



weSDN: **SDN Extends to Wireless End Devices**

Mostafa Uddin, Old Dominion University

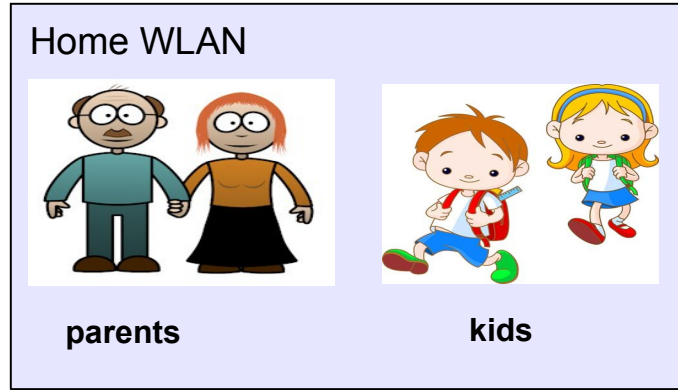
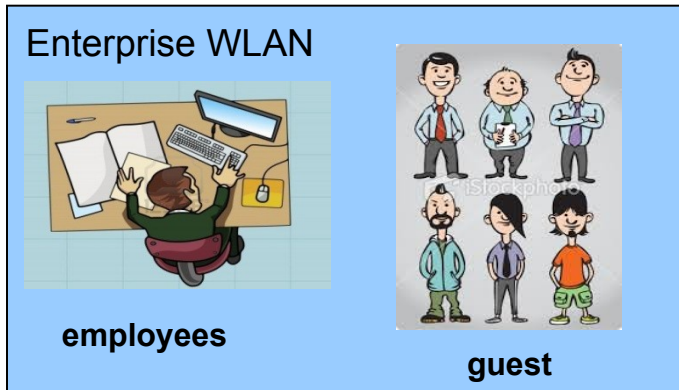
Mentor: Jeongkeun “JK” Lee



WLAN Virtualization

Wireless LANs are becoming ubiquitous

WLAN virtualization enable effective sharing of wireless resources by a diverse set of users with diverse requirement



WLAN Challenges

Interference

Radio Resource Management (RRM) can monitor and react to interference

Lack of control over Client-to-AP uplink traffic

(SDN enable) APs can only control down-link traffic

Note: Ethernet is p2p link, SDN-enabled Ethernet edge can do virtualization

Wireless medium is shared

Uncontrolled uplink TX can impact other client performance



Our Solution: Extend SDN to Wireless End Device

Control the uplink TX from the client side using SDN framework

Manage uplink 802.11 QoS settings

e.g. One client greedily using highest priority can unfairly dominate uplink air-time resource.

Can Enable end-to-end QoS provisioning.



weSDN approaches

Use **Open vSwitch** on end device to monitor and manage application traffic

Use **pTDMA**, a TDMA like scheduling, to virtualize airtime resources between network slices

Maintain **pTDMA** scheduling using **Linux Qdisc**

Provide trusted information about user's **application traffic**



Design Question 1

Will end user allow the network to control their device traffic?

It is not a new concept to centrally control the client devices (e.g. PC COE, BYOD solutions, VPN client, mobile WAN acceleration).

It allows following benefits:

- Support enhance network security, end-to-end QoS and WLAN virtualization.
- Users can have better and predictable network performance.

Operators can take drastic measures (drop frames, TCP ACKs) on non-participating clients



Design Question 2

Where should the SDN APIs be integrated in the client software stack?

We believe WiFi protocol and the client WiFi stack are better to be kept intact, Why?

