the preliminaries of IEEE 802.11 system and *STRALE*. Section III introduces the WiSHFUL architecture. The implementation of *STRALE* in WiSHFUL is described in Section IV, and Section V presents the performance evaluation and demonstration results. Finally, Section VI concludes the paper.

II. Preliminaries

A. Aggregated MAC protocol data unit (A-MPDU)

In order to enhance MAC efficiency, IEEE 802.11n/ac defines frame aggregation, called aggregate MAC protocol data unit (A-MPDU), which enables aggregating multiple MPDUs into a single frame [14, 15]. Using A-MPDU, a station transmits several MPDUs in a single transmission attempt, thus, PHY/MAC protocol overhead, such as the time for PHY preamble transmission and contention including several inter frame spaces, can be amortized over multiple MPDUs. Additionally, the individual subframes (i.e., MPDUs) can be positively and negatively acknowledged by using block acknowledgement (BlockAck), and hence, the transmitter can selectively retransmit the failed subframes in the next A-MPDU transmission [14].

A-MPDU length is constrained for fair sharing of the wireless medium among all stations. The maximum A-MPDU length is 65,535 bytes and 1,048,575 bytes for the 802.11n and 802.11ac, respectively. A-MPDU length is limited by the transmission time as well. The maximum A-MPDU transmission time is 10 ms and 5.484 ms for the 802.11n and 802.11ac, respectively.

B. STRALE: A standard-compliant and mobility-aware PHY rate and A-MPDU length adaptation algorithm

A transmitted signal undergoes distortion such as pathloss, fading, and noise. Therefore, in order to receive the signal correctly, channel estimation needs to be conducted to obtain current channel state information (CSI). In 802.11 system, channel estimation is performed using training symbols in the physical layer convergence protocol (PLCP) preamble at the beginning of a frame [14, 15]. This channel estimation is valid with an assumption that the CSI does not change during the packet reception. However, if the transmitted frame length is long compared with the channel coherence time, e.g., when A-MPDU is used, the CSI obtained at the beginning of the frame may not be valid to decode the latter part of the frame. Especially, when the A-MPDU is transmitted in mobile environments since the CSI rapidly changes, the latter subframes of the A-MPDU suffer from high subframe error ratio (SFER), thus resulting in severe throughput degradation. This phenomenon is referred to as caudal loss [11, 16, 17].

To resolve the caudal loss problem, *STRALE*, a mobility-aware PHY rate and frame length adaptation algorithm, is proposed [11]. *STRALE* is designed to comply with the 802.11 MAC standard, without any requirements for PHY layer modification.

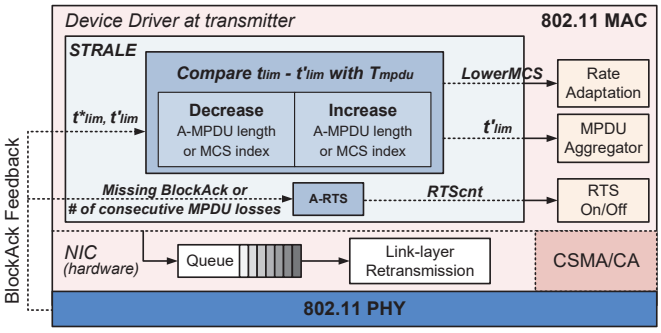


Fig. 1. Implementation structure of *STRALE*. *STRALE* is implemented only on *ath9k* device driver at one end of the wireless link (i.e., transmitter).

STRALE implements two major functions: 1) estimating the degree of mobility and 2) PHY rate or A-MPDU length adaptation. The degree of mobility of a station is estimated based on a transmission result of an A-MPDU, i.e., BlockAck content. *STRALE* computes the optimal A-MPDU length for the previous A-MPDU transmission (t^*_{lim}) using the received BlockAck, and the time limit of the next A-MPDU (t'_{lim}) is obtained statistically. If the difference between t'_{lim} and the previous A-MPDU transmission time (t_{lim}) is larger than the transmission time of a single MPDU (T_{mpdu}), *STRALE* judges that the degree of mobility is quite high and attempts to lower PHY rate or A-MPDU length. In this case, *STRALE* decreases PHY rate one level lower if $TP(r_-, t_{lim}) > TP(r, t'_{lim})$, where $TP(r, t_{lim})$ is the estimated throughput using r and t_{lim}, and r_- is the PHY rate one lever lower than r, respectively. The lowered PHY rate by *STRALE* is tracked using $LowerMCS$ flag. If $TP(r_-, t_{lim}) < TP(r, t'_{lim})$, however, the A-MPDU length is decreased from t_{lim} to t'_{lim}. On the other hand, if the difference between t'_{lim} and t_{lim} is shorter than T_{mpdu}, it is considered that the current channel condition is good enough to support a longer A-MPDU. In this case, *STRALE* increases PHY rate one level higher, only if the PHY rate was already lowered previously, as tracked by $LowerMCS$. If not, t'_{lim} is recalculated (larger than t_{lim}) and the length of the next A-MPDU is increased.

Fig. 1 depicts the operation of *STRALE* with a blueprint.[2] Although *STRALE* can be applicable to any existing hardware by simply modifying the accompanying device driver, prototype of *STRALE* is implemented only on *ath9k* device driver [12], while the source code of *STRALE* is opened to the public [18]. Moving forward, in this paper, we have extended *STRALE* to be available on top of any device supported by WiSHFUL by separating the adaptation algorithm from the device driver.

III. WiSHFUL Architecture

The WiSHFUL architecture enables flexible and unified control for wireless systems and platforms [4] by exposing

[2]In the original *STRALE*, adaptive request-to-send (A-RTS) is also proposed in order to resolve hidden node problem. In this work, however, the A-RTS part is omitted due to ease of implementation.

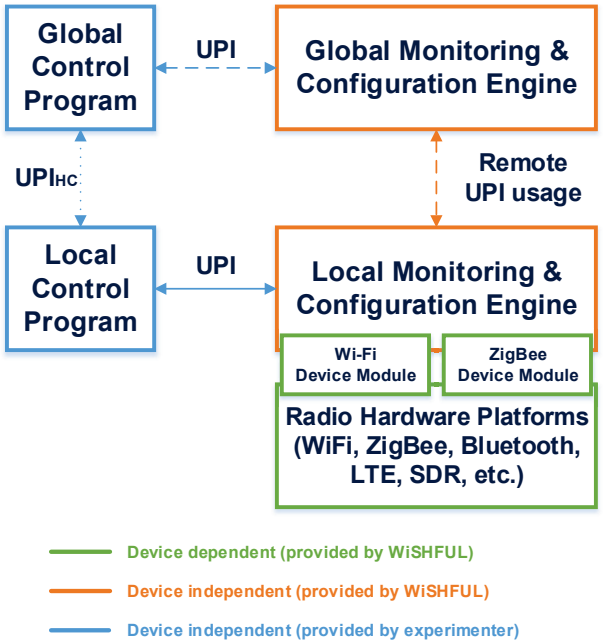


Fig. 2. Overview of the WiSHFUL architecture. The solid line and the dashed line represent local and remote mode, respectively. The dotted line indicates hierarchical control.

platform-independent programming interfaces over heterogeneous hardware platforms. In this regard, it significantly reduces the threshold for implementation and experimentation in view of stimulating wireless innovation.

A. Overview

Fig. 2 illustrates the WiSHFUL architecture. The device-independent control programs (CPs) implement the control logic for an experiment. They can operate on a single node (i.e., local control) or on a group of nodes (i.e., global control). The CPs control the networking behaviour by means of UPI calls, either local or remote mode. The UPI calls are delegated by the local or global monitoring & configuration engine (MCEs). The MCEs implement the core services of the framework device-independently. The device modules, which are device-dependent, translate the device-independent UPI calls to device specific application programming interface (API) calls. The WiSHFUL architecture also enables user-defined control flows between global and local CPs, i.e., hierarchical control (UPI_{HC}). Beside additional flexibility, hierarchical control allows experimenters to delegate time-sensitive operations to the local level, closer to the hardware, while maintaining a network-wide view on the global level.

B. Context-aware UPI execution

An open source implementation of the WiSHFUL objectives and architectures, discussed in Section III-A, is available online [5], called WiSHFUL Framework. WiSHFUL Framework is a software platform offering UPIs for radio and network control of wireless communication platforms.[3] WiSHFUL

[3]The framework is implemented based on Python, and the implemented code is split into Python modules for modularity.

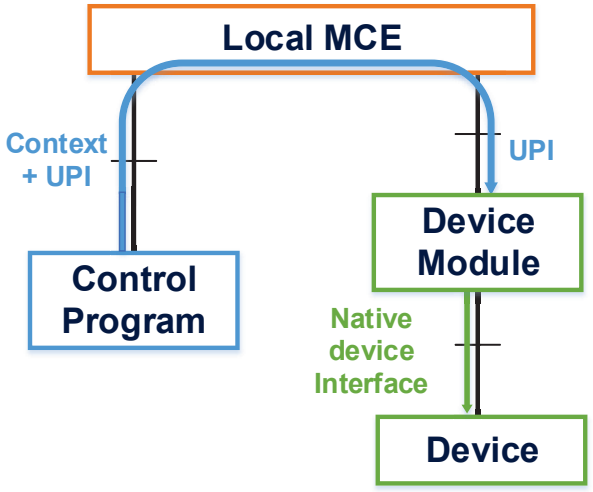


Fig. 3. Context-aware execution of UPI calls. The figure depicts a local control program executing a UPI call locally.

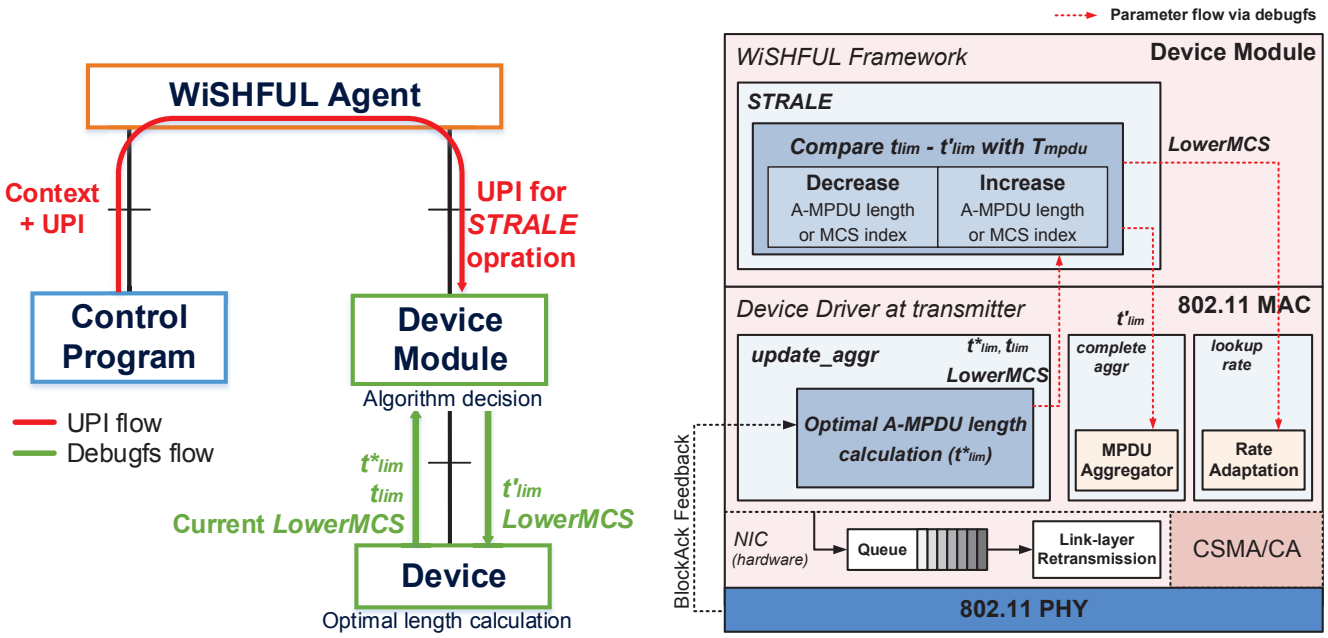
Framework offers: 1) flexibility of controlling configuration and operation of wireless communication networks, 2) simplified way for programming different devices and protocols, and 3) unified interfaces to hide hardware implementation specifics.

