From CSI Tool to PicoScenes Wi-Fi Sensing Middleware

Supercharging Your Next Wi-Fi Sensing Research

Zhiping Jiang, Jun Luo, Zhe Chen
School of Computer Science and Technology
Xidian University, Xi'an, China
Nanyang Technological University, Singapore
AIWISE, Guangzhou, China

https://ps.zpj.io

Dr. Jiang Zhiping (蒋志平)

A lecturer with the School of

Computer Science and Technology,

Xidian University, Xi’an, China.

The founder and core developer of

PicoScenes Wi-Fi sensing middleware.

In the next 40 minutes …

- A recap of the current status of Wi-Fi Sensing research (5)
- A brief introduction to PicoScenes (15)
- Several demo videos (10)
- How can researchers benefit from PicoScenes? (10)
- Q & A

Is Wi-Fi sensing still an active research field?

Is Wi-Fi sensing still an active research field?

Seems not …

- Researchers shift to new directions, acoustic, mmWave, LoRa, ZigBee, RFID, CTC, *etc*.

- Methodology:
  - The model-based papers are extremely hard to get in.
  - The learning-based approach thrives, but it is not the silver bullet.

Why ?

巧妇难为无米之炊

The cleverest housewife cannot cook without rice ! 😁

Three Barriers in the Wi-Fi sensing research

- Unknown baseband design & its influence on CSI
- Incapable hardware
- Incapable software

Three Barriers in the Wi-Fi sensing research: incapable hardware

Out-of-date

- IWL5300/QCA9300/ESP32 based on 802.11n, decade ago;
- BCM43xx, 11ac, near a decade
- ...

No low-level controls

- AGC, calibration
- I/Q, PA,
- timing, clocking
- ...

No more PHY layer output

- baseband signals
- OFDM symbols
- CSI by L-LTF
- ...

No advanced features

- more radio chains
- external clock
- full-duplex
- ....

SDR ? *USRP/ WARP/ RTL-SDR/ LimeSