Enabling TDMA for Today's Wireless LANs

Zhice Yang[1†], Jiansong Zhang[2†], Kun Tan[2], Qian Zhang[1], Yongguang Zhang[2]
zyangab@connect.ust.hk, jiazhang@microsoft, kuntan@microsoft, qianzh@cse.ust.hk, ygz@microsoft
[1]CSE, Hong Kong University of Science and Technology, [2]Microsoft Research Asia
[†]Co-primary Authors

***Abstract*—Today's WLANs are struggling to provide desirable features like high efficiency, fairness and QoS because of the use of Distributed Coordination Function (DCF). In this paper we present OpenTDMF, an architecture to enable TDMA on commodity WLAN devices. Our hope is to provide the desirable features without entirely rebuilding the WLAN infrastructure. OpenTDMF is inspired by and architecturally similar to Software Defined Networking (SDN). Specifically, we leverage the backhaul of WLAN to coordinate all the stations for channel access. This fine-grained coordination is performed in a decoupled control plane which includes a central controller and programmable APs. To realize OpenTDMF on commodity WLAN devices, we develop several novel techniques to achieve μs-level time synchronization among all the APs. We also enable AP-triggered uplink transmission so that all the transmissions in the WLAN can be determined. We implemented a prototype of OpenTDMF based on commodity WLAN devices. Empirical results validate the OpenTDMF design and demonstrate its benefits.**

I. Introduction

Wireless LANs are based on *distributed coordination function* (DCF) to share the wireless medium. Benefit from its distributed and asynchronous nature, DCF makes WLANs low-cost to implement and deploy, which is an important reason for its wide adoption today as the indispensable infrastructure for Internet access. However, on the other hand, it has been well understood that DCF is inefficient in resolving congestion and weak in handling interference, *e.g.* the hidden/exposed terminal problems. As a consequence, today's WLANs still struggle to provide efficient and fair channel access and QoS, which are especially desirable in enterprise environments to support voice/video conferencing, video broadcasting, *etc.*

It has been proved in cellular networks that TDMA, which schedules channel access in a synchronized and centralized manner, is an effective means to support these desirable features. Thus, our goal in this paper is to build TDMA on top of today's WLANs. The hope is to provide these features without rebuilding the entire WLAN infrastructure. In literatures, some efforts have been made for similar purposes. For example, CENTAUR [1] and TRACK [2] schedule downlink traffic to mitigate hidden/exposed terminals. Different from these work, we target to provide an architecture to schedule all the traffic including downlink and uplink, and to support all the desirable features including efficiency, fairness and QoS.

We present OpenTDMF[1], an arcitecture that enables TDMA for today's enterprise WLANs. OpenTDMF borrows from the *Software Defined Networking* (SDN) design, in that it separates

[1]TDMF stands for Time Division Multiple Flows

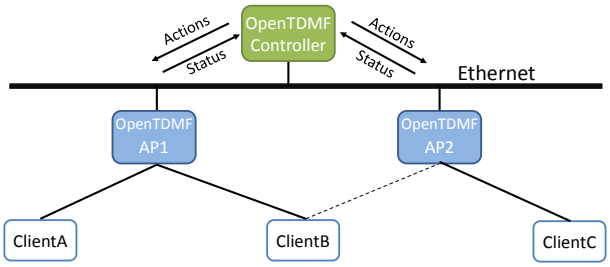


Fig. 1. **OpenTDMF Architecture**. OpenTDMF enables TDMA with commodity devices in today's enterprise WLANs. The central controller coordinates flow-level channel access for the whole WLAN. Every AP coordinates packet-level channel access for its locally associated stations.

control plane from data plane and exposes the channel access functions to higher layers. As illustrated in Fig. 1, data plane is controlled by the *OpenTDMF interface* which resides in APs. Analogous to the OpenFlow [3] interface, OpenTDMF interface operates over a table of {FlowID, Actions} tuples, where an action specifies how a flow access wireless channel. Flow-level actions prevent the backhaul from overwhelmed by packet-level information. OpenTDMF interface also allows every AP to focus on coordinating its locally associated devices without worrying about the interference across APs. The control plane in our design is managed by OpenTDMF Controller, which takes input per-flow QoS requirements (*e.g.* bandwidth and delay) specified by applications and channel status (*e.g.* link quality, collision graph, *etc.*) reported by APs. According to these QoS requirements and channel information, OpenTDMF controller calculates actions for every flow and sends these actions to every AP periodically.

To realize OpenTDMF on commodity devices, we need to address two main technical challenges. First, OpenTDMF performs centralized coordination for channel access which requires time synchronization among APs. In OpenTDMF, we perform time synchronization on backhaul network. Specifically, we use *Precise Time Protocol* (PTP) and develop several novel techniques to handle the delay variance on Ethernet and in AP's firmware (non-realtime linux), which can easily reach hundreds of microseconds otherwise [1]. As a result, we provides an accuracy in microsecond level, which is unperceivable because it is much smaller than transmission time of a WLAN packet ($100 \sim 1000\mu s$). Comparing to solutions performed in air interface, our solution does not hurt spectrum efficiency and has better scalability [4]. Second, OpenTDMF APs should determine channel access for all the traffic of their locally associated stations. While determining downlink traffic

Ethernet			IP				TCP/UDP		Channel Access	
Src	Dst	Type	Src	Dst	Proto	ToS	Src Port	Dst Port	Time Slots	Priority

Fig. 2. **OpenTDMF Flow Table** . OpenTDMF defines flows based on packet headers, similar to OpenFlow.

is straightforward, determining uplink traffic is fundamentally challenging because today's WLANs assume every device determines channel access by itself. Our solution is inspired by an interesting observation on commodity Wi-Fi chip, based on which we realize AP-triggered uplink transmissions on commodity Wi-Fi devices.

We implemented OpenTDMF in TP-Link 4900 APs by modifying the firmware. Evaluation results validated OpenTDMF's ability in enabling TDMA. We also demonstrate OpenTDMF's benefit on a small-scale network, which shows significant performance gain over DCF.