One of the main features of WiSHFUL Framework is the ability to attach a context to a UPI call. The context of the UPI call allows experimenters to exactly define *where*, *when*, and *how* the UPI function is executed:

- **Where:** on which devices and/or wireless interfaces does the UPI function need to be executed.
- **When:** at which time (absolute time reference or relative to the beginning of the procedure) does the UPI call need to be passed to the device module.
- **How:** is the UPI call blocking or non-blocking for the CP. In the latter case, a callback mechanism is used to receive the response.

Fig. 3 depicts how a control program delegates the UPI function together with a call context to the local MCE. The local MCE then passes the UPI function to the device module, taking into account the call context. The device module translates the generic UPI call to an API call in the native device's programming interface.

Note that the definition of UPIs is independent of the native device interfaces, thus reducing the complexity for an experimenter, because control logic can be created without requiring detailed information about the specific operation of the native device interfaces. For example, in consideration of changing the transmission power, a specific command to control the transmission power of a wireless networking interface is device-dependent, e.g., iwconfig command needed for Linux-based device and GNURadio control needed for USRP device. Thanks to the UPIs, however, the same function can be used to change the transmission power on both types of devices.



(a) UPIs and *debugfs* flow of *STRALE* on WiSHFUL Framework.

(b) Key operation of *STRALE* implementation on WiSHFUL Framework.

Fig. 4. *STRALE* implementation on WiSHFUL Framework. The control program commands a context through WiSHFUL agent using the UPIs, and the device module then receives and translates the UPIs for *ath9k* device driver. Additionally, the parameters between the device module and the device driver are exchanged via *debugfs*.

IV. *STRALE* Implementation on WiSHFUL Framework

In this section, we describe the detailed implementation of *STRALE* integrated into WiSHFUL Framework. At first, we newly define additional UPIs for *STRALE* operation to the existing set of UPI functions. The new UPIs contain the context of *STRALE* such as PHY rate and A-MPDU length update functions, and turning on/off *STRALE* algorithm. We then extend the device module which is responsible to translate the new UPIs to the native device programming interface calls.

Note that the original *STRALE* is fully implemented on *ath9k* device driver at one end of the wireless link, (i.e., transmitter). Specifically, after receiving BlockAck, the transmitter immediately determines A-MPDU length and PHY rate for the next A-MPDU transmission. Therefore, *STRALE* is basically per A-MPDU adaptation algorithm. However, since WiSHFUL Framework cannot obtain BlockAck information from PHY layer directly, an alternative way needs to be considered for the device module implementation.

Accordingly, we reform the structure of *STRALE* such that it can be implemented on WiSHFUL Framework. As a result, the time-critical operations such as t^*_{lim} calculation module remains at the device driver, but the adaptation algorithm is migrated to device module. To be specific, after receiving BlockAck, t^*_{lim}, t_{lim}, and the used $LowerMCS$ flag are sent from the device driver to the device module. Subsequently, a calculation of the next $LowerMCS$ and t'_{lim} is conducted in the device module, and then, the device driver uses $LowerMCS$ and t'_{lim} for the next A-MPDU transmission, updated from the device module.

We use *debugfs* [13] for exchanging parameters between the device module (user space) and the device driver (kernel space). Using *debugfs* allows to read/overwrite variables defined in the device driver, without source-code modification during run-time. Therefore, we register three parameters (i.e., t^*_{lim}, t_{lim}, and the used $LowerMCS$) as the input to the device module, and two parameters (i.e., t'_{lim} and the next $LowerMCS$) as the adaptation output, so that 1) the device module can read and write them directly, and simultaneously, 2) the device driver can use t'_{lim} and $LowerMCS$ for the next transmission.

Fig. 4 illustrates the detailed operation of *STRALE* on WiSHFUL Framework. The UPI and parameter flows via *debugfs* are demonstrated in Fig. 4(a). Red and green arrows indicate UPI and *debugfs* flows, respectively. The control program commands a context through WiSHFUL agent using the UPIs. The device module then receives and translates the UPIs for *ath9k* device driver. The operation on the device module is depicted in Fig. 4(b). The *STRALE* implementation can be divided into two blocks, i.e., WiSHFUL Framework and the device driver at the transmitter, where the parameter flows are drawn with red-dotted arrows.