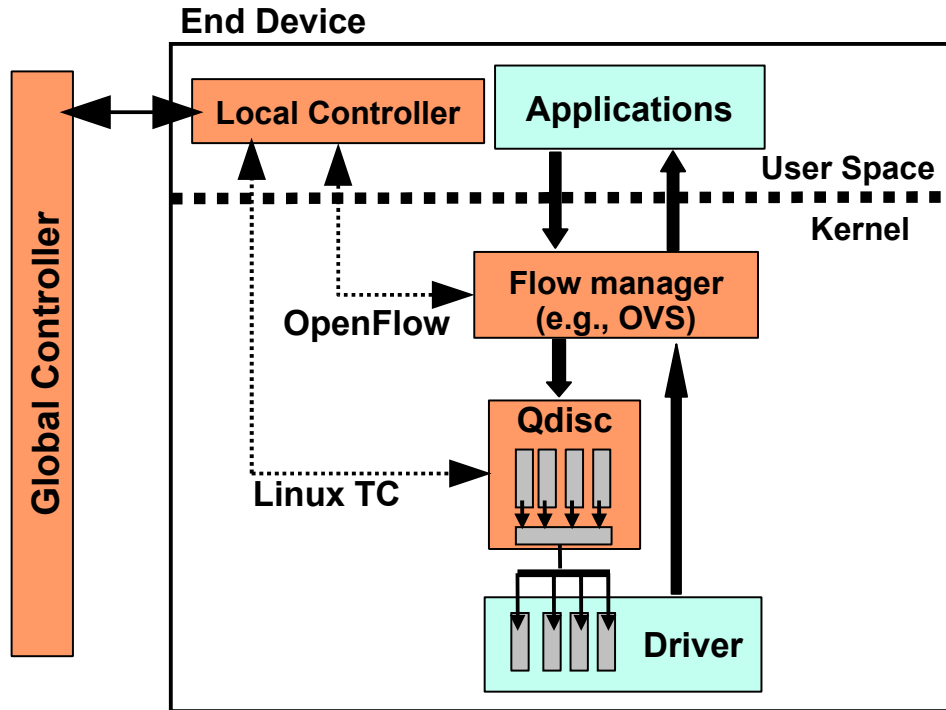
- Hard to deploy diverse vendors, chipsets, drivers.

Instead of driver hacking we want to leverage and extend existing OS/SDN APIs

- Implement the traffic airtime scheduler in Linux Traffic Control (TC) qdisc.
- Integrate the Open-vSwitch (OVS) above the qdisc in Android.

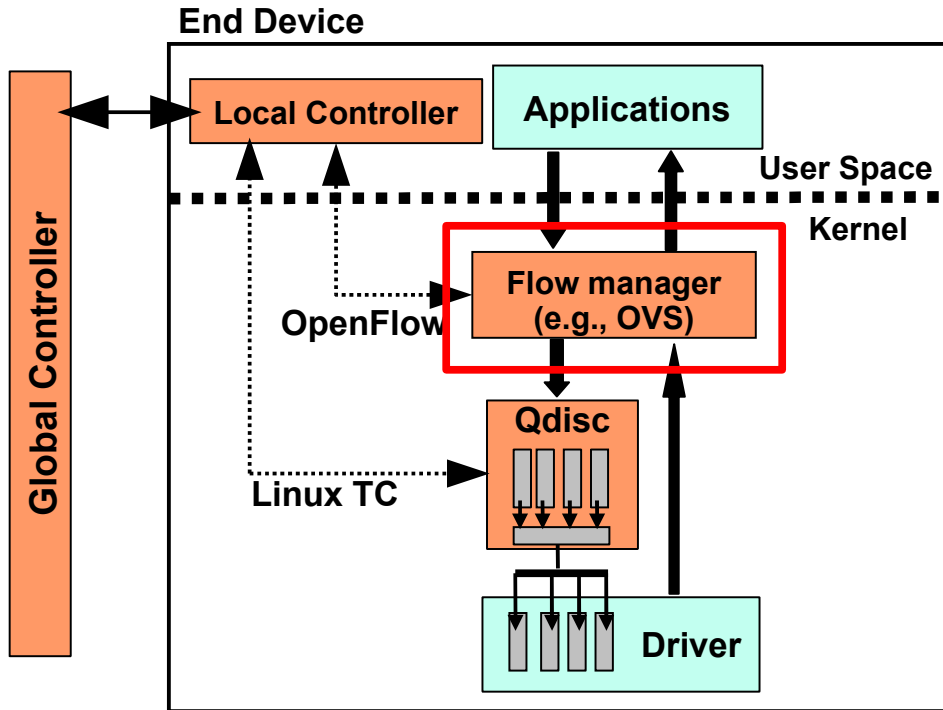


weSDN Architecture



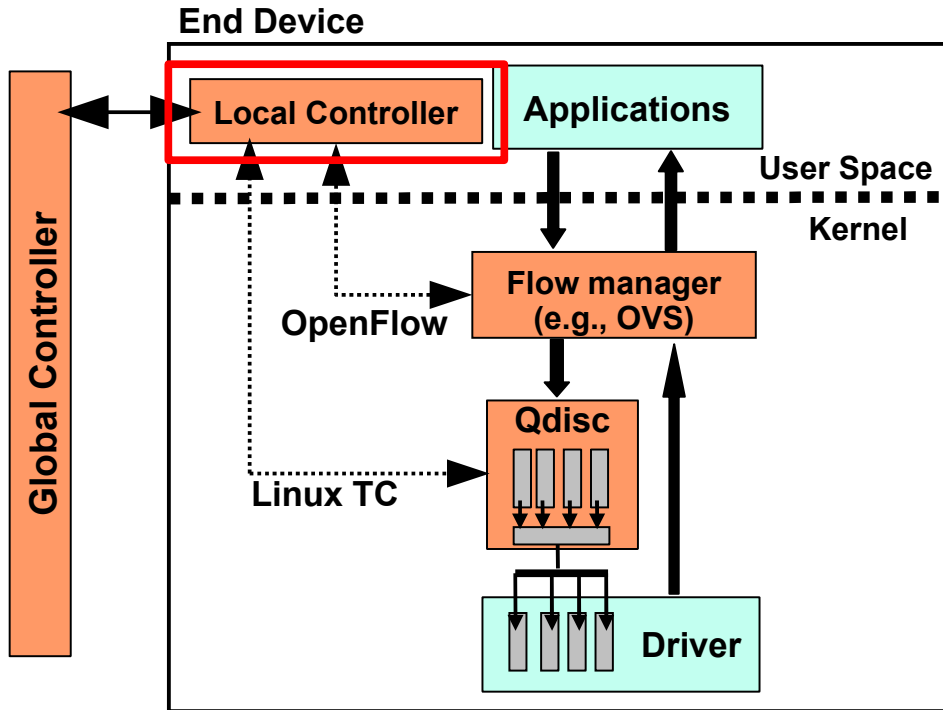
1. pTDMA Scheduler (Linux Qdisc)
2. Flow manager (e.g. Open vSwitch, OVS)
3. Local Controller
4. Global Controller

Flow Manager



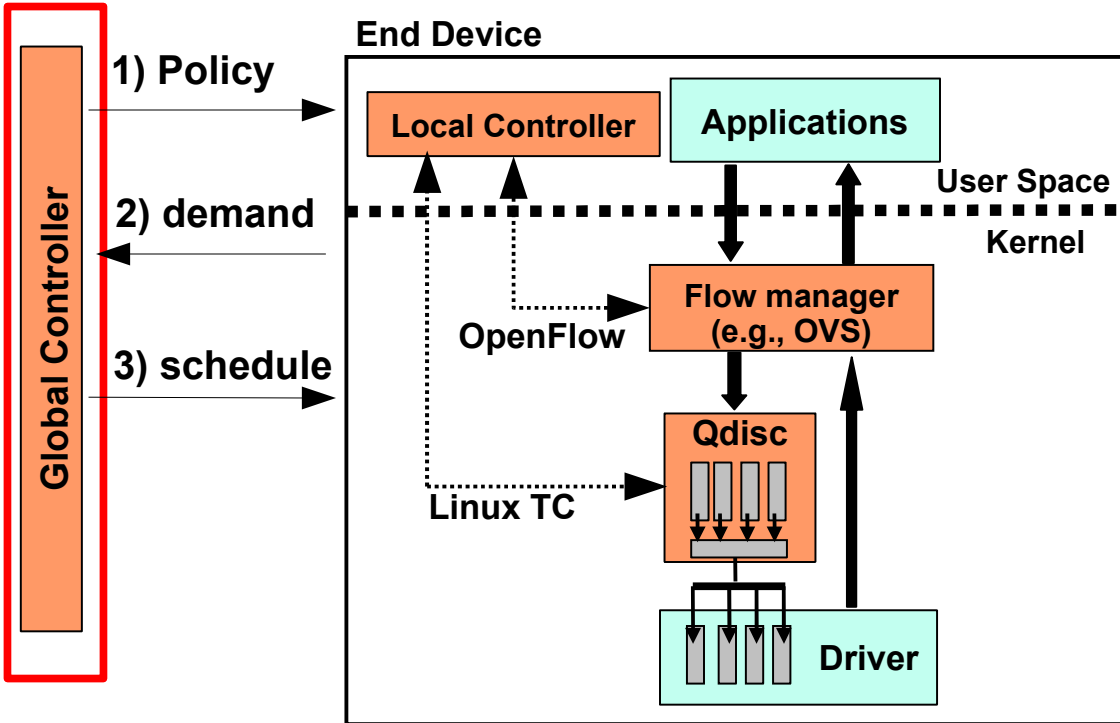
1. It is a software OpenFlow switch (e.g. OVS)
2. Collect Flow statistics:
 - OF Stat extension: burst duration, burst rate and inter-burst time.
3. Ensure correct QoS marking
 - e.g IP DSCP/TOS or VLAN PCP