Our contributions in this paper are summarized as follows:

- We provide a thoughtful study towards providing high accuracy time synchronization among commodity APs over backhaul networks. Our solution is able to synchronize the transmission of APs within $10\mu s$ even under near-saturated data traffic in the backhaul.
- To the best of our knowledge, we are the first to enable AP-triggered uplink transmission on commodity Wi-Fi devices. Moreover, we show that network efficiency can be improved by up to 30% benefit from contention-free channel access.
- We implemented and evaluated the OpenTDMF design on commodity WLAN devices and demonstrates the feasibility of enabling TDMA. Tests on small-scale network show 210% gain with exposed links, significant improvement on fairness in hidden links and the ability to provide QoS.

II. OpenTDMF's Architecture

To enable TDMA, we need to schedule and enforce channel access in a synchronized and centralized manner. OpenTDMF provides a means by leveraging backhaul networks to coordinate channel access for all the devices in WLANs, thereby enabling TDMA.

Architecturally, OpenTDMF borrows from the SDN design in that it separates control plane from data plane, and exposes channel access functions to higher layers which are deeply hidden in lower layer of network stack traditionally. The data plane is controlled by OpenTDMF interface which is managed in software by OpenTDMF controller. In this section, we first elaborate OpenTDMF's components. We then use an example to illustrate how TDMA is enabled with OpenTDMF. Finally, we discuss the technical challenges to realize TDMA on commodity WLAN devices.

A. OpenTDMF Components

OpenTDMF contains a centralized controller and a set of APs which are connected using Ethernet. The controller and all APs jointly perform the coordination function for the whole WLAN. Specifically, OpenTDMF controller coordinates all APs in coarse-grain, and each AP coordinates its locally associated stations in fine-grain. The communication channel between OpenTDMF controller and APs is the OpenTDMF interface.

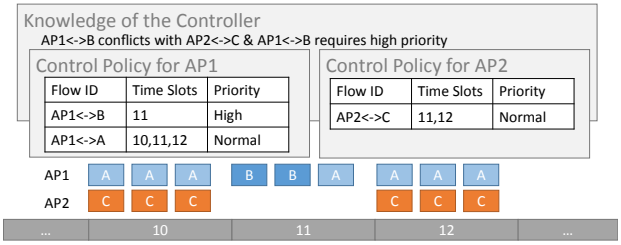


Fig. 3. **Example of TDMA with OpenTDMF**. Corresponding topology is shown in Fig. 1. OpenTDMF enables TDMA and brings the benefits that (i) random back-offs are eliminated, (ii) collisions are avoided, (iii) transmission order can be flexibly set.

OpenTDMF interface operates over a table indexed by flows, which means OpenTDMF controller coordinates channel access in flow-level. Flows are identified using packet headers of various layers as in Fig.2. The actions specified for each flow contains two elements: *slots* and *priority*.

OpenTDMF divides time domain into *slots*. Formally, a slot with index K is defined as: $[T_K^s, T_K^e) := [T_0 + K \times S, T_0 + (K+1) \times S)$, where T_K^s and T_K^e are start time and end time of slot K. T_0 is an arbitrary reference time. S is the slot size. The slot size should be large enough to carry multiple packets, and small enough to be unperceivable to upper layer protocols and applications, therefore typical slot size is $2ms \sim 10ms$. For the sake of simplicity, slot size is identical in our design. Moreover, to ensure a consistent view on all APs, slot should be synchronized in terms of both boundary time and index.

Through the OpenTDMF interface, OpenTDMF controller assigns a set of slots for each flow. In its set of slots, packets of this flow can be transmitted into the air. Typically, OpenTDMF controller assign non-overlapping slots to interfering flows. OpenTDMF controller also assign earlier slots or more slots to provide low latency or high throughput for flows having QoS requirements. Therefore, by scheduling slots through OpenTDMF interface, spectrum efficiency and fairness could be optimized, QoS could be guaranteed.

OpenTDMF controller also assigns priority to each flow for the purpose of supporting QoS in finer-grain. According to the value of priority, every AP determines packet-level transmission orders of its local flows. Basically, flows have higher priority value will be transmitted earlier. Flows have the same priority value will be transmitted in a round-robin fashion to ensure fairness. OpenTDMF controller determines priority of flows either by referencing the IP layer *Type of Service* (ToS) element of a flow, or the QoS requirement explicitly specified by the corresponding application.

The slots and priority values are computed by OpenTDMF controller using a scheduling algorithm, such as weighted fair queuing [5]. The scheduling algorithm takes input collision graph of the whole WLAN. The collision graph can be computed using existing techniques, such as micro-probing [1].

B. TDMA with OpenTDMF

To illustrate the benefits brought by OpenTDMF, we use an example to demonstrate the results of TDMA based channel access enabled by OpenTDMF. The example uses the topology in Fig.1 in which client A and client B are associated to AP1, client C is associated to AP2. AP2 interferes with client B therefore flows on AP2 and flows on client B should not transmit simultaneously. The example includes three flows on three clients respectively, in which the flow on client B has high priority whereas the other two flows have normal priority. The detailed transmission order in 3 slots $\{10,11,12\}$ as well as the behavior of OpenTDMF are shown in Fig. 3. We can see the controller assigns all slots to the flow on A because it does not interfere with any other flow. The flow on B and the flow on C are assigned to non-overlapping slots so that their transmission will not collide. As a result, in slot 10 and slot 12, flow on A and flow on C transmit concurrently. In slot 11, flow on C is stopped to prevent from interfering with the flow on B. Moreover, in slot 11, packets of the flow on B are transmitted first because this flow has higher priority. Finally, since the order of packets inside a slot is solely determined by the AP, random back-offs are no longer needed. To conclude this example, we can see TDMA based channel access enabled by OpenTDMF brings a lot of benefits including higher efficiency and guaranteed fairness and QoS.

C. Challenges on Commodity Devices

To implement OpenTDMF on today's WLAN devices, we need to address two main technical challenges. First, we need to synchronize AP's time on us-level. One option for time synchronization is using broadcast in air interface. 802.11 standards define *Timing Synchronization Function* (TSF) to synchronize clocks on multiple Wi-Fi devices through beacons [6]. The problem of TSF and other air interface solutions [4], [7], [8] is that the synchronization error accumulates with increased number of hops. Moreover, it requires full connectivity of APs which is not necessarily satisfied. In OpenTDMF, we choose to perform time synchronization on backhaul network to avoid above problems. However, it is still non-trivial because we need to handle the delay variance on Ethernet and in AP's firmware (non-realtime linux) which can easily reach hundreds of microseconds [1].

Second, OpenTDMF APs should be able to determine channel access for all their locally associated devices. Determining channel access for uplink traffic is challenging because commodity devices assume every device determines channel access by itself. We need a solution that is both feasible on commodity devices and unharmful to spectrum efficiency.

In next two sections, we elaborate our solutions for these challenges.

III. Time Synchronization and Accurate Execution

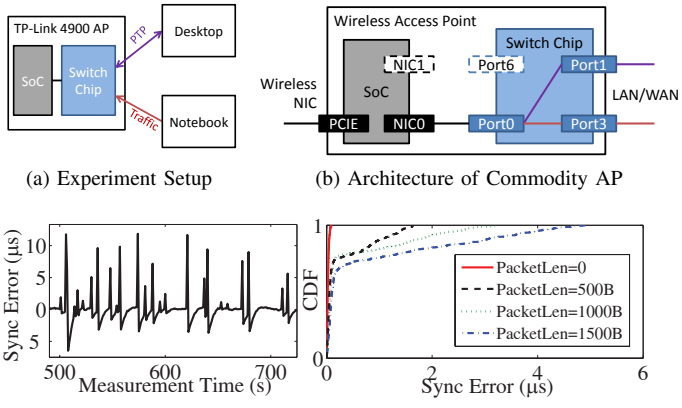
OpenTDMF requires APs to accurately execute the arranged transmissions, *i.e.* APs should synchronize their slot boundaries and exactly start their transmissions at the boundaries. In this section, we describe how we achieve us-level time synchronization on commodity APs and how we ensure accurate execution of transmission actions on non-realtime OS.

A. Time Synchronization on Commodity APs

In OpenTDMF, we take advantage of the backhaul network of WLANs and employ IEEE 1588 *Precise Time Protocol* (PTP) [9] to enable high accuracy synchronization. In this subsection, we first briefly describe the PTP protocol. Then we show the practical problem on applying PTP to commodity APs and our solution.

1) PTP Basics: Different from *Network Time Protocol* (NTP) [10] which solely relies on time server's broadcasting messages, PTP further designs handshaking protocols to estimate the propagation delay on network and compensates the delay, therefore it is possible to achieve high-accuracy time synchronization using PTP.

In real systems, the main challenge for PTP is delay estimation, whose achievable accuracy is normally bounded by the variance of delay measurements. The variance mainly comes from the queuing delay in the software network stack. Since OS handles incoming packets in the soft interrupt (sirq), packets are processed in first come first serve manner. The embedded CPU on commodity AP does not have the ability to handle all the packets in the line rate, so the PTP packet is queued in the hardware and its timestamping by the software stack is delayed by the bursty traffic. If the PTP software takes delayed timestamp, the timestamping process itself will introduce non-trivial variance that easily exceeds tens of μs. Therefore, the synchronization accuracy goes down when the PTP packet encounters the incoming traffic. Fortunately, we observe that in commodity devices, it is popular the *Network Interface Card* (NIC) hardware adds timestamp for every incoming packets before the software stack. These devices include the TP-Link 4900 AP we use for our prototyping as well as most network adapters for PC. Therefore in OpenTDMF, we take hardware timestamping as an assumption.



(a) Experiment Setup (b) Architecture of Commodity AP

(c) Synchronization Results With Hardware Timestamp

Fig. 4. **PTP Experiment**. This experiment demonstrates that even the PTP packets are set to higher priority. Non-trivial delay variance still exists.

2) None-preemptive Swithing Delay: Hardware timestamp can solve the queuing delay problem, however, we found that

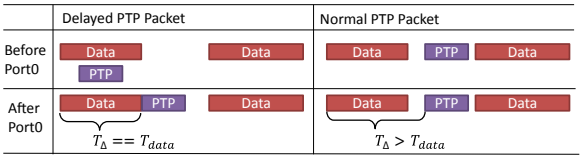


Fig. 5. **Delay Detection.** Figuring out those PTP packets that are delayed by a data packet and filtering them out.

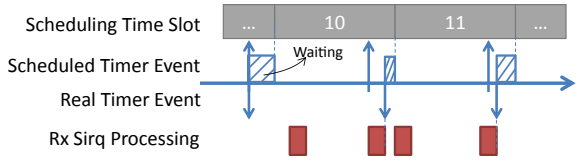


Fig. 6. **Timer+Waiting**. Timer events are scheduled earlier than the beginning of the scheduled time slot. The timer event handler busy wait for the scheduled time to avoid delay caused by other process.

it does not solve all the problem.

We use experiment results to demonstrate the problem. We set up the experiment as Fig. 4(a). We use a desktop equipped with Intel 217 Ethernet NIC as the master clock and use TP-Link 4900 wireless AP as the slave. The desktop is connect to one of the LAN ports of AP. Both the master and slave are running the open source *linuxptp* [11] software. One notebook is connected to another LAN port of the AP and generates UDP packets with different length to emulate normal data traffic. We summarize the results as below:

The left subfigure of Fig. 4(c) shows the synchronization error when the notebook generate traffic with 1500byte packets. A lot of spikes show that the AP is not consistently synchronized. Moreover, it is interesting that the right subfigure of Fig. 4(c) shows high correlation between synchronization error and packet length, which implies that the PTP packets are still delayed by data packets though higher priority has been set.

Finally, we figure out the problem is caused by the non-preemptive packet switching in AP's embedded switch and NIC. Specifically, as the hardware structure shown in Fig. 4(b), both incoming PTP packets and data packets are forwarded to the same port which connects to the NIC of AP's embedded processor. When the PTP packet arrivals at the switch, the forwarding will be delayed if some data packet is under transmission. This is because normally AP's switch does not allow the PTP packet to preempt the ongoing data transmission.

To mitigate this problem, we propose to filter out those PTP packets that are delayed by data packets. The heuristic metric is shown in Fig. 5. We compare the timestamp of a PTP packet and the data packet priori to it. We denote the time difference between these two packets as T_Δ. We also record the packet length of the previous data packet as T_{data}. If the PTP packet is delayed by the data packet, T_Δ will be equal to T_{data}, otherwise T_Δ will be greater than T_{data}. Using this metric, we modify the *Linuxptp* program to identify those delayed PTP packet and drop them. When there are larger volume of traffic, PTP packets will be dropped in higher probability. The PTP master then increases the measurement frequency to maintain consistent synchronization accuracy.

B. Accurate Execution

OpenTDMF also requires every AP to start transmission exactly at slot boundaries, *i.e.* accurate execution. Timer is an intuitive choice to generate event for the start of each time slot. The challenge comes from the fact that the current OS on the commodity APs such as `Openwrt` [12] is not a realtime OS, so the start of time of the event cannot be guaranteed. When the timer fires, it is likely that some other routine is in processing and cannot be preempted. Moreover, usually the embedded processor of commodity AP is weak, meaning that the the timer event could be delayed for a long time.

We solve this problem using a "Timer+Waiting" timer design. As shown in Fig. 6, for each scheduled time event, the AP sets its software timer a little bit earlier than the required time. After the software timer fires, the timer handler continuously polls the current time and blocks other process. Once the scheduled time is reached, the transmission starts. In this way, the error is bounded by the time spend in getting current time, which only takes several hundreds of nanoseconds in our testbed AP.

In order to estimate the waiting time, we reference the logged timestamps of repeated timer events under different traffic load. Due to the space limitation, we just describe the results here. The timer (hrtimer) in our test bed performs quite well in most of time but has the maximum value up to several μs in very rare cases. So we adopt $20\mu s$ in the implementation. When the slot size is $5ms$, the overhead for AP's processor is merely less than 1%.

IV. Determine Channel Access for Uplink Traffic

OpenTDMF also requires APs to determine channel access for all of its local associated clients. One possible approach is to synchronize all the clients through air interface, and assign different transmission time to different clients. The problem of this method is that AP needs to send fine-grained scheduling information to all clients using air interface, which may significantly hurt spectrum efficiency. Moreover, the synchronization error using air interface could be large on commodity WiFi devices. In details, the air interface method only synchronizes WiFi NIC's clock, since the timestamp of WiFi packet is generated by NIC's clock. However, the transmission execution is triggered using processor's clock, which is different from NIC's clock. These two independent clocks require additional effort to synchronize. Moreover, synchronization with multiple wireless hops adds additional synchronization error which can be more than $20\mu s$ [13].

In this section, we describe how we determine channel access for uplink traffic on commodity AP and client devices.

A. Polling

We use polling based method to realize tight control for uplink traffic, meaning all uplink transmissions are triggered by AP's packets. The concern for polling based method is the delay that a client reponds to a poll packet. If the response is generated in client's software, the delay caused

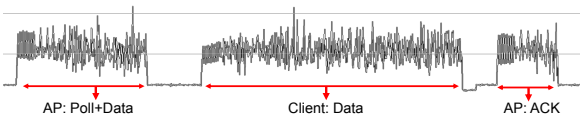


Fig. 7. **Polling Illustration**. This is the time-domain waveform of a polled transmission captured by software radio.

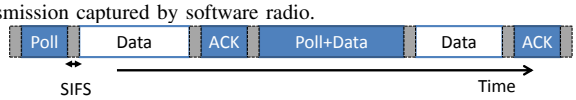


Fig. 8. **Uplink Transmission with Polling**. The timing diagram of one polled uplink transmission.

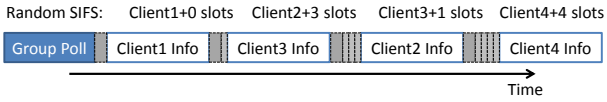


Fig. 9. **Group Polling**. Four clients contend for the transmission opportunity in the group polling period. Each client chooses its number of backoff slots randomly.

by network queuing could be as large as hundreds of us or more. This large response delay wastes much air time therefore may render polling based methods undesirable. Fortunately, we observe that commodity Wi-Fi devices usually support the feature of responding to poll packets in hardware chip, in which the response delay is merely $10us$ *Short Inter-Frame Space* (SIFS) required by 802.11 standards [6]. This hardware feature is designed to support the *Point Coordination Function* (PCF) mode which is an optional mode of 802.11 [6]. Although PCF itself is usually not implemented in WiFi devices' software, the feature of responding poll is normally implemented by chip vendors for the purpose of supporting possible future extension. Moreover, we can even avoid the overhead introduced by the poll packets because 802.11 allows to set a downlink data packet as poll [6]. In Fig. 7, we show the waveform of a polled transmission captured by a software radio. We can see the gap between a *poll+data* packet and the polled data packet is almost the same as the gap between the polled data packet and its ACK which must be SIFS.

In OpenTDMF, we extensively leverage this hardware feature for uplink control. Specifically, we set all the clients to a "gated" mode in which packets are gated to wait for transmission until a poll packet destined for the client is received. In this way, OpenTDMF APs can solely determine channel access for all its associated clients. We always try to transmit a downlink packet as poll to trigger a uplink transmission for the purpose of reducing the overhead introduced by poll packets. We also notice that even every uplink transmission is triggered by a short non-data poll packet, the spectrum efficiency is still improved because back-off is no longer needed. In Fig. 8, we show an example of OpenTDMF's uplink transmissions.

OpenTDMF APs need to learn the number of buffered packets in clients' queue so that they can determine whether to send poll packets. We piggyback this information on some uplink packets, for which we only need to pad several bytes to the MAC packet encapsulated in the WiFi driver. For the case that a client needs to send the first packet of a flow, we develop the *group poll* technique which is described subsection.

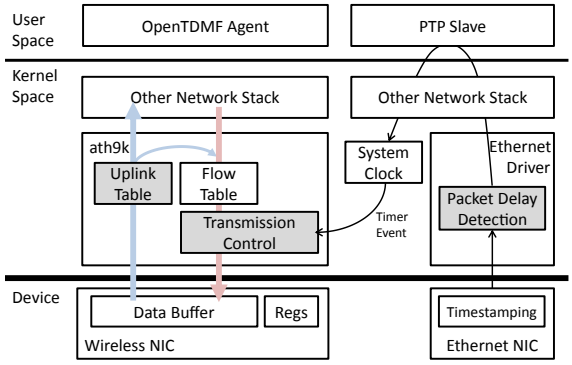


Fig. 10. **Software Architecture of OpenTDMF AP.**

B. Group Polling Periods

An important issue of the polling based method is the first access, *i.e.* how can a silent client starts a new uplink flow? or how can a newly coming client initiates the first packet? To solve this problem, we introduce *group polling periods* in which a group of clients can contend for transmitting their first uplink packets. Specifically, at the beginning of a group polling period, a poll packet carrying a *group address* is sent by AP. In order to reply to this group poll packet, new clients and silent clients who have a first packet to sent temporarily modify their chip's MAC address to the group address. To avoid collisions among multiple replying clients, we also mimic the random back-offs by temporarily modifying chip's SIFS (a configurable parameter) to a randomly selected value in a window, so that multiple clients can stagger their replies, as shown in Fig. 9.