V. Performance Evaluation

In this section, we evaluate the performance of *STRALE*, newly implemented on WiSHFUL Framework. We use Qualcomm Atheros AR9380 NIC along with HostAP [19] to build an AP on a Ubuntu 14.04 based machine. The AP employs *ath9k* device driver as a local device module, and WiSHFUL Framework is installed on top of the AP. As an associated station, we use commercial off-the-shelf table PC, Samsung

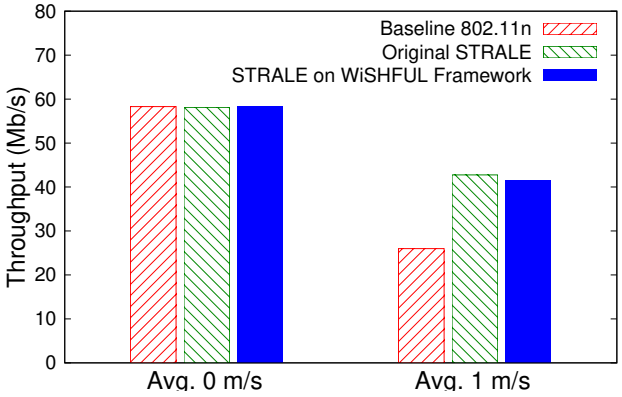


Fig. 5. Throughput results. *STRALE* on WiSHFUL Framework achieves 41.4 Mb/s, which is 3.1% lower than that of the original *STRALE*, when the station moves around the AP with an average speed of 1 m/s.

Galaxy Tab S 10.5. Note that *STRALE* is operated only on the transmitter, i.e., the AP, as explained in Section II-B.

A. Delay measurement

STRALE originally determines PHY rate and A-MPDU length for every individual A-MPDU. WiSHFUL Framework, however, is built in Python, such that the execution and processing speed might be slower than C-based ath9k device driver. Moreover, exchanging parameters between the device module and the device driver also causes an additional delay. To investigate the impact of the delay, we measure the entire delay from the time when BlockAck feedback is updated, to the moment when newly calculated PHY rate and A-MPDU length is actually used for the next A-MPDU transmission. The AP transmits saturated UDP traffic to the station, which moves around the AP at an average speed of 1 m/s, and the results are averaged over 30 s.

The measured delay is 2.769 ms with a standard deviation of 2.625 ms, while the average A-MPDU length is 3.082 ms. Unfortunately, after receiving BlockAck, PHY rate and A-MPDU length determined by *STRALE* cannot be used directly since the AP should immediately create an A-MPDU for the next transmission. Instead, they can be adopted after transmitting one or two A-MPDUs, which results in one-step behind PHY rate and A-MPDU length adaptation. Accordingly, the performance of *STRALE* newly implemented on WiSHFUL Framework might be degraded due to late response to the mobility fluctuation. An investigation of throughput reduction caused by one-step behind adaptation is presented in the following.

B. Performance evaluation of STRALE on WiSHFUL Framework

We conduct experiments to verify the successful implementation of *STRALE* on WiSHFUL Framework, and to investigate the impact of one-step behind PHY rate and A-MPDU length adaptation. Saturated UDP downlink traffic is generated from the AP to the station, where the average moving speed of the station is given to 1 m/s. The station is deployed in proximity to the AP, to minimize MPDU losses caused by the fading.

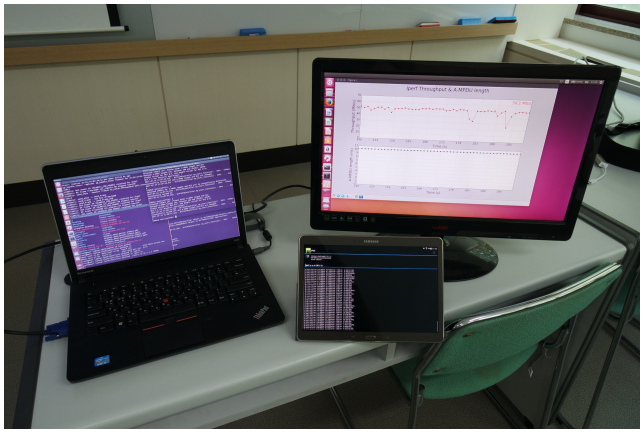


Fig. 6. A scene of demo presentation. The laptop is deployed as an AP, and the tablet PC is deployed as a station. The throughput and A-MPDU length are displayed using the visualization tool on the screen.

