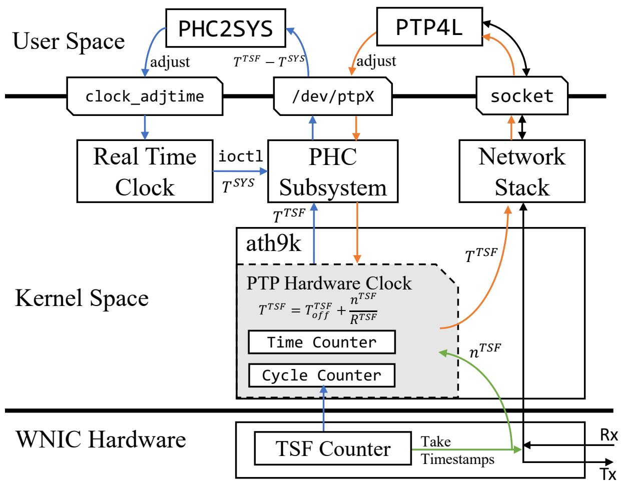


Figure 7: Hardware PTP Implementation in ath9k. This flow chart describes the host of secondary clock. The side of primary clock is similar.

## Supplemental Material

### A. Hardware PTP Implementation in ath9k

We realize hardware PTP by implementing a PTP Hardware Clock (PHC) interface for the ath9k driver. The PHC subsystem of Linux is originally designed for supporting Ethernet NICs hardware PTP. So with our patch of ath9k driver, existing PTP applications can work with Wi-Fi devices without modifications. The source code is available at [7].

Figure 7 shows the case of using `linuxptp` for PTP synchronization. `PTP4L` and `PHC2SYS` are two main components in `linuxptp`. `PTP4L` implements PTP network to synchronize two clocks connected by network. `PHC2SYS` is used to synchronize two clocks in the same host, *i.e.*, system clock and PTP clock. `linuxptp` works with ath9k-PHC interface to achieve synchronizing system clocks of two Wi-Fi connected hosts. The ath9k-PHC interface has three main functions. The working flow is highlighted with three colors in Figure 7.

First (green), the ath9k-PHC realize the mapping function in equation (1), which converts the value of TSF counter $n^{\text{TSF}}$ to the TSF time $T^{\text{TSF}}$. This is done with the help of the time-keeping accessories: the `cyclecounter` and `timecounter` infrastructure provided by Linux are used to convert the $n^{\text{TSF}}$ to $T^{\text{TSF}}$ with two adjustable parameters, $\{T^{\text{TSF}}_{\text{off}}, R^{\text{TSF}}\}$. `timecounter` also ensures $T^{\text{TSF}}$ won't jump back and forth while the underlying cycle counter overflows.

Second (orange), the ath9k-PHC manages the TSF timestamps of PTP packets. TSF timestamps are extracted from the Tx/Rx descriptors of packets, and then converted to $T^{\text{TSF}}$ through the first function. PHC needs to properly deliver $T^{\text{TSF}}$ to the upper layer application, *e.g.*, `PTP4L`. We adopt the standard PTP approach. $T^{\text{TSF}}$ is first packed to the sharedinfo field of its `skbuff`, and then is posted to the error queue of the socket file descriptor opened in the application. The application uses `recvmsg` system call to retrieve the timestamps. The host of the secondary clock can use the timestamps to estimate the clock offset and sync its TSF clock to the primary clock through the `/dev/ptp` interface abstracting the ath9k PHC clock.

Third (blue), the ath9k-PHC provides an interface for PTP subsystem to access the TSF time. It is responsible for reading TSF counter, and mapping it to the TSF time. To sync its system clock to the TSF clock that already synchronized to the primary clock, the PHC subsystem read the TSF clock and the system clock alternately for multiple times and their differences are used by `PHC2SYS` to adjust the parameters of the system clock. Then, the system clock is also synchronized to the primary clock.

