SWAN: Stitched Wi-Fi ANtennas
Yaxiong Xie, Yanbo Zhang, Jansen Christian Liando, Mo Li
School of Computer Engineering, Nanyang Technological University, Singapore
{yxxie,zhang.yanbo,cjansen,limo}@ntu.edu.sg

ABSTRACT
This paper presents our experience in designing, implementing, testing, and applying a general-purpose antenna extension solution with commodity Wi-Fi. The proposed solution, SWAN, builds an array of stitched antennas extended from the radio chains of commodity Wi-Fi. SWAN has low hardware cost and provides easy-to-use interfaces embedded in the Linux kernel. Two application cases for wireless sensing and communication are presented that proves the usefulness of the solution. SWAN is able to provide over 3× performance improvement on Wi-Fi azimuth estimation and localization and over 30% improvement on Wi-Fi throughput over original Wi-Fi AP with three fixed antennas.

CCS CONCEPTS
• Networks → Wireless access points, base stations and infrastructure;

KEYWORDS
Antenna Extension, MIMO, Localization, AoA Estimation, Antenna Selection, Antenna Switching, Spatial Diversity

ACM Reference Format:

1 INTRODUCTION
An array of many antennas helps a wide range of wireless sensing and communication applications, e.g., a phased-array antenna enables RF sensing system to estimate the signal azimuth with higher resolution and better accuracy [17, 51], which further supports many applications including indoor localization, passive tracking, activity recognition, etc. Communication also benefits from the spatial diversity or multiplexing gain provided by a large antenna array, e.g., antenna selection significantly improves the communication throughput of MIMO systems.

Most commodity Wi-Fi network interface cards (NIC), however, only support three antennas (some Wi-Fi APs have five or six antennas but they are separately used for 2.4 GHz and 5 GHz radios). Increasing the antenna number results in increased design complexity, and thus, high cost. While the antenna elements (either printed or external antennas) themselves are usually inexpensive, the RF components of the radio chains, including low-noise amplifiers, downconverters, analog-to-digital (ADC) or digital-to-analog converters (DAC), are expensive and they do not follow Moore’s law. The digital processing complexity also increases exponentially with the antenna number [16, 43], e.g., BigStation [56] has to shift the basedband computation to PC software using software-define-radio and requires 15 PCs to support real-time signal processing of 12 antennas. A many antenna Wi-Fi NIC, therefore, requires extremely high speed baseband processor, and thus high hardware cost.

While tremendous efforts have been made to building many antenna systems, most of those efforts focus on building specially purposed systems like large scale MIMO [8, 16, 33, 49, 56], full-duplex communication [5, 40, 44], synthetic aperture radar (SAR) [1, 27, 38], and with special hardware support [7, 49] or software-defined-radio platforms [43, 47]. Few studies have been performed with commodity Wi-Fi. Phaser [11] is a recent attempt to build large phased-array on commodity Wi-Fi by combing multiple 3-antenna Wi-Fi NICs. Phaser makes use of the radio chains of multiple Wi-Fi NICs and by doing that introduces significant hardware cost - more radio chains than antennas are used in Phaser (one radio chain from each of those NICs is connected with each other for synchronization purpose and thus wasted). Phaser is a special purpose platform dedicated to angle-of-arrival (AoA) estimation. Since the NICs are on different Wi-Fi APs, they do not work cooperatively as a cohort to provide general Wi-Fi communication services.

In this paper, we propose SWAN (Stitched Wi-Fi ANtennas) as a low-cost general-purpose antenna extension to commodity Wi-Fi. Instead of including more radio chains or NICs which are expensive, we provide an array of stitched antennas extended from the original three radio chains of the...
Wi-Fi AP. The antennas are stitched with RF switches, and the AP is able to configure the RF switches to select a combination of antennas for transmitting or receiving a packet. All components are connected through standard hardware interfaces and work in a plug-and-play mode, without any hardware modifications at the AP side. The total hardware cost of SWAN including antennas, RF switches, an external Arduino board for control, etc., is below $100. We show that SWAN can be easily scaled to support tens or even hundreds of antennas to one commodity AP.

SWAN also provides a set of general user programming interfaces embedded in the Linux kernel of the Wi-Fi AP. SWAN follows the general Wi-Fi transmission and reception procedures in Linux kernel so the users of SWAN can simply use existing socket interfaces. At the same time, SWAN allows its users to configure the antenna combinations to send and receive packets through simple descriptors. The control details for accurately and swiftly configuring RF switches as well as annotating packets are made transparent to the users of SWAN. The code of SWAN is public[30]. SWAN devises special solutions to the challenges that arise from the system requirement in fast control exchange as well as reliable packet annotation, the effectiveness of which are evaluated with experiments on our 12-antenna SWAN prototype.

Being a general purpose platform, SWAN allows easy scripting to task underlying APIs to support different user applications. In this paper, we describe our experience in using SWAN for two application cases: (1) We build a virtual phased-array and form a circular array for wireless azimuth estimation and indoor localization; (2) We build a MIMO communication system with antenna selection that harnesses spatial diversity to improve throughput. Our experiment results show significant improvements brought by SWAN to the performance of both application cases.

The rest of the paper is structured as follows. We start with the design of SWAN in §2. §3 and §4 present our experiences applying SWAN to building a virtual phased-array for wireless sensing and to improving MIMO communication with antenna selection, respectively. §5 illustrates how SWAN can scale to support a large number of antennas. §6 discusses related works in the field. §7 concludes this paper.

2 SWAN DESIGN

We introduce the detailed design of SWAN. We begin with the architecture and hardware design, followed by the user interface and workflow of SWAN.

2.1 Architecture

Architecture. SWAN extends the antenna array of commodity AP by connecting each radio chain to multiple antennas through RF switches, which features a plug-and-play mode.

Figure 1: (a) The architecture of SWAN. (b) A prototype built with a WPJ558, three PE42442 RF switches and an Arduino board.

Figure 1 (a) illustrates the architecture of SWAN based on single pole four throws (SP4T) RF switches as an example. Specifically, each of the three radio chains of the AP is connected to four antennas through SP4T RF switch using coaxial cable (RG50) and a 12-antenna array is thus obtained. SWAN uses an external micro-controller (MCU) that connects to all RF switches and controls the antenna switch through GPIO. Finally, the external MCU is connected to the AP through Ethernet cable which enables the control exchange between the AP and the MCU. For each radio chain, the AP is able to pick up any one out of the four antennas to transmit or receive a Wi-Fi packet, which gives $4 \times 4$ possible antenna combinations for configuring the three radio chains. The AP configures the selected antenna combination and signals the MCU to switch the three SP4Ts to connect to the selected antennas for packet transmission and reception.