To fit in OpenTDMF's slot based structure, we force the group polling period to be less than one slot. To ensure this, we decide that every client only transmits an ACK-size small packet to reply a group poll, which only carries necessary flow information. In our current design, we can allow up to 15 clients to reply in a single group polling period. If the number of silent clients is too large, we further divide them into multiple groups by allocating multiple group addresses. The mapping from a slient client to a group address is determined by AP and carried in beacon. After a group polling period, the silent clients change back to their own MAC addresses for the purpose of being able to reply ACK to its normal downlink packets.

V. Implementation

We implement OpenTDMF on commodity devices (sources are available at http://www.yangzhice.com). The AP for prototyping is TP-Link 4900 wireless access point. Its SoC is freescale P1014 with 800MHz PowerPC CPU and hardware-assisted-PTP NIC. The WiFi chips in the AP are Atheros AR9381 and AR9580 for 2.4G and 5G correspondingly, and they are compatible with `ath9k` driver. The AP runs `Openwrt` ver 12.09. As Fig. 10 shows, we patch `Openwrt`, `ath9k` driver and `linuxptp` to implement OpenTDMF interface. Some unpatched APs are set to station mode to act as clients, whose `ath9k` driver is enabled to support Polling. We use another two desktops with Debian 7.6 and Intel I217

Ethernet card. One is used as PTP master clock running PTP in master mode and the other running PTP in client mode is implemented as central controller. The controller communicates with APs with an user space agent through TCP. The agent controls the driver settings such as slot length and maintains the flow table through `debugfs`.

A. PTP Synchronization with Packet Delay Detection

To support accurate synchronization, we first enable PTP hardware timestamp function of the SoC by modifying and porting the PTP driver and device tree to `Openwrt`. The PTP enabled NIC is driven by 400MHz clock, in principle, the synchronization error can be as low as $\pm 5ns$. In the network stack of the kernel, we add entries for each packet to record its previous packet's packet length and receiving timestamp. The Ethernet driver is modified to be able to recognize the above information and identify the delayed packets and tag them. In the user space, `Linuxptp` is modified to be able to identify and drop the delayed PTP packets. We increase the synchronization frequency by up to $4\times$ to supplement the dropped ones when the network traffic is extremely busy.

B. Accurate Execution

To support the accurate execution, we implement the "timer+waiting" mechanism in section III-B. The software timer is implemented by `hrtime` in Linux. The waiting function is implemented by polling time through `get_time()`. To make the software transmission command feasible for the WiFi chip we implemented bellowing features in `ath9k`:

In order to arrange the packet for each slot, normal transmission packets are redirected to a new priority queue which provides slot based arrangement by estimating packet duration and transmission rate. We disable the hardware retransmission by setting the `tx_tries` flag on the packet to 1 to accurately estimate transmission time of each packet. The retransmission is implemented in software by adding multiple copy of the packet the transmission queue. If the packet is acked, the Tx interrupt handler, which is invoked after each packet transmission, will destroy all the reference of the packet in the software queue.

In order the control the transmission period of the WiFi chip through software, we configure the chip as `CBR_GATED` and set the `CBR_INTERVAL` to the minimum one $1\mu s$. In this mode, the transmission is enabled when `ONE_SHOT_ARM` register is `1`. As soon as the software writes `1` to `ONE_SHOT_ARM`, the transmission will start immediately and the value of `ONE_SHOT_ARM` will be clear by the hardware once the transmission is finished. The transmission is finished when there is no packet or the WiFi chip receives a packet with `end_of_list` flag. Therefore we tag the last packet in each time slot with `end_of_list`.

In order to harness the legacy 802.11 MAC, we first disable the carrier sense of the AP by enabling `FORCE_CHANNEL_IDLE` register. The backoff is disabled by setting `IGNORE_BACKOFF` register. Other inter frame timing values such as SIFS can also be adjusted, but we leave them as default.

C. Uplink Access Control with Polling

To arrange the uplink packets, we only modify the `ath9k` driver. The driver is complied as kernel module, which can be installed by the user just like installing a normal software. First, we configure the chip as `HCF poll gated`, in which the transmission is triggered by the received wireless packet satisfies two conditions: (1) the destination MAC address is the same as the client's MAC address and (2) the subtype field has `QoS Poll` function. Similar to the APs, other 802.11 MAC features, *e.g.* carrier sense and retransmission, are disabled. In practice, we find there are confusion in the WiFi chip for handling QoS Poll packets on whether transmitting the queued packet or sending ack. We solve this by adding `no ack` tag in the poll packet, so that the client can transmit data correctly. Second, we add a handler in the original beacon processing function to enable group polling. The group poll MAC address is write into `STA_ID` register to replace the station's MAC address. The SIFS is set according to a random number. After the group polling period, the register settings are back to normal.

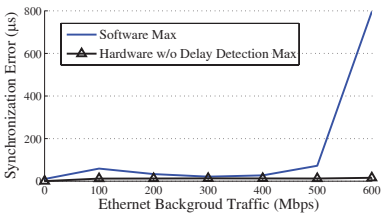
VI. Microbenchmark Evaluation

A. Time Synchronization and Accurate Execution

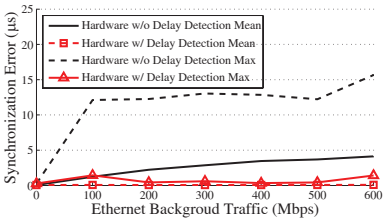
1) Clock Synchronization on Commodity APs: To evaluate the synchronization accuracy, we use the offset parameter in PTP synchronization, which describes the difference between the PTP master clock and the local clock. As our goal is to synchronized all the APs to the same master clock, the offset value is just the synchronization error. In the evaluation, we use two TP-Link 4900 APs as slave clock and record the time offset from the master PC during the synchronization period (10mins). To emulate the background traffic, each AP is connected to a traffic generator, which is continuously generating UDP packets. The maximum background is fixed to 600M, at which the CPU of the AP is in 99% usage for handling sirqs of incoming packets.

We first compare normal PTP software implementation [14] and the normal hardware-assisted PTP implementation [11] without delay detection. Results are shown in Fig. 11. The maximum synchronization error of the software implementation goes to hundreds of microseconds when the background traffic increases to 600Mbps. The value of the error is close to the duration for several WLAN packets. Even with less traffic load, the synchronization error is tens of microseconds due to the software queuing delay as we mentioned in section III-A1. The hardware timestamping is much stable since it is finished by device in hardware.

Second, we compare our scheme using delay detection and the normal hardware-assisted PTP implementation. Although the normal hardware PTP does provide significant stability in providing accurate timestamp, it suffers switching delay when background traffic increases. As shown in Fig. 11(b), APs



(a) Software v.s. Hareware Timestamp



(b) Hareware w/ and w/o Delay Detection

Fig. 11. PTP Synchronization Accuracy.

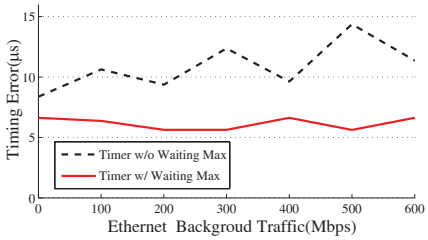


Fig. 12. Timer Timing Error w/ and w/o Waiting.

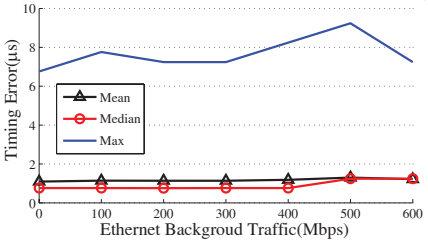
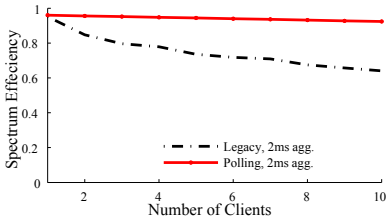
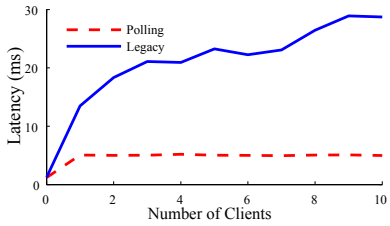


Fig. 13. Synchronized Execution Timing Error.



(a) Polling Efficiency



(b) Polling Delay

Fig. 14. Polling Evaluation for Uplink Scheduling.

with delay detection achieves stable synchronization within $2\mu s$ even under extreme background traffic.

2) Local Execution Timing: As the execution timing is affected by multiple factors across the AP platform, we use a TP-Link WDN4800 wireless card in monitor mode to sniff the air interface to capture the transmitted packets by the AP platform. The received wireless packets in the card are timestamped in μs resolution by the hardware. We fix the time interval of the transmission in the AP platform to 5ms, and measure the inter-packet time spacing as the metrics for the accuracy of the local execution time. Tests with different background traffic are measured with 8000 wireless packets.

Results in Fig. 13 show that the waiting mechanism is effective in increasing the timing accuracy of the scheduled transmission. Note that the accuracy of the measurement of inter-packet time depends on the timestamping accuracy of the sniffer card, the reported accuracy is $5\mu s$ [13]. We believe our timing accuracy is hided by the measurement limitation. Nevertheless, the maximum timing error is bounded to microseconds level and is not higher than $7\mu s$.

3) Synchronized Execution Timing: The overall performance of synchronized execution timing is evaluated with two APs with their local clocks synchronized through PTP with delay detection. The timing measurement method is the same as that of the previous subsubsection. The two APs are assigned with neighboring time slots, thus the inter-packet time is a indicator of the timing accuracy of their synchronized execution.

Results are shown in Fig. 13. Under different traffic load, the error of synchronized execution timing is normally around $1\mu s$ and is less than $10\mu s$ in the worst case.

B. Uplink Traffic Arrangement

We evaluate the uplink traffic arrangement ability with polling in two aspects. The first is the ability in providing high efficient access. The second is the ability in scheduling prioritized uplink traffic. The evaluation is taken in single AP without slot scheduling, or it can be viewed as the local scheduling ability of the AP in the assigned time slot.

Fig. 14 (a) shows the results for the first aspect. The evaluation is performed with one OpenTDMF AP and ten clients. Ten clients are sending uplink traffic to the AP. In legacy method, client contend for channel randomly. Collisions and backoffs reduces the spectrum efficiency. With polling ability and AP use downlink traffic to poll them one by one. With negligible overhead, the polling access increased the throughput by 30%.

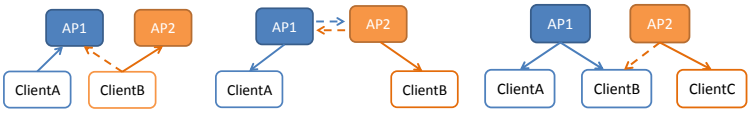
Fig. 14(b) shows the results for the second aspect. the experiment settings are the same are the previous one, except one additional client is added to accept pings from the AP. To illustrate the prioritize arrangement ability of the uplink, the ping packets are set to of high priority. The results shows the polling method can constantly ensure the access delay of the high priority ping compare to legacy random access.

VII. TDMA Scheduling

In this section, we demonstrate the feasibility and benefit of TDMA over commodity WLAN with OpenTDMF enabled devices. Specifically, three examples are used to compare the wireless network performance of scheduled TDMA with legacy WLANs in scenarios containing hidden terminal, exposed terminal and node with requirement on bandwidth-guarantee.

We conduct the evaluation in our indoor office. The topology of the three examples is shown in Fig. 15. APs in TDMA scheduling are enabled with OpenTDMF interface and connected to the central controller through Ethernet. Experiments with legacy WLAN settings share the same infrastructure but APs are with default factory firmware. All the wireless nodes are set to 802.11a mode in channel 165. The transmission are generated with UDP packets with payload of 1470 bytes. The time slot in OpenTDMF is set to $5ms$, which is selected in the same level as the maximum A-MPDU[2] time in 802.11n.