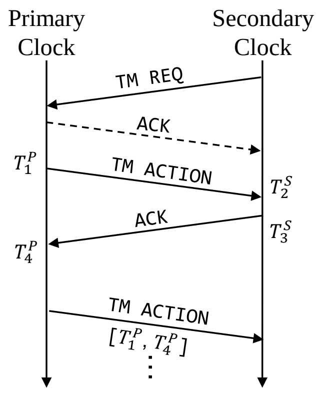


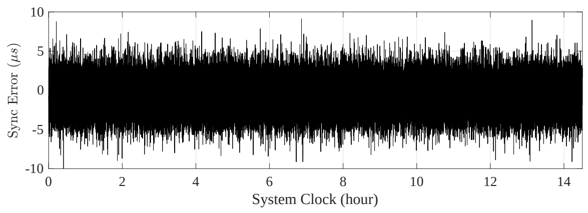
Figure 8: Timing Measurement (TM) Protocol.

### B. Measurement of FTM/TM Synchronization

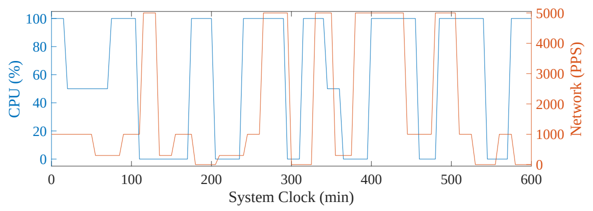
FTM/TM is the latest feature of the IEEE 802.11 standard. Their mechanisms are similar to hardware PTP in using dedicated WNIC hardware for timestamping. FTM and TM differ in resolution. TM is at the level of 10 ns, while FTM is at ps-level. FTM's high time resolution is designed for measuring the time-of-flight (ToF) of radio waves. Another goal of TM/FTM (IEEE 802.1as Clause 12 [2]) is to extend the LAN PTP synchronization to Wi-Fi devices, targeting the same problem as our PTP implementations.

As shown in Figure 8, the secondary WNIC initiates TM procedures. The primary WNIC sends four `TM ACTION` packets per second. After receiving each `TM ACTION` packet, the secondary WNIC reports four hardware TM timestamps ($T_1^{P\text{TM}}$, $T_4^{P\text{TM}}$, $T_2^{S\text{TM}}$, $T_3^{S\text{TM}}$) and one timestamp by the system clock. The system time $T^{S_{\text{SYS}}}$ is taken when the `TM ACTION` packet is received. From the four TM timestamps, $\Delta$, *i.e.*, the offset between the TM clocks of the two WNICs, can be calculated. Note that from $\Delta$, one cannot infer whether there are biased errors, but the variation of $\Delta$ is related to the stability of synchronization.

Since there is a fixed frequency offset between two TM clocks, $\Delta$ is either monotonic increasing or decreasing. For this reason, we leverage $T^{S_{\text{SYS}}}$ as the time reference to extract



(a) Error v.s. Time



(b) Sample of Load Pattern

Figure 9: Software PTP Long-term Sync Results. (a) Raw error traces (mean=0.17 $\mu s$, std=1.8 $\mu s$). (b) Sample trace of random CPU and network load applied to the client.

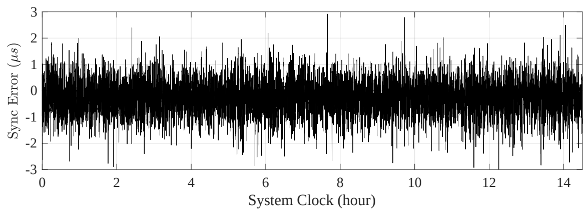


Figure 10: Hardware PTP Long-term Sync Results. Raw error traces (mean=-0.09 $\mu s$, std=0.63 $\mu s$).

the temporal variance in $\Delta$. A line $\Delta_i = a \cdot T_i^{S_{\mathrm{SYS}}} + b$ is used to fit the offset values. The std of TM synchronization is represented by the std of the line fitting residuals. The results are shown in Table 9.

Additionally, we note the std of the offset (around 0.3 $\mu s$) is still very large compared with its 10 ns-level timestamp resolution. We find this is because the actual resolution of TM timestamps of this WNIC and firmware is 1 $\mu s$. The evidence is the TM timestamps obtained either from the driver or by sniffing `TM ACTION` packet have the following relation: $T_1^{P_{TM}} \equiv T_4^{P_{TM}} \pmod{100}, T_2^{S_{TM}} \equiv T_3^{S_{TM}} \pmod{100}$. It is likely the firmware or hardware intentionally mask the resolution.

## C. Figures of Long-term Measurement

Figure 9 and Figure 10 for long-term error trace.