The system can easily be scaled to a larger array of antennas by applying higher split RF switches (e.g., using SP8T for eight throws, which gives us $8 \times 8 \times 8$ antenna combinations), or cascaded connection of multiple RF switches (which we will detail in Section 5).

Hardware. We build a SWAN prototype with commercial-off-the-shelf (COTS) and low-cost hardware, as shown in Figure 1 (b). The hardware used can be easily replaced with other general COTS substitutes of similar functionality.

We test with two different AP models - COMPEX WPJ558 and TP-Link WDR4300 (cost ~$67). The SP4T RF switches used are Peregrine PE42442 (cost ~$1.32), which provides fast switch speed, i.e., 225ns, low signal loss, i.e., 0.8 and 1.0 dB in 2.4GHz and 5GHz respectively, and high linearity, i.e., $58\text{dBm IIP3}$ and $110\text{dBm IIP2}$ [18]. We use an Arduino board (cost ~$22.9) to serve as the external MCU with its GPIO to control the PE42442 switches. Specifically, we connect two GPIO pins of the Arduino to a two-pin voltage CMOS control interface of PE42442. Arduino MCU controls PE42442 by changing the voltage state on the two control pins. The Arduino board
where each entry records a frame ID.

Transmission flow. SWAN follows the normal packet transmission flow of Linux kernel. In addition, SWAN allows associating each packet with a 3-byte descriptor ant-comb to indicate which antenna combination is to be used to transmit the packet. Each byte of ant-comb indicates the antenna to use for one radio chain. The value of ant-comb descriptor is set by the user application when generating the Wi-Fi packet and the antenna combination is thus selected (otherwise the default antenna combination will be used).

Since the Wi-Fi NIC takes a period of time to gain the channel access before transmitting the packet, SWAN makes use of that time interval to signal the choice of antenna combination to the Arduino MCU and switch the antenna accordingly before the actual transmission of every packet.

Figure 2 (a) illustrates the workflow for transmitting a packet in SWAN. Wi-Fi NIC prepares a transmitting queue (TX-PKT Queue in Figure 2 (a)) to buffer all data packets delivered from Linux kernel and pending to be transmitted. To track the packets that have been delivered to NIC, common NIC drivers in the kernel (e.g., ath9k for most Qualcomm chips) assign a unique frame ID for every packet, e.g., tx_desc_id in ath9k. SWAN thus, maintains an in-transmission packet queue for all packets ath9k has delivered to NIC, where each entry records a frame ID tx_desc_id as well as its antenna combination ant-comb. When NIC successfully transmits a packet, an 802.11 ACK frame will be received and in the interrupt request (IRQ) handler of 802.11 ACK frame, SWAN does two jobs – (1) to dequeue the relevant record for the transmitted packet as normally done in ath9k IRQ procedure, and (2) to obtain the antenna combination descriptor for the next packet and send it to the Arduino. The Arduino MCU then configures the RF switches before the next packet is on its way to the antenna.

When the transmission of one packet in the NIC queue fails, an interrupt is still triggered by the NIC. SWAN dequeues the failed packet from the in-transmission packet queue according to the frame ID. If a packet is passed to ath9k when the in-transmission queue is still empty, i.e., no prior packet is buffered in the NIC, SWAN still enqueues its frame ID and ant-comb but immediately sends the antenna combination to the Arduino without waiting for the ACK interrupt triggered by the successful reception of last packet.

Reception flow. SWAN follows the packet reception flow of Linux kernel, but provides additional functionalities. Figure 2 (b) illustrates the workflow for receiving a packet in SWAN. First, SWAN tags a 3-byte descriptor ant-comb to every received packet, which annotates the antenna combination used to receive the current packet. The ant-comb descriptor can be extracted by users of SWAN to facilitate various application purposes. Second, SWAN adds a control interface config_rx_ant() in the NIC driver for setting a sequence of receiving antenna combinations. Specifically, a receiving antenna queue (RX-ANT Queue in Figure 2(b)) is constructed where each entry gives an ant-comb indicating the antenna combination to receive the packet.

In the IRQ handler of each received data packet, SWAN does two jobs – (1) to examine the actual antenna combination used to receive the packet and annotate the packet with the ant-comb descriptor, and (2) to decide the antenna combination used to receive the next packet and send it to the Arduino. SWAN decides the antenna combination based on the next user defined ant-comb in the receiving antenna queue. The entry is dequeued when a successful packet is received using that antenna combination. If the queue is empty, the default ant-comb is used for receiving new packets.

Challenges. The designed user interface and workflow ensures correct control logic in SWAN to operate the stitched antennas. In order to make the solution practically feasible, however, we face two challenges. First, SWAN requires fast...
control exchange between the AP and the Arduino to achieve μs-level antenna switch. This is necessary to accommodate the antenna switch within short Wi-Fi transmission and reception intervals. Second, SWAN requires reliable packet annotation, i.e., the antenna combination ant-comb applied to each packet is accurate. There are several ways that could lead to wrong antenna configuration in SWAN, e.g., failures in switching the antennas due to the lost of control commands, or delayed antenna switching that results in incorrect antenna combinations used to transmit or receive the target packet. In the following sections, we will detail our techniques to address the two challenges.

2.3 Fast control exchange

Time stringency. 802.11 Wi-Fi transmission and reception impose stringent time requirement on antenna switching. Wi-Fi adopts CSMA where the shortest time interval from AP and the Arduino, e.g., Time stringency. Therefore, SWAN has to signal the Arduino and switch the antenna combination within the 34 μs DIFS duration, to guarantee correct antenna configurations.

Maintaining a flow over the Ethernet cable between the AP and the Arduino, e.g., TCP, is a natural choice for conveying those control messages, which, however, is known time consuming (hundreds of microseconds to milliseconds of latency). We propose a fast control exchange method which rides on Linux TCP sockets but bypasses all time-consuming operations. First, the antenna combinations are finite, e.g., $4 \times 4 \times 4 = 64$ combinations with three SP4T switches, so a total number of 64 repeated ant-comb control messages are used. We thus can build template TCP sockets beforehand and reuse them in Linux kernel without instantly constructing them. Second, we bypass all TCP retransmission and flow control procedures to save time because the Ethernet cable directly connects the AP and the Arduino, and only serves the control exchange between them. Third, there is no need for parsing the TCP or IP header as SWAN is the only user of the physical link, so we let the Arduino skip the standard packet inspection procedure in the Linux Ethernet driver.