[2]For aggregated packet, the maximum aggregation time is $5.484ms$



(a) Example1 (b) Example2 (c) Example3

Fig. 15. **Topology of Scheduling Examples.** Solid arrow denotes the direction of the data transmission through wireless links. Dash arrow denotes the interference signal.

	Example1			Example2			Example3		
Control Policy	Flow ID	Time Slot	Priority	Flow ID	Time Slot	Priority	Flow ID	Time Slot	Priority
AP1	A->AP1	1 mod 2	Normal	AP1->A	ALL	Normal	AP1->B	1,2 mod 3	High
							AP1->A	ALL	Normal
AP2	B->AP2	0 mod 2	Normal	AP2->B	ALL	Normal	AP2->C	0 mod 3	Normal

Fig. 16. **Scheduling Polices.** Time slot and priority assignment in three interference topology.

Example 1: Uplink Hidden Links Example 1 is designed to compare the performance with hidden links. In Fig. 15 (a), as ClientA and ClientB can not sense each other, the transmission from ClientB generates collisions in AP1.

Throughput results across 10 mins are shown in Fig. 17(a). As CTS/RTS is normally not enabled in APs, legacy 802.11a can not suppress the transmission form ClientB, so almost all the packets are corrupt in AP1 and the throughput falls to 280Kbps. On the contrary, transmission from ClientB to AP2 reaches 28.6Mbps, which is 102 times. The Jain's fairness index of legacy throughput is 0.5049, meaning that the two flows are externally unfair.

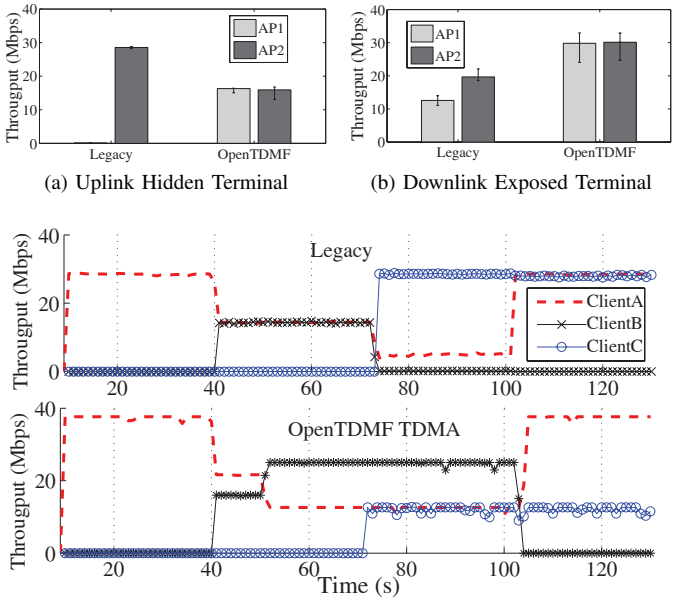
Through the OpenTDMF interface, the central controller issues scheduling policy in Fig. 16 to direct the two uplink flows to different time slots. In this case, AP1 and AP2 generate dedicated polling packets for uplink in indifferent time slots, so that the the packet from ClientB has no chance to collide with packet to AP1. The two flows achieve similar throughput around 16Mbps. The Jain's fairness index of scheduled throughput is 0.9999, meaning that the wireless spectrum is fairly shared.

Example 2: Exposed Links Examples 2 is designed to compare the performance with exposed Links. In Fig. 15 (b), as ClientA and ClientB can not overhear each other, the concurrent transmission is possible.

Throughput results across 10 mins are shown in Fig. 17(b). As AP1 and AP2 can sense each other, the legacy nodes can not fully utilize the spectrum. The throughput of the two clients are 44.1% and 68.9% if the throughput of legacy AP2 in Fig. 17(a), which can be treated the baseline throughput case without sharing. This is mainly because the carrier sense mechanism in AP1 do not allow transmission when AP2 is transmitting and vice versa.

To enable the concurrent transmission opportunity, OpenTDMF schedules the two flows in same slots. Note that OpenTDMF disable carries sense except the group polling periods. The transmission of the two flows will concurrently start. To avoid the collision of ACKs from ClientA and ClientB, we disable per-packet ACK and use software block ACK instead. The total throughput of the two clients is 59.96Mbps, which is 210% of the throughput of the legacy AP2 in Fig. 17(a). The gain is larger than 2 times, this is mainly because of the reduced time in backoff.

Example 3: Bandwidth Guarantee Example 3 is designed to show the ability of OpenTDMF for providing bandwidth guarantee through scheduling. The setting is the same as the previous examples (Fig.1 and Fig.3). The flow of ClientB starts at 40s with 16Mbps and increases to 30Mbps at 50s. The



(a) Uplink Hidden Terminal (b) Downlink Exposed Terminal

(c) Bandwidth Guarantee for ClientB

Fig. 17. Results of Scheduling Examples. We compare the performance of legacy 802.11a with the OpenTDMF Scheduling

flow of ClientB is end at 100s. ClientB requires the bandwidth guarantee for its flow for at least 25Mbps, but both the flow from ClientA and the flow from AP2 contend for wireless medium.

In legacy AP, the traffic in the same AP cannot identify the importance of ClientB's flow and divides the bandwidth equally between ClientA and ClientB. ClientB get about 15Mbps from 40s-60s. Even worse, when AP2 starts transmitting from 50s, the throughput of ClientB decreases dramatically. The packet loss of ClientB is resulted from the collision with packets from AP2, which can not hear AP1. The large amount of retransmission of ClientB also decreases the available airtime of AP1, so the throughput of ClientA decreases to several Mbps before the end ClientB's flow. The bandwidth of ClientB is not guaranteed during its transmission.

OpenTDMF guarantees the bandwidth of ClientB by (1) assigning high priority in time slots and (2) stagging interfering flows from ClientB's slots. First, according to the bandwidth requirement, the scheduler decides to increase ClientB's priority in 2/3 of transmission slots, so that ClientB can grab airtime from ClientA. Second, based on the knowledge of the interference flow from AP2, the scheduler also avoids ClientB and ClientC sharing the same slot. As a result, ClientB's flow are restricted in several slots with high priority (Fig. 16).