Local Controller



1. Identify flows correspond to each application.
2. Generate flow rules for OVS
 - Based on per-application policy given by central controller or the user.

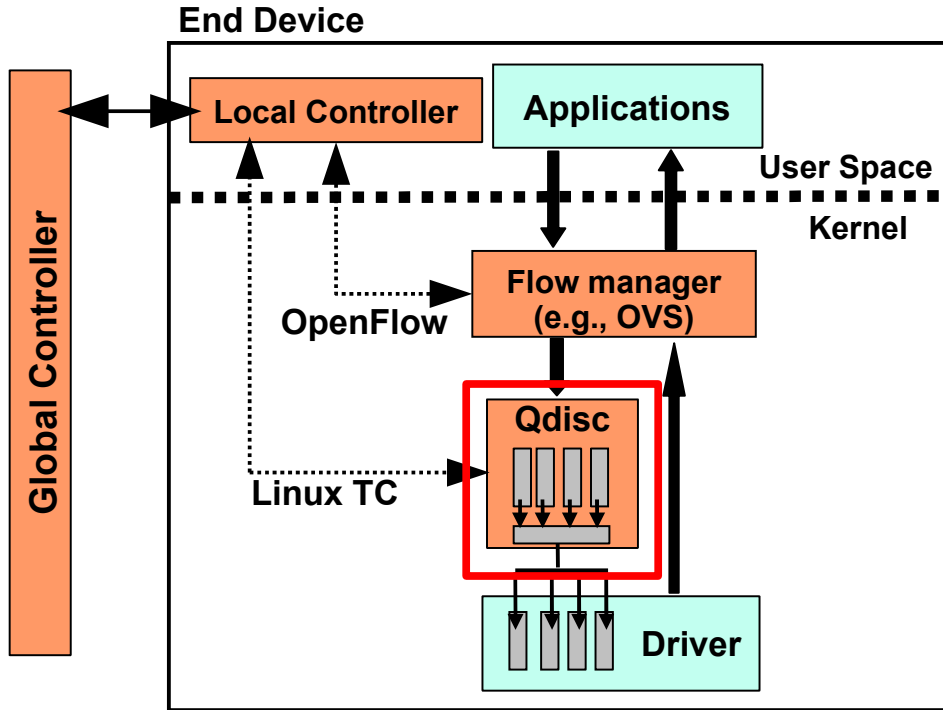
Global Controller



Interacts with local controller in 3 steps

- 1) Provides per-slice/users/app policy to local controller.
- 2) local controller send aggregate demand to global controller.
- 3) Compute schedules and send to end device.

pTDMA Qdisc



1. Receive *time window* from the local controller to start/stop dequeuing.

- Time Window: e.g. [Start time, active duration, sleep duration]

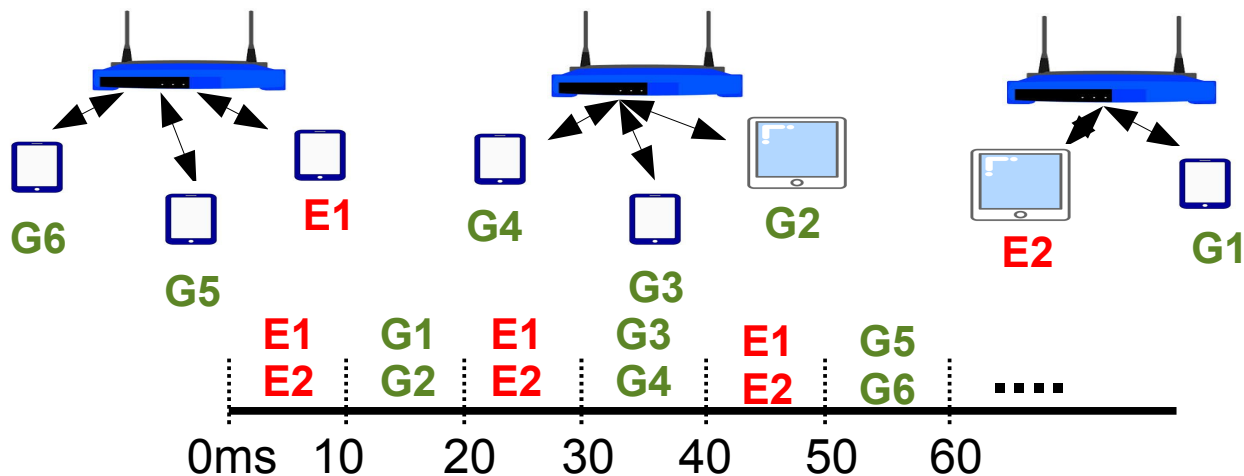
- e.g. 05:30:30, 10ms, 30ms

2. pTDMA qdisc is an extension to linux multiq that supports 802.11e QoS.

pTDMA

Manage airtime share between network instances (their clients) that collocate in space and channel

Assigning separate airtime slices among different network instances



***p*TDMA: Scheduling Principle**

Allocate large enough time window to transmit and receive multiple packets

Schedule multiple clients in a common slot to maximize channel utilization

The interval between consecutive time windows should be based on applications' traffic pattern & demand



Technical Challenges

Milli-second level synchronization between the phones is needed for effective *p* TDMA

Achievable by GPS

Note) traditional per-packet TDMA requires micro-second level time sync

Driver buffering delay is large

Bufferbloat: Large ring packet buffer (100 to 300, total bytes >150KB) used by WiFi, Ethernet drivers

Byte Queue Limit(BQL) for Ethernet driver in Linux: buffer size limit is dynamically set based on recent “byte” dequeued by the NIC

We set hard byte limit in Wi-Fi Driver to 15KB, enough for 10 pkt 802.11 aggregation



Prototype

Prototyped *weSDN* client-side component on eight Google Nexus 4 Android phones

Root the device to install OVS and *p*TDMA qdisc kernel modules.

Re-Build the kernel image

To implement the Wi-Fi driver byte limit in Nexus 4 WiFi driver

Note) some other phones have Wi-Fi driver as kernel module (e.g. Nexus S)