**Packet sketcher.** We build a packet sketcher to "sketch" template packets for control exchange. We build a TCP connection and then send all possible antenna combinations in sequence, which triggers the Ethernet NIC driver (ag71xx for Atheros AR8327N in our system setting) to construct all relevant TCP packets and send through $ag71xx\_hard\_start\_xmit()$ to the NIC hardware. The packet is buffered in $sk\_buff$ for each transmission. We modify $ag71xx\_hard\_start\_xmit()$ of the AP to store all constructed packets as templates to signal different antenna combinations. Figure 3 illustrates the process. Packet sketching is immediately done after the TCP connection is established and the stored packet templates can be used as long as the TCP connection lasts.

**In operation.** SWAN swiftly react to the Wi-Fi packet transmission or reception. The antenna combination ant-comb is sent to the Arduino by the 802.11 ACK or data frame IRQ handler of ath9k. Specifically, the AP’s IRQ handler directly sends over the $sk\_buff$ corresponding to the selected antenna combination through $ag71xx\_hard\_start\_xmit()$.

On the Arduino side, the Ethernet NIC stores the received packets in a linear buffer and handles the buffer to the NIC driver. Since SWAN is the only application running on the Ethernet link, there are only two types of packets that are supposed to be received by the Arduino — (1) the TCP control packets including SYN, ACK, and FIN, which the Arduino’s NIC driver needs to decode and then pass to the TCP stack for maintaining the TCP connection, and (2) the AP’s command that conveys the antenna combination ant-comb, which due to the time limit should not be passed to TCP stack for further inspection.

SWAN is able to distinguish the two types of packets by solely examining their packet length. The command packet has a fixed size of 90 bytes including 20 byte payload, and the TCP control packet only has the fixed header and is of 70 bytes. Therefore, every time when the Ethernet driver receives a packet, it examines the packet length and directly extracts the antenna combination from the payload (always at the end of the packet) without further parsing the entire packet headers. By doing the above SWAN effectively conveys control commands through the TCP pipe.

![Figure 3: Packet sketching in SWAN.](image)

2.4 Reliable packet annotation

As we previously mentioned, the antenna configuration in SWAN may fail due to the lost of control commands or delayed antenna switch that takes effect after the actual packet is transmitted or received. It is important for SWAN to detect such failures and feedback to its users in a reliable way, so the users of SWAN are aware and thus able to adjust their operations accordingly. SWAN returns an ant-comb descriptor to the user application after every packet transmission or reception, which tells the actual antenna configuration used.
for that packet. We use the value of 0xFFFFFFF to indicate a failed or uncertain antenna configuration.

For every antenna configuration command, SWAN requires the Arduino to send back an TCP packet for acknowledging the successful reception of the transmitted command. We denote such a packet as SWAN ACK. The AP waits for the SWAN ACK from the Arduino. The SWAN ACK is considered not received if it does not yet arrive by the completion of current packet transmission or reception (until the 802.11 ACK or data frame interrupt generated from the NIC chip). Figure 4 gives an example timeline when the AP transmits a packet. SWAN annotates each successfully transmitted or received packet based on the relationship of the following three time parameters: 1) the switching delay $t_{sw}$ between the completion of the previous packet, e.g., at $t_1$ in Figure 4, and the completion of antenna switch by the Arduino at $t_4$; 2) the channel accessing interval $t_{ac}$ before the start of the packet transmission or reception, as shown in Figure 4; and 3) the air time $t_{air}$ of the received or transmitted packet.

There are four possible cases:

- **Case 1:** The switching delay $t_{sw}$ is smaller than the channel accessing interval $t_{ac}$, i.e., $t_{sw} < t_{ac}$. In this case, the antenna configuration is deemed successfully done before the actual Wi-Fi packet transmission or reception. Therefore, SWAN annotates the packet with the expected antenna combinations ant-comb.

- **Case 2:** The switching delay $t_{sw}$ is larger than the channel accessing interval $t_{ac}$, but smaller than $t_{ac} + t_{air}$. In such a case, the antenna switching behavior might happen in the middle of the transmission or reception of the current packet, so the configuration is considered failed or uncertain. SWAN annotates the packet with 0xFFFFFFF for the user application to take further actions.

- **Case 3:** The switching delay $t_{sw}$ is larger than the packet air time, i.e., $t_{sw} > t_{ac} + t_{air}$. In this case, the antenna switching behavior is done after the completion of current packet transmission or reception, so SWAN uses the previous ant-comb to annotate the packet. The user application may need to take further actions with such a case.

- **Case 4:** In case the SWAN ACK from the Arduino is not received, SWAN annotates the packet with 0xFFFFFFF.

**Antenna switching delay.** Directly measuring the delay by $t_{sw} = t_4 - t_1$ is inaccurate, since the AP and the Arduino are not synchronized. SWAN thus divides the delay $t_{sw}$ into three parts, and measures them separately. The first two parts are the software delay $t_{AP}$ at the AP side and $t_{Ar}$ at the Arduino side, which are measured according to the clock of AP and Arduino respectively. Specifically, $t_{AP}$ describes the interval that the AP takes in its kernel to prepare the control command and is measured as $t_{AP} = t_2 - t_1$. $t_{Ar}$ describes the time interval between the Arduino receives the control command at $t_3$ and performs the antenna switching at $t_4$, and is measured as $t_{Ar} = t_4 - t_3$. The third part is the transmission delay $t_{eth}$ on the Ethernet cable between the AP and the Arduino, which involves the clocks at the AP and the Arduino, and cannot be measured directly. SWAN makes use of the SWAN ACK from Arduino and measures the roundtrip-time (RTT). Specifically, the Arduino measures the time $t_5$ it sends the SWAN ACK and uses a separate timing packet to deliver $t_5$, $t_3$, and $t_{Ar}$ to the AP, as illustrated in Figure 4.

The transmission delay over Ethernet is thus calculated as:

$$t_{eth} = \frac{[t_6 - t_2] - (t_5 - t_3)}{2}$$

where $t_6$ is the AP’s timestamp in receiving the Arduino ACK. The total switching delay $t_{sw}$ is obtained as $t_{AP} + t_{eth} + t_{Ar}$. Since the SWAN ACK is fixed, Arduino can prepare the template beforehand. The timing packet is also prepared beforehand, which is a TCP template packet with empty payload. The timing parameter $t_3$, $t_5$, and $t_{Ar}$ are then instantly appended to the end of the template as the payload.

**Channel accessing interval.** To measure the channel accessing interval $t_{ac}$, SWAN samples the end of previous transmission or reception, e.g., $t_1$ in Figure 4, and the start of transmitting the current packet, i.e., the start of transmitting the preamble $t_{pre}$. The IRQ handler of ath9k logs $t_1$. The Wi-Fi NIC logs $t_{pre}$ of every packet it successfully transmits or receives, and then reports it to ath9k along with the received data frame or the ACK of transmitted frame. The channel accessing interval is thus measured as $t_{ac} = t_{pre} - t_1$.

**Packet air time.** The air time $t_{air}$ of one packets consists of the air time of the preamble and the payload. The air time of packet preamble is known, e.g., 40 μs for 802.11n packets. The air time of payload can be calculated using the packet payload length and the setting of the data rate.

### 2.5 Evaluation

**Methodology.** We conduct experiments with the 12-antenna prototype that we build with WPJ558 and the Arduino board,
We let the SWAN AP first transmit 2 \times 10^5 packets and then receive 2 \times 10^5 packets from the client. The AP switches the antenna combination for every transmitted and received packet so that 4 \times 10^5 control messages in total are exchanged between the AP and the Arduino.

**Antenna switching delay.** Our experiment results demonstrate that SWAN can promptly send the control commands from the AP to the Arduino, which ensures short antenna switching delay \( t_{sw} \). We measure the software delay \( t_{sw} \) at the Arduino side, the software delay \( t_{AP} \) at the AP side, and the transmission delay \( t_{eth} \), described in Section §2.4. We sum the three delays and obtain the overall switching delay \( t_{sw} \). Figure 5 plots the statistics of those delays from the 4 \times 10^5 measurements. We see that the software delay \( t_{Ar} \) and \( t_{AP} \) are smaller than 4.0 \( \mu s \) and 5.4 \( \mu s \), respectively for 90% cases, which credits to the packet sketching and fast control exchange approaches used in SWAN. The transmission delay \( t_{eth} \) has a 90% quantile of 22.4 \( \mu s \). Therefore, the overall switching delay \( t_{sw} \) is below 29.5 \( \mu s \) for 90% control exchanges. The switching delay \( t_{sw} \) is smaller than DIFS (34 \( \mu s \)) for 98.5% cases, which means 98.5% control commands are guaranteed to take effect in transmitting or receiving the packet with the correct antenna configurations. For the rest 1.5% cases, we find that 89.8% of them are still successfully transmitted or received with correct antenna configurations since their switching delays \( t_{sw} \) are smaller than the channel accessing interval \( t_{ac} \).

**Channel accessing interval.** The channel accessing interval \( t_{ac} \) is the time before the packet is transmitted or received on the antennas. DIFS sets a lower bound for \( t_{ac} \), but its actual duration is much larger in statistics. We measure the interval \( t_{ac} \) for all 4 \times 10^5 transmitted and received packets and plot the \( t_{ac} \) statistics in Figure 6. We see that, Wi-Fi selects the back-off window size from [0,16] randomly and selects the shortest window size 0 for only 5.8% packets. We also observe that 7.9% packets have large \( t_{ac} \) (> 173\( \mu s \)), which are caused by various reasons like packet retransmissions, channel contention failures and so on. Overall most packet transmissions and receptions give sufficient channel access interval for SWAN to control the antenna configurations in time. Over 99.8% of those packets are configured with correct antenna combinations.

**SWAN ACK delay.** We also measure the delay of the reception of the SWAN ACK \( t_{ac} \) and timing packet \( t_{ac} \) from the Arduino to the AP, and plot the statistics in Figure 7. We see that the delays are smaller than 60.3 \( \mu s \) and 67.4 \( \mu s \) for the SWAN ACK and timing packet for 90% cases. To help better understand such delay, we compare the delay with the DIFS + preamble = 74 \( \mu s \), which sets a lower bound of the time interval before the completion of the current packet transmission or reception (assuming back-off window = 0 and payload = 0). We see from Figure 7 that 99.0% of SWAN ACKs and 97.8% timing packets are received before DIFS + preamble. Therefore, the processing of the control ACK and timing packet are less likely to interfere with the handling of new packet transmission in the AP’s NIC driver.

<table>
<thead>
<tr>
<th>Success ratio</th>
<th>Transmission</th>
<th>Reception</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>99.78%</td>
<td>99.92%</td>
<td>99.97%</td>
</tr>
</tbody>
</table>

**Table 1: Success rate of antenna switching and packet annotation of SWAN.**

**Overall success ratio.** We assess the overall success ratio of antenna switching and the accuracy in packet annotation. We let the SWAN AP communicate with the client over a static channel and measures the CSI of each antenna as its signature. When a packet is received, we can compare its received CSI with all antenna signatures and figure out the true antenna configuration used for that packet. We compare the true antenna configuration with the ant-comb that
the AP sends to the Arduino and derive the success ratio in antenna configuration. We also compare the true antenna configuration with the annotated ant-comb of each packet derive the success ratio of annotation. Table 1 summarizes the success ratios for transmission antenna switching, reception antenna switching, and packet annotation. We see that SWAN successfully switches the antenna for 99.78% packet transmissions and 99.92% packet receptions. The packet annotation is correct for 99.97% packets. Among the unsuccessful switching cases, we further observe that the percentage of case 4 is below 0.001% since packet transmission failure over Ethernet cable is rare. The percentage of case 2 is below 1% as switching antenna in the middle of transmitting or receiving a packet results in packet loss for most of the time and retransmitting the packets allows SWAN adequate time to configure the antenna correctly. At last, around 99% switching failures fall into case 3.

3 CASE I: BUILDING VIRTUAL PHASED-ARRAY

Phased-array is widely used as a sensing interface to estimate angle-of-arrival (AoA) or angle-of-departure (AoD) [3, 29, 31]. Recent advances [11, 22, 23, 26, 51, 53] have built phased-array on commodity Wi-Fi with three antennas. SWAN enables a new opportunity to greatly increase the antenna number in the phased-array and thus provide angle estimation with much finer resolution and higher accuracy.

3.1 Building virtual phased-array

We measure CSI from all antennas to obtain a virtual phased-array. We use Atheros-CSI-Tool [50] which extracts the CSI of all data subcarriers and collect CSI from all 12 antennas on our SWAN prototype (12 × 56 matrix on a 20 MHz Wi-Fi channel). Limited by the radio chains on the AP, we cannot concurrently measure all 12 antennas. Instead we measure the 12 antennas when receiving multiple packets and synthesize the CSI matrix from individual CSI measurements.

CSI grouping. The phase error across different antennas, however, hinders us from directly grouping CSI matrices across packets. A phased-array must preserve accurate difference of CSI phases across antennas, which however is impaired by the carrier frequency offset (CFO) that adds random phase offsets to individual packets [50]. The intuition of our solution is illustrated in Figure 8. When we measure the CSI of first three antennas (antenna 1-3 in the figure) with the first packet, due to CFO there is an offset $e_1$ of the measured phases on all the three antennas from their true phases. Nevertheless the phase differences are accurate across themselves. When we measure the CSI of the second three antennas (antennas 4-6), similarly our measured phases are subject to an offset $e_2 \neq e_1$, as Figure 8 (a) depicts. In order to correctly group the two measurements, we need to remove the difference $\Delta e = e_2 - e_1$, which however is impractical to measure with commodity Wi-Fi [50].

We propose to cancel $\Delta e = e_2 - e_1$ by using a redundant antenna in both consecutive packet measurements. As shown in Figure 8 (b), we keep the first antenna in our antenna combination for the second measurement (antenna 1, 4, and 5). By aligning the phase offset of antenna 1 with its offset in the previous measurement, we align all phase measurements across five measured antennas, which preserve the phase difference across all antennas. A scheduled antenna measurement scheme is accordingly designed to ensure one redundant antenna included in any consecutive measurements (so we obtain CSI from two new antennas in each measurement). Script 1 gives the user application script for the scheduled antenna measurements.

It is worth emphasizing that the antenna measurement process can be completed within very short time. For example, only five packets are needed to collect CSI from all 12 antennas in Script 1. Each packet takes around 190μs and the entire measurement process can be completed in less than 1ms, which is shorter than the coherence time in many static and mobile scenarios. Therefore, we are able to estimate the channel’s intrinsic characteristics like the AoA of signal propagation paths. The theoretical upper bound that our antenna measurement scheme can support under various channel dynamics is derived in Section §3.3. Unlike the time varying offsets $e_1$ and $e_2$, the phase offsets across radio chains are fixed and can be accurately measured by connecting the radio chains of transceivers. We apply the phase calibration presented in [53] to eliminate the phase offsets introduced by radio chains.

3.2 AoA estimation

We perform AoA estimation atop the phased-array. Uniform linear array (ULA) is widely used because of the ease of signal processing, but only provides a field view of 120°, i.e., [30°, 150°], due to its linear arrangement of antennas [17]. On the other hand, uniform circular array (UCA) provides a field view of 360° and maximize the coverage in sensing
where MUSIC to deal with the coherent signals received from antennas. To estimate those three unknown parameters, we minimize the difference between the signal $y(t)$ we receive:

$$[T, \Theta, A] = \arg \min_{T,\Theta,A} \|y(t) - s(t)\|$$

where $T = [t_1, t_2, \ldots, t_L]^T$, $\Theta = [\theta_1, \theta_2, \ldots, \theta_L]^T$ and $A = [a_1, a_2, \ldots, a_L]^T$ are the ToF, AoA and amplitude of $L$ paths. We borrow the method of xD-Track [51] to solve the optimization problem described in Eqn 4 to obtain $\Theta$.

### 3.3 Evaluation

We evaluate the performance of SWAN in AoA estimation in this section. We also build and evaluate an indoor localization system based on the estimated AoA.

**Methodology.** We build a 9-antenna UCA for Wi-Fi APs in our design ($N=9$). In the UCA, the distance between two adjacent antennas are fixed to $\lambda/2$, where $\lambda$ is the wavelength of the carrier wave. All the antennas are located at the vertices of a nonagon, with an edge length of $\lambda/2$. The radius of the UCA is $r = \lambda/(4 \sin 20^\circ)$. Figure 9 (a) depicts the testbed setting of the UCA. The antenna distance between antennas in ULA is also fixed to be $\lambda/2$. We also build a 3-antenna ULA for comparison. The distance between adjacent antennas is also fixed to be $\lambda/2$.

We configure an Arduino YUN to be a Wi-Fi signal emitter. A commodity Wi-Fi AP (WPJ558 or WDR4300) works in monitor mode to receive the signal sent from the Arduino YUN. Line-of-Sight (LoS) between the AP and the Arduino YUN is ensured during the experiment. The AP collects the CSI from the UCA and sends to a server to process and estimate AoA. Figure 9 (a) shows the 9-antenna UCA that we build for AoA estimation.

We derive the relationship between signal phases across antennas in the UCA. For an UCA with radius $r$ and consisting of $N$ uniformly distributed antennas, as shown in Figure 9 (b), the angular position of its $n$th antenna is given by $n\phi$, where $\phi = 2\pi \left(\frac{n}{N}\right)$ and $n = 1, 2, \ldots, N$. Following the geometry in Figure 9 (b), the steering vector of the UCA is given by:

$$c(\theta) = \begin{bmatrix} e^{2\pi \frac{r \cos(\theta - \phi)}{\lambda}} \\ e^{2\pi \frac{r \cos(2\theta - \phi)}{\lambda}} \\ \vdots \\ e^{2\pi \frac{r \cos(N\theta - \phi)}{\lambda}} \end{bmatrix}$$

where $\theta$ is the AoA of an incident wave.

MUSIC cannot be applied to UCA for AoA estimation since it cannot work with mutually coupled signals received from antenna array. Spatial smoothing can be applied together with MUSIC to deal with the coherent signals received from ULA, which, however, fails for signals received from UCA. Therefore we adopt a maximum likelihood estimation algorithm that works with mutually coupled signals. If we denote the transmitted signal as $u(t)$, the signal received by the UCA can be modeled as:

$$s(t) = \sum_{l=1}^{L} a_l c(\theta_l) u(t - \tau_l)$$

where $L$, $\theta$, $\tau$, and $\alpha$ is the number of multipath signals, the AoA, the time of flight (ToF), and the amplitude, respectively. To estimate those three unknown parameters, we minimize the difference between the signal $s(t)$ we model and the signal $y(t)$ we receive:

$$[T, \Theta, A] = \arg \min_{T,\Theta,A} \|y(t) - s(t)\|$$

where $T = [t_1, t_2, \ldots, t_L]^T$, $\Theta = [\theta_1, \theta_2, \ldots, \theta_L]^T$ and $A = [a_1, a_2, \ldots, a_L]^T$ are the ToF, AoA and amplitude of $L$ paths. We borrow the method of xD-Track [51] to solve the optimization problem described in Eqn 4 to obtain $\Theta$.