In Fig. 17(c). During the whole transmission of ClientB's flow, 2/3 of the bandwidth is accommodated to ClientB with higher priority. Therefore, ClientA can only get 1/3 bandwidth if ClientB has traffic larger than 2/3 bandwidth. From 70-100s, ClientC'flow has no impact on ClientB as they are not in same slots. After ClientB's flow is end in 100, ClientA can obtain full usage of slots, and its throughput goes up to the full bandwidth. As the scheduler keeps the scheduling policy, ClientC still keep 1/3 bandwidth from 100s in the example. The example demonstrates the bandwidth-guarantee for ClientB.

Throughput Gain in Contention-Free Access As transmissions in OpenTDMF do not require contention. The benefit is two folds, first, it avoids collisions after random backoff. Second, the backoff time is reduces. Therefore we notice that the maximum throughput under TDMA scheduling is higher than legacy schemes in the above examples. As the number of nodes in the examples are small, the gain is mainly comes from the saved backoff time. Though theoretical study, the maximum bandwidth for UDP 1470 payload with 54Mbps PHY in legacy 802.11a is 29 Mbps, the same value for TDMA scheduling is 37.6 Mbps, which has 30% gain. For uplink traffic, the upper bond of gain is the same as the downlink. The lower bound of the gain is 13.8%, which is in the situation with dedicated polling packets.

VIII. Related Work

Medium Access Control in WLANs. IEEE 802.11 standards introduce *Point Control Function* (PCF) [6] which is designed for a single AP and partially coordinates transmissions in a centralized manner. Different from PCF, OpenTDMF is designed for the whole WLAN which may consist of tens/hundreds of APs. MiFi [15] proposed a centralized scheme based on PCF to improve fairness for the whole WLAN. Although its goal overlaps with OpenTDMF, it does not target for commodity WLAN devices as us. Time division MACs [8], [16] are also proposed for mesh networks. These schemes rely on air interface for synchronization and coordination, which is limited by the the problems as discussed in Section III. Domino [17] targets WLANs but also solely relies on air interface for coordination.

Coordination on the Backhaul. Recently, some literatures have also discussed the idea of leveraging the backhaul to improve performance for WLAN [1], [2], [5], [7], [13]. These designs generally have different goals, *e.g.* mitigating hidden/exposed terminals [1], [2] which is one of the benefits that TDMA can bring, or enabling new features [5], [7], [13] which is complimentary to TDMA. Centaur [1] also builds on commodity WLAN devices therefore is the most similar to OpenTDMF. However, it focus on hidden/exposed terminals and leaves uplink transmissions unscheduled. Centaur is also different from OpenTDMF in its architecture that it redirects some traffic to the central controller, whereas OpenTDMF controller only manages the decoupled control plane, therefore OpenTDMF has advantage in scalability.

Software Defined Network. OpenTDMF is inspired by SDN and the OpenFlow [3] design. There are also proposals to bring SDN to the wireless WLAN, their goals are different therefore complimentary to us. For example, OpenRF [5] exposes PHY layer interface to control beamforming and interference alignment in commodity devices. OpenRadio [18] reuses physical layer blocks to support multiple technologies such as LTE or WiFi.

IX. Conclusion

This paper introduces OpenTDMF, an architecture that enables TDMA for today's WLANs. To realize the OpenTDMF design on commodity devices, we propose solutions to achieve μs-level time synchronization among APs through backhual network, and develop polling based approach to tightly control uplink transmissions. Experimental results on the OpenTDMF prototype show that OpenTDMF can significantly improve spectrum efficiency, fairness and QoS.

X. Acknowledgement

This work was supported by grants from 973 project 2013CB329006, China NSFC under Grant 61173156, RGC under the contracts CERG 622613, 16212714, HKUST6/CRF/12R, and M-HKUST609/13, the grant from Huawei-HKUST joint lab.

References

[1] V. Shrivastava, N. Ahmed, S. Rayanchu, S. Banerjee, S. Keshav, K. Papagiannaki, and A. Mishra, "CENTAUR: Realizing the full potential of centralized wlans through a hybrid data path," ser. MobiCom '09.

[2] J. Huang, G. Xing, and G. Zhou, "Unleashing exposed terminals in enterprise wlans: A rate adaptation approach," in *IEEE INFOCOM*, 14.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*

[4] V. Shrivastava, S. K. Rayanchu, S. Banerjee, and K. Papagiannaki, "Pie in the sky: Online passive interference estimation for enterprise wlans." in *NSDI*, vol. 11, 2011, p. 20.

[5] S. Kumar, D. Cifuentes, S. Gollakota, and D. Katabi, "Bringing cross-layer MIMO to today's wireless LANs," ser. SIGCOMM '13.

[6] *ANSI/IEEE Std 802.11-2012, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*. IEEE Press.

[7] N. Ahmed, S. Keshav, and K. Papagiannaki, "OmniVoice: A mobile voice solution for small-scale enterprises," ser. MobiHoc '11.

[8] P. Djukic and P. Mohapatra, "Soft-TDMAC: A software TDMA-based MAC over commodity 802.11 hardware," in *IEEE INFOCOM 2009*.

[9] "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," pp. c1–269.

[10] ntp.org: Home of the network time protocol. [Online]. Available: http://www.ntp.org/index.html

[11] Linux ptp project. [Online]. Available: http://linuxptp.sourceforge.net/

[12] Openwrt. [Online]. Available: https://openwrt.org/

[13] J. Manweiler, N. Santhapuri, S. Sen, R. Roy Choudhury, S. Nelakuditi, and K. Munagala, "Order matters: Transmission reordering in wireless networks," ser. MobiCom '09.

[14] The ptp daemon. [Online]. Available: http://ptpd.sourceforge.net/

[15] Y. Bejerano and R. Bhatia, "MiFi: a framework for fairness and QoS assurance for current IEEE 802.11 networks with multiple access points."

[16] D. Koutsonikolas, T. Salonidis, H. Lundgren, P. LeGuyadec, Y. C. Hu, and I. Sheriff, "TDM MAC protocol design and implementation for wireless mesh networks," ser. CoNEXT '08.

[17] W. Zhou, D. Li, K. Srinivasan, and P. Sinha, "DOMINO: Relative scheduling in enterprise wireless LANs," ser. CoNEXT '13.

[18] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "Openradio: a programmable wireless dataplane." ACM.