Experiment


We formed two network slices

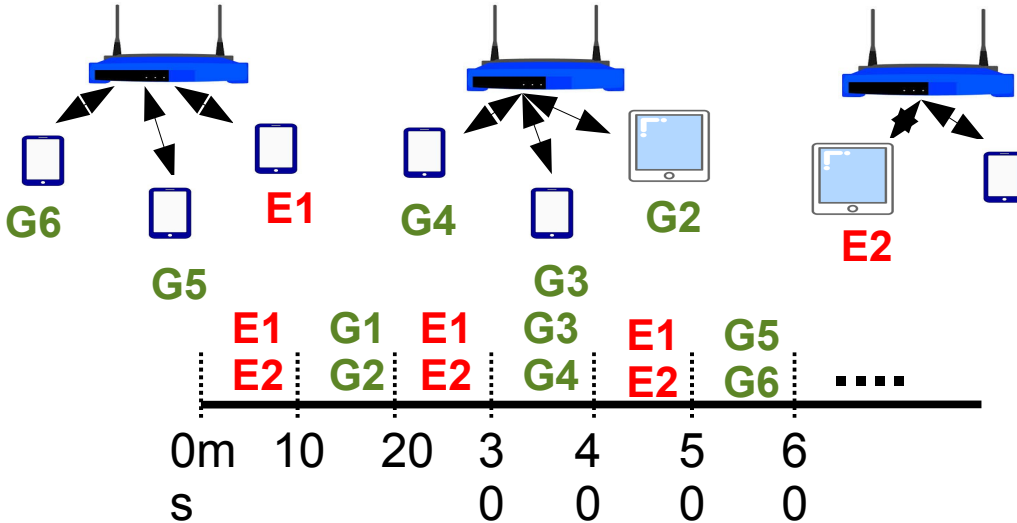
“employee” network with 2 devices

“guest” network with 6 devices

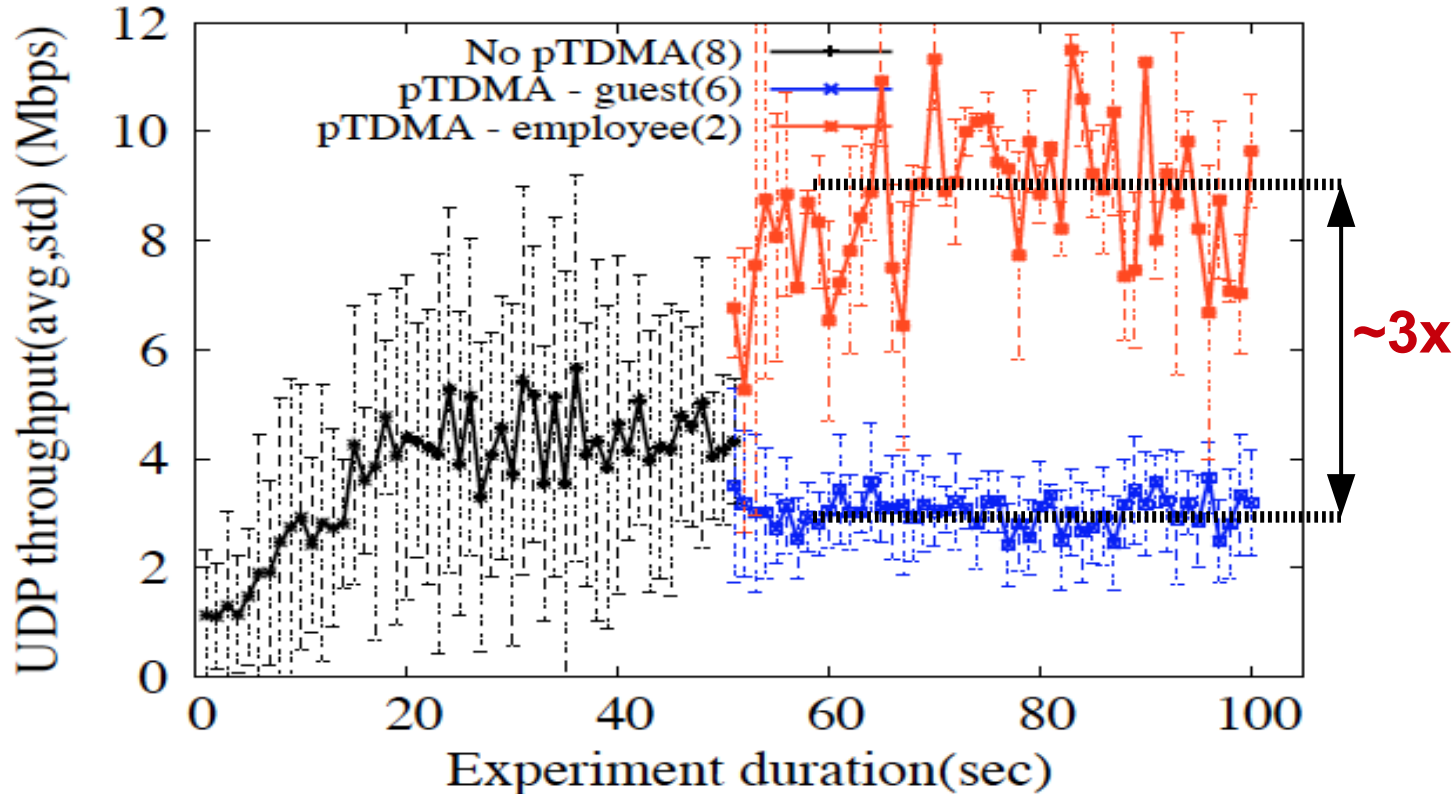
Applied following p TDMA schedule with 50:50 airtime share between two slices

3:1 airtime ratio btw an employee and a guest.