**AoA accuracy.** We collect CSI measured from both the UCA and the ULA at each of 200 test locations and estimate the

---

**Script 1: Scheduled antenna measurement**

```
Input: No. of antennas $N_r$ attached to each chain

1 ant-comb[3]=0;  // 3-byte descriptor
2 m = 0;
3 for i = 1; i ≤ ⌈ $\frac{N_r}{8}$ ⌉ do
4   for j = 0; j ≤ 3; j ≠ m do
5     ant-comb[j]++;
6   if ant-comb[j] > $N_r$, then
7     m = (m + 1) mod 3;
8 config_rx_ant(ant-comb[3]);
9 [pkt_i, csi_i, ant-comb[1]] = recv_pkt()
10 CSI = [csi_1, csi_2, ..., csi_{N_r}];  // CSI grouping
```
AoA of the direct path based on the same maximum likelihood method described in Eqn 4. The ground truth is calculated based on the physical locations of the AP and the Arduino YUN. In the experiment we ensure that all test locations are within the field view of the ULA, that favors the ULA performance. We plot the CDF of AoA estimation errors in Figure 10. The median error from the 9-antenna UCA is 2.6°, while that from the 3-antenna ULA array is 7.1°. The 90 percentile errors from the UCA and ULA are 5.7° and 17.1°, respectively. We see 3x improvement of 9-antenna UCA over 3-antenna ULA in AoA estimation accuracy.

**The field of view.** Our experiment demonstrates that UCA is able to have a 360° field of view. We put the AP with UCA at one location and the Arduino YUN at 76 different locations. The ground truth AoA of those 76 locations covers a range of [0°, 360°]. We estimate the AoA and plot the median AoA error at each location in Figure 11. We see that the AoA estimation is uniformly accurate across all the test directions with a maximum median error of 9.8°. The average error from all tests is 2.8°. The UCA built with SWAN provides 360 degree field of view with moderate accuracy.

**Localization.** We use two APs with UCA to estimate the AoA from the Arduino YUN. The direct path AoAs are then used to locate the Arduino YUN, in the same way as ArrayTrack [53]. We repeat the test with ULA at exact the same AP and Arduino YUN locations. Again we ensure that the Arduino YUN is in the field view of the ULA, which favours its performance. Figure 12 plots the results. The median error achieved is 0.45 m with the 9-antenna UCA and 1.43 m with the 3-antenna ULA, respectively. The 90 percentile error is 0.65 m with the UCA and 2.48 m with the ULA. We see 3.8x improvement in localization accuracy.

**Antenna number.** More antennas in the array, more accurate results we may expect from SWAN. The maximum number of antennas SWAN can enable is limited by the channel coherence time. When the channel is more stable, SWAN can keep switching across the antennas for every received
packet and collect more CSI from more antennas. Theoretically, the coherence time is the time interval during which the channel is considered not varying, which is usually approximated from the relative speed of the transceiver. If the Wi-Fi is transmitting at 600 Mbps, as shown in Figure 2, the expected transmission time of each packet is around 190 μs. With such an transmission rate, we derive the theoretical number of antennas that SWAN can enable for one sweeping. Figure 13 plots the results. We see that even when the transmitter moves at 7 m/s, SWAN is still able to provide us a virtual phased-array with 93 antennas.

We perform experiments to further study the impact of the mobility to our 9-antenna UCA. We let the Arduino YUN move away from the UCA with a constant speed and estimate the AoA estimation errors. We see that the error increases slowly with the increase of speed, i.e., from 2.56° at 1 m/s to 4.19° at 7 m/s. We do not observe severe performance degrade due to such level of movement.

4 CASE II: COMMUNICATION WITH ANTENNA DIVERSITY

SWAN increases the number of antennas that a commodity Wi-Fi AP can communicate with. Antennas in the extended array may experience entirely different channel conditions from each other. Wisely selecting the antenna combinations with the best channel conditions helps to maximize the spatial diversity of the AP and achieve the best communication throughput with its clients.

4.1 Communication with antenna diversity

Spatial diversity. We observe that the signal strengths of antennas in SWAN are highly heterogeneous. To demonstrate the heterogeneity, we divide an area of 1.6m × 2m into 20 40cm×40cm grids and put a Wi-Fi sender at each of the grids (30 in total) to transmit packets. We put a SWAN AP 10m away as the receiver and let it switch the receiving antennas. In Figure 14 (a), we plot the RSSIs from all 12 antennas with the sender at 30 locations. We calculate the maximum RSSI difference for each sender location by $RSSI_{\text{max}} - RSSI_{\text{min}}$, where $RSSI_{\text{max}}$ and $RSSI_{\text{min}}$ are the maximum and minimum RSSI obtained from the 12 antennas. Figure 14 (b) plots the results. From the two figures, we see high variation of RSSIs across antennas as well as across sender locations. In comparison, in Figure 14 (c), we also plot the RSSIs across packets on each of three antennas. We see minor variations of RSSIs on the same antenna within ±1 dB from the mean value. The above experiment results demonstrate the spatial diversity across antennas in SWAN. Selecting antennas with higher RSSI helps to achieve higher throughput. Frequency selective fading is another factor that significantly affects the channel condition beside signal strength [39]. Since the antennas in the array experience independent fading, selecting the antenna combination that results in lower selective fading can also improve the link throughput.

Antenna selection. To harvest the antenna diversity enabled by SWAN, we build an antenna selection module and embed it into the 802.11 network stack, which automatically selects the antenna combination for transmitting Wi-Fi packets. Specifically, the antenna selection module first measures and then compares the channel quality of all possible antenna combinations and then selects the one that provides the best channel. It is widely known that RSSI is not enough to quantify the quality of the wideband, frequency selective faded Wi-Fi channel [37, 48, 58]. CSI matrix fully characterizes the Wi-Fi channel but cannot provide quantitative comparison across channels [13]. Thus, we use the effective-SNR [13, 36] calculated with both RSSI and CSI as the metric to quantify the channel quality and select the antenna combinations with highest effective SNR to communicate with.

To measure the RSSI and CSI on each antenna, we adopt a simple antenna sweeping mechanism described in Script 2 to collect RSSIs and CSI for all 12 antennas in the extended array, which requires four consecutive packets. Wi-Fi NIC is able to derive both RSSI and CSI simultaneously from a received data packet. Therefore, the antenna selection module makes use of existing data packets if the AP is receiving from its clients (uplink traffic). Otherwise, the AP generates uplink traffic by explicitly requesting the client to send the probing packets to the AP. During the antenna sweeping, the AP as the receiver
does not switch its antenna until it successfully receives one packet with its current antenna combination. The client as the sender keeps track of successfully transmitted packets and stops when the required number of packets are delivered. Though the CSI is measured for the channel from the client to the AP, the selected antenna combination also works for the reverse channel because of channel reciprocity.

There are several other modules other than antenna selection module in the Wi-Fi network stack to select or adapt the communication parameters including the channel frequency, the channel bandwidth (20 or 40 MHz), the guard interval (400 or 800 ns) and the data rate. The modules adapt their parameters in parallel and at different time scale. The channel frequency and bandwidth selection changes only when the current channel becomes overcrowded. The antenna selection module changes the antenna combination when the channel coefficients change. Therefore, we keep tracking the channel variation and restart the antenna sweeping and re-select the best antenna combinations, when the channel varies. The data rate is varied all the time even when the channel is stable as it takes time for the data rate to converge to the optimal. All the modules operate independently. A joint rate antenna selection algorithm may further improve the communication throughput. Developing the optimal configuration selection algorithm is left as our future work.

Channel variations. To track channel variations, SWAN keeps a record of the most recent received CSI $csi_i$ over the previous antenna combination and calculates the difference with the new CSI measurement $csi_n$ by $\Delta csi = csi_n - csi_i$. Theoretically, the difference $\Delta csi$ should be minor for static channel. But the time varying phase and amplitude errors due to carrier frequency offset (CFO), symbol timing offset (STO), and sampling frequency offset (SFO) [50, 54] make the CSI vary even when the channel remains unchanged. Figure 15 presents the measured CSI difference between consecutive packets. The phase and amplitude differences are linear for static channel. Specifically, the amplitude difference is flat with a fitted line of zero slope. The phase difference is fitted to a sloped line due to the frequency dependent errors introduced by STO and SFO [50, 54]. On the other hand, when the channel varies, the linearity of phase and amplitude differences disappears, as suggested in Figure 15.

Above observations allow us to use linear fitting to track the channel variance. We thus measure the linearity of both phase and amplitude of $csi$. Specifically, we do the linear curve fitting, and calculate the goodness of our fit. In statistics, $R^2$ is used to measure goodness of a fit, which is calculated as:

$$R^2 = \frac{\sum_{i=1}^{N} (y_i - \bar{y})^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}$$

where $N$, $y_i$, $\hat{y}_i$ and $\bar{y}$ are the number of data points, the $i$th data points, the $i$th fitted data and the mean of all data, respectively. We calculate the $R_p^2$ and $R_a^2$ of the phase and amplitude of $\hat{csi}$, separately and derive the averaged $R_{csi}^2$:

$$R_{csi}^2 = \frac{R_p^2 + e^{-\|\epsilon\|}R_a^2}{2}$$

where $\epsilon$ is the slope of the fitted curve of $csi$ amplitude, which captures the flatness of the amplitude curve. According to Eqn 6, $R_{csi}^2$ falls into the range of $[0, 1]$ and we treat the channel with $R_{csi}^2 > 0.9$ as static and otherwise varied. For robustness, the average $R$ factors of three most recent three packets is used. When CSI is not available, RSSI is used for tracking channel variations. From Figure 14 (c), we see that RSSI variations is less than 1 dB for static channels and we thus identify the channel to be varying when the difference of RSSI relative to the mean is greater than 2 dB for any two of the three antennas, and for three consecutive packets.

### 4.2 Evaluation

We evaluate the throughput with the extended antenna array provided by SWAN. We let one WPJ558 Wi-Fi router work in AP mode. One Arduino YUN and another WPJ558 work in client mode and connect to the AP. Arduino YUN has one antenna and the WPJ558 has three antennas, which are used

---

**Script 2: Antenna Sweeping**

**Input:** No. of antennas $N_t$ attached to each chain

1. $\text{ant-comb}_1[3]=\{0\}$
2. for $i = 1; i \leq N_t$, do
3. for $j = 0; j \leq 3; j \neq m$ do
4. \hspace{1cm} $\text{ant-comb}_1[j] = i$
5. \hspace{1cm} $\text{config}_\text{rx}_\text{ant}(\text{ant-comb}_1[3])$;
6. \hspace{1cm} $\{\text{pkt}_i, \text{csi}_i, \text{ant-comb}_1\} = \text{recv}_\text{pkt}()$
throughput across all 64 fixed antenna combinations in SWAN, and find out the combination that maximizes the averaged throughput over the 400 locations. We plot the results from the three settings in Figure 16. We see that SWAN with our antenna selection mechanism achieves a 70% quantile of 68 Mbps, which is 11 Mbps and 21 Mbps higher than fixed antenna combination that can achieve maximum average throughput and the original Wi-Fi antennas. We repeat the above experiment with WPJ558 which supports MIMO. The results are plotted in Figure 17. We see similar fact that SWAN with our antenna selection mechanism achieves a 70% quantile of 195 Mbps, which is 32 Mbps and 48 Mbps higher than systems using the fixed antenna combinations and original three fixed Wi-Fi antennas, respectively.

The experiments show that SWAN improves the Wi-Fi system throughput in two aspects. First, SWAN provides higher antenna diversity compared with all existing Wi-Fi APs with only three fixed antennas. The best antenna combination provides more than 10 Mbps throughput gain on top of Wi-Fi. But a fixed antenna combination cannot fit all channel conditions. Therefore, the second improvement comes from the auto antenna selection according to the channel measurement. By adjusting the antenna configurations across locations SWAN can achieve additional 11 Mbps and 32 Mpbs throughput gains over the fixed combinations, for SISO and MIMO link, respectively. We see SWAN achieves over 30% throughput improvement on both the SISO and MIMO settings over the original Wi-Fi antennas.

**Antenna selection.** We evaluate the performance of our antenna selection mechanism. We use all 64 antenna for communication at each of the 400 locations, which gives us the best antenna combination for each location. Each antenna combination is represented as a unique number in the range of \([1, 64]\). In Figure 18 (a), we compare the best antenna combination with that selected by our auto antenna selection mechanism, and plot the difference. Zero difference means...
the auto selected combination gives the best. In experiment, our mechanism picks the best at 365 out of all 400 locations. Even for those not picking the best antenna combination, our pick still gives very high throughput compared with the best. To illustrate, in Figure 18 (b), we plot the measured throughput of all 64 antenna combinations at one location. The highest throughput is achieved with 17th combinations. Our mechanism selects 29th combination, which achieves a throughput very close to the best.

**Mobile channels.** We let the client move along a pre-defined trajectory. We divide the trajectory into 1m segments and measure the average throughput achieved for each segment. We plot the throughput of SISO link in Figure 19, and that of MIMO link in Figure 20. Similar to the static experiments, we also test the throughput from the best fixed antenna combination and that from the original Wi-Fi. We see that SWAN with auto antenna selection outperforms any fixed combination. SWAN achieves average throughput of 53.3 Mbps, which is 7.4 Mbps and 14.8 Mbps higher than the best fixed combinations and that of the original Wi-Fi in SISO case. In MIMO case, SWAN with auto antenna selection achieves average throughput of 153.9 Mbps, which is 23.0 Mbps and 34.4 Mbps higher than the other two. The experiment results suggest that our mechanism is able to timely detect the channel variations in mobility, and switch the antennas in order to achieve higher throughput. Similar to the static experiments, SWAN provides gains in two-folds – the antenna diversity from extended array and channel adaptation from automatic antenna selection module.

## 5 SWAN IN SCALE

Previous discussions and experiments are based on our 12-antenna prototype setting. SWAN can easily scale to a larger array of antennas. The current design of SWAN is based on a single tier of RF switches, e.g., three SP4T for 12 antennas, SP8T for 24 antennas, or SP12T for 36 antennas. Through cascaded connection, we can extend COTS RF switches to more throws. Figure 21 gives an example where we connect five SP4T in a cascaded way to form a 16 throws (SP16T) switch. A single MCU, e.g., an Arduino board, coordinates all SP4T switches and builds a SP16T transparent to the AP. Using three such SP16T, we can get a 48 antenna array. The control exchange of SWAN can also scale. A key factor that limits the number of antennas that SWAN can support is the storage required to store the template TCP packets for signaling different antenna combinations. The size of each template sk_buff is 274 bytes. For an antenna array of 192 antennas (i.e., 64 to each radio chain), the RAM space required is 71.8Mb, which is affordable on normal Wi-Fi routers, e.g., both WDR4300 and WPJ558 have 128Mb RAM. To further scale beyond that, SWAN assembles ad hoc control packets. SWAN employs a general packet template with intact IP and TCP headers but empty payload. When assembling the ad hoc control packet, SWAN duplicates the empty TCP template and insert the antenna combination descriptor ant-comb to the payload. The control packet is then sent to the Arduino, which however does not need to decode the packet so not verify the CRC (which does not likely match the control payload) in ag71xx_rx_packet() in the kernel. By doing this, SWAN only needs minimum RAM space to save the empty TCP template, with a trade-off in slight increase of its tAP time and thus slight increase of its chance in antenna mis-configurations (which however can be detected and annotated in most cases by SWAN).

## 6 RELATED WORK

**Antenna extension on commodity Wi-Fi.** Phaser [11] enables a large phased-array on commodity by combing
the channel measurement (CSI) collected by multiple Wi-Fi NICs. Phaser has more radio chains than antennas since one radio chain from each of those NICs is connected with each other for synchronization purpose and thus wasted, which results in significant hardware costs. The multiple NICs used by Phaser are on different Wi-Fi APs so that they do not work cooperatively as a cohort to provide general Wi-Fi communication services. Prior works [1, 27, 38] also emulate large antenna arrays using Synthetic Aperture Radar (SAR). SAR however, requires physical movement of the antennas, incurring huge delay [2], and thus cannot meet the $\mu$s level switching speed requirement. AmorFi [24] connects multiple radio front ends with multiple APs together via optical fiber and then selects the radio front ends for each AP. AmorFi requires dedicated and costly hardware and cannot be extended to other applications such as AoA estimation.

Many antenna system. Many efforts has been made to build many antenna systems for distributed MIMO [14, 25], multi-user beamforming [4, 10, 16, 42, 43, 45, 55, 56], and full duplex communication system [5, 40, 44], based on software-define-radio (SDR) platform. All those systems enables tens of radio chains by connecting multiple synchronized SDR boards together to work as a single AP. The channel capacity increases and the total user the system can simultaneously communicate with also increases, proportionally with the radio chains. The number of radio chain on a single piece of SDR is, however, still limited, e.g., four for a single WRAP board. Therefore, multiple pieces of SDRs are required to enable many antennas, which results in tremendous hardware cost and makes those techniques impractical to be directly implemented on commodity Wi-Fi systems.

Phased-array. Phased-array is widely used in many different systems as a sensing interface to estimate AoA or AoD [3, 29, 31]. Recent advances [11] have built phased-array on commodity Wi-Fi with three antennas and enables applications such as indoor localization [11, 26, 27, 53], passive tracking [23, 51], security [52], Wi-Fi imaging [15] and activity recognition [1, 38] with the estimated AoA or AoD. SWAN enables a new opportunity to greatly increase the antenna number in the phased-array and thus provide angle estimation with much finer resolution and higher accuracy.

Antenna diversity. Equipping MIMO systems with more antennas than radio chains and selecting the optimal antenna group to perform the MIMO communication can improve the capacity and reliability of MIMO communications, with significantly reduced hardware complexity and cost [6, 12, 34, 35, 41]. SWAN is the first system that realizes similar gain on commodity Wi-Fi systems, with low cost COTS hardware components. Theoretically, spatial shift keying (SSK) [9, 19, 21, 46] and spatial modulation [20, 32, 57] can make use of the antenna diversity and achieve multi-folds throughput improvement, which however requires modifications to the encoding and decoding modules of Wi-Fi PHY. Hybrid beamforming [8, 33, 49] harvests the spatial diversity with analog beam steering using phase shifter and share similar architecture with SWAN. Existing hybrid beamforming systems are based on software-define-radio platforms and are impossible to be directly applied to commodity Wi-Fi devices without addressing similar challenges that SWAN has met including the fast control exchanging and reliable annotation. Most existing efforts including hybrid beamforming do not support antenna switching and thus cannot harness the spatial diversity for wireless sensing applications.

7 CONCLUSION

This paper introduces our design and implementation experience of SWAN. SWAN serves as a general, plug-and-play antenna extension solution to commodity Wi-Fi devices. SWAN enables $\mu$s level control between the AP and the Arduino MCU. SWAN provides easy-to-use user interface for building diverse applications. Our experimental study shows that SWAN provides fast and reliable antenna configuration that helps to significantly improve the performance of RF sensing and communication applications.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers and shepherd for their valuable comments and helpful suggestions that improve the quality of this paper. The authors would also like to thank Fan Yi and Agustinus Wellson Tengourtius for helping with the experimentation. The work is supported by the Singapore MOE Tier 1 grant RG125/17, Tier 2 grant MOE2016-T2-2-023, and NTU CoE grant M4081879.
REFERENCES


[50] Yaxiong Xie, Zhenjiang Li, and Mo Li. 2015. Precise Power Delay Profiling with Commodity WiFi. In *ACM MobiCom*.