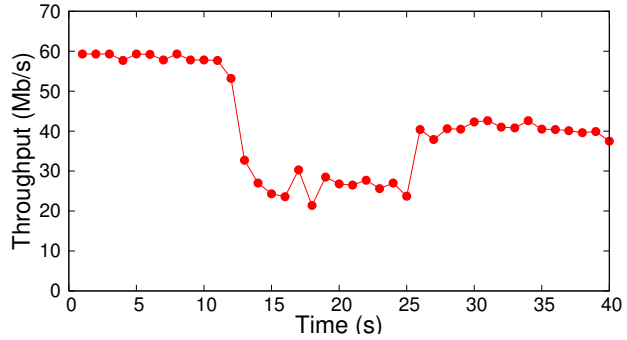
For comparison, we fix the A-MPDU length to 10 ms as a baseline 802.11n. All results are averaged over 5 runs, where each run lasts for 30 s.

Fig. 5 shows the throughput according to the degree of the mobility. When the station does not move (*Avg. 0 m/s*), there is almost no difference in throughput for all scenarios, because the maximum A-MPDU length can be successfully used. However, when the station is moving (*Avg. 1 m/s*), the baseline 802.11n provides the worst performance due to caudal loss. The original *STRALE* shows the highest throughput of 42.74 Mb/s in the mobile environment, while the WiSHFUL enabled variant provides slightly lower throughput performance than the original implementation. However, the throughput difference is only 3.1%, which allows us to conclude that the performance is not seriously affected by the delay explained in Section V-A. Consequently, *STRALE* operates well even in WiSHFUL Framework integrated variant implementation.

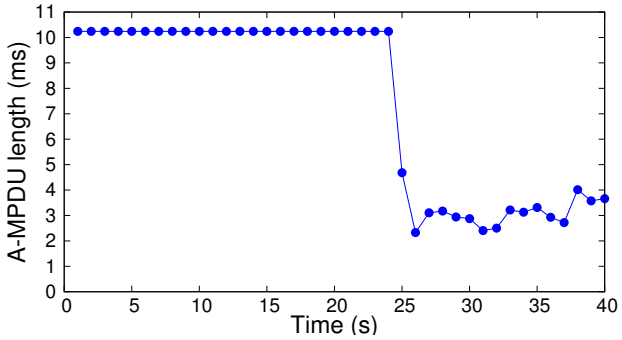
C. Demonstration

Now, we demonstrate our implementation with a timelined scenario. We firstly move the station without activating *STRALE*, observing the performance degradation. Then we activate *STRALE* to cope with the caudal loss problem. In order to show the operation of *STRALE* on WiSHFUL Framework in real-time, we develop a Python-based program, which displays real-time application-level throughput from *Iperf* [20] as well as the A-MPDU length averaged over the last one second. Fig. 6 portrays a scene from a demo presentation.

The throughput and A-MPDU length results of the demonstration are presented in Fig. 7(a) and Fig. 7(b), respectively, which are redrawn from the graphs shown in the Python-based program for better visibility. At first, the station does not move, and the PHY rate and the A-MPDU length are fixed to 65 Mb/s and 10 ms, respectively. The station begins to move at an average speed of 1 m/s at 11 s. Since *STRALE* is not activated yet, the caudal loss problem becomes intensified, severely degrading the achieved throughput under 22 Mb/s. On the other hand, when *STRALE* is activated at 25 s, we observe



(a) The throughput curve of the demonstration. The stations starts moving at 11 s at an average speed of 1 m/s, and *STRALE* is activated at 25 s through UPI.



(b) A-MPDU length variation during the demonstration. After activating *STRALE*, A-MPDU length is adaptively reduced in order to cope with caudal loss problem.

Fig. 7. Demonstration result of *STRALE* on WiSHFUL Framework. Note that the figures are redrawn using Gnuplot.

that the throughput becomes about 40 Mb/s, by adaptively reducing A-MPDU length as shown in Fig. 7(b).

VI. Conclusion

WiSHFUL proposes an architecture for flexible and unified control of wireless systems and platforms. The WiSHFUL architecture is implemented in WiSHFUL Framework, which offers UPIs for experiments on various platforms. In this paper, in order to illustrate the convenience and benefit of the WiSHFUL architecture, we have implemented *STRALE* on WiSHFUL Framework. We add UPI functions for *STRALE*, and extend the device module for *ath9k* with the new UPI functions by redesigning the structure of *STRALE*. We evaluate the performance of the newly ported to WiSHFUL *STRALE* by comparing with the original implementation. We also demonstrate *STRALE* on WiSHFUL Framework by employing our own Python-based visualization tool.

STRALE can be extended to all WiSHFUL supported devices by developing a new device module for the specific device. Furthermore, any proposed control algorithm can be added to WiSHFUL Framework by defining required UPIs and developing/updating the corresponding device module, following the paradigm of our approach to integrate *STRALE* into WiSHFUL. Overall, the porting of *STRALE* into WiSHFUL can be considered an example application of the framework to enable control algorithm implementation over UPIs.

Acknowledgements

This research was supported by the International Research & Development Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning of Korea (NRF-2014K1A3A7A03073470) and the European Commission Horizon 2020 Programme under grant agreement n645274 (WiSHFUL) and the FWO SBO SAMURAI: Software Architecture and Modules for Unified RAdIo control project.

References

[1] NI USRP RIO software defined radio, http://www.ni.com/en-us/shop/select/software-defined-radio-reconfigurable-device/.
[2] WARP: Wireless open-access research platform, https://warpproject.org/trac/.
[3] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. M. Voelker, "Sora: High Performance Software Radio Using General Purpose Multi-core Processors," in *Proc. USENIX NSDI*, 2009.
[4] P. Ruckebusch, S. Giannoulis, P. G. Domenico Garlisi, P. Gawowicz, A. Zubow, M. Chwalisz, E. D. Poorter, I. Moerman, I. Tinnirello, and L. DaSilva, "WiSHFUL: Enabling Coordination Solutions for Managing Heterogeneous Wireless Networks," *IEEE Communications Magazine*, vol. 55, no. 9, 2017.
[5] WiSHFUL Github source code repository, https://github.com/wishful-project/.
[6] WiSHFUL UPIs documentation, https://wishful-project.github.io/wishful_upis/.
[7] IRIS: The reconfigurable radio testbed, https://iris-testbed.connectcentre.ie/.
[8] GNURadio, https://www.gnuradio.org/.
[9] WMP: Wireless MAC processor, http://wmp.tti.unipa.it/.
[10] B. Jooris, J. Bauwens, P. Ruckebusch, P. D. Valck, C. V. Praet, I. Moerman, and E. D. Poorter, "TAISC: A Cross-platform MAC Protocol Compiler and Execution Engine," *Computer Networks*, vol. 107, pp. 315–326, 2016.
[11] S. Byeon, K. Yoon, C. Yang, and S. Choi, "STRALE: Mobility-Aware PHY Rate and Frame Aggregation Length Adaptation in WLANs," in *Proc. IEEE INFOCOM*, May 2017.
[12] Ath9k: Atheros Wireless Driver, http://wireless.kernel.org/en/users/Drivers/ath9k/.
[13] M. Vipin and S. Srikanth, "Analysis of Open Source Drivers for IEEE 802.11 WLANs," in *Proc. ICWCSC 2010*, Jan. 2010.
[14] IEEE 802.11, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*, IEEE Std., Mar. 2012.
[15] IEEE 802.11ac, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Enhancements for Very High Throughput for Operation in Bands below 6 GHz*, IEEE Std., Dec. 2013.
[16] S. Byeon, K. Yoon, O. Lee, W. Cho, S. Oh, and S. Choi, "MoFA: Mobility-Aware Frame Aggregation in Wi-Fi," in *Proc. ACM CoNEXT*, Dec. 2014.
[17] O. Lee, W. Sun, J. Kim, H. Lee, B. Ryu, J. Lee, and S. Choi, "ChASER: Channel-Aware Symbol Error Reduction for High-Performance WiFi Systems in Dynamic Channel Environments," in *Proc. IEEE INFOCOM*, May 2015.
[18] *STRALE* code, https://github.com/pathetique/STRALE.git/.
[19] HostAP: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator, http://hostap.epitest.fi/hostapd/.
[20] Iperf: TCP/UDP Bandwidth Measurement Tool. https://iperf.fr/.