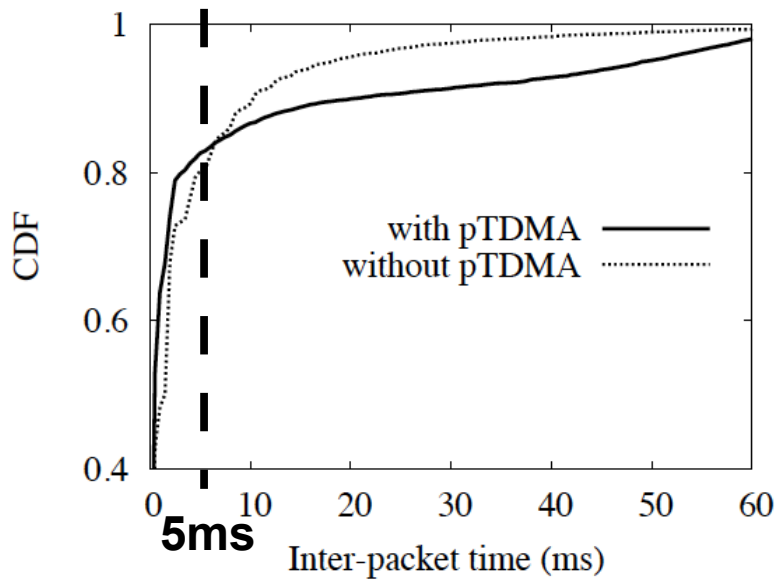
(but all devices are connected to one )



Evaluation: Uplink **UDP**



Evaluation: Sleeping Time

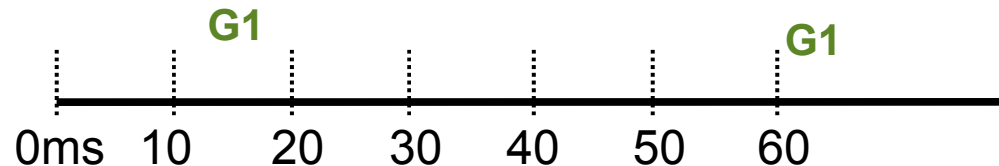


Assume the driver goes to sleep state after 5ms of inactivity in WMM-PS

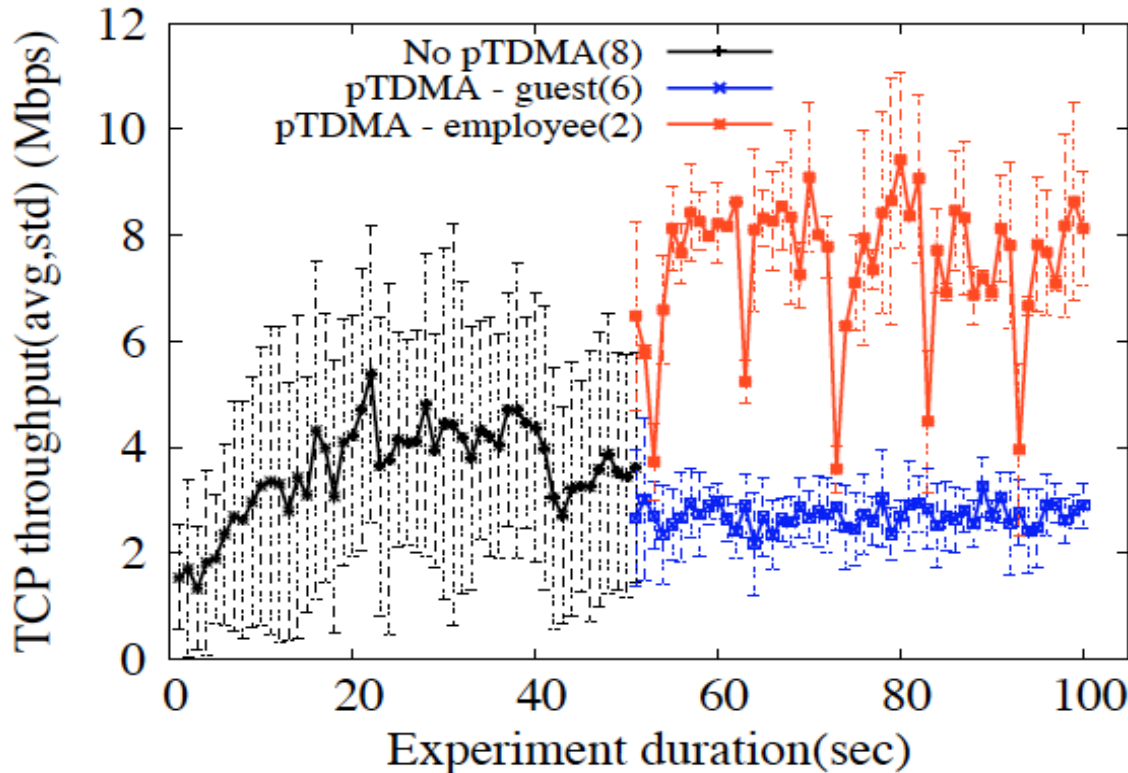
In *non-pTDMA*, client sleeps 28% of the time.

In *pTDMA*, client sleeps 80% of the time

Client device *pTDMA* schedule



Evaluation: Uplink TCP



1. Increased transmission time in pTDMA schedule do not adversely effect TCP performance.

Conclusion

Summary

Integrate SDN API to end clients for WLAN virtualization

Demonstrate WLAN virtualization by p TDMA

Future Work

OpenFlow Statistics extension for burst measurement

p TDMA scheduler

Controller Implementation

In-band control channel



Acknowledgement

- **Jean Tourrilhes**
- **Souvik Sen**
- **Kyu-Han Kim**
- **Sujata Banerjee**
- **Manfred Arndt, ATG**
- **Zafar Qazi**

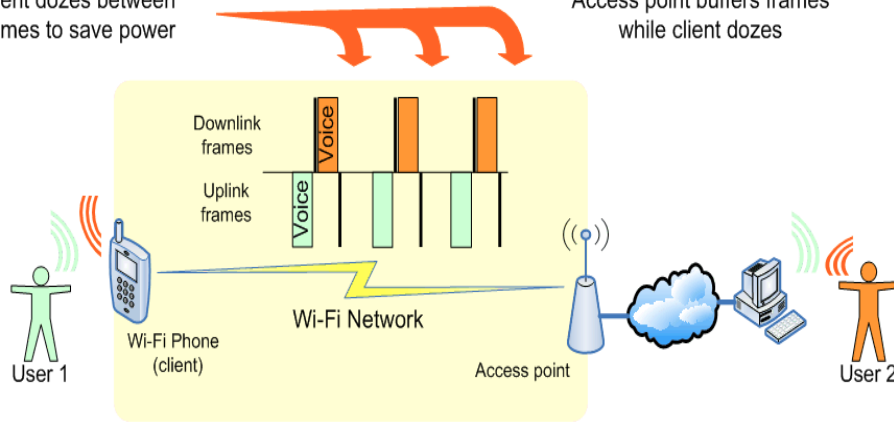


Backup Slide: Power Saving

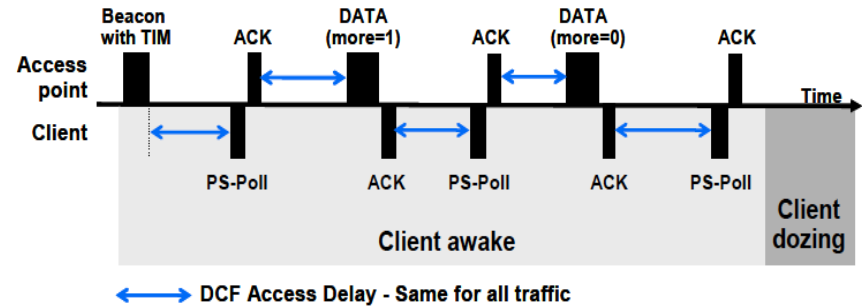
WMM Power Save in a Wi-Fi Network

Client dozes between frames to save power

Access point buffers frames while client dozes



Wi-Fi legacy power save



1. *weSDN* leverage WMM-PS to indirectly confine the downlink traffic to the time window.
2. *pTDMA* allows to efficiently utilize the WMM-PS to have more sleeping time without sacrificing the throughput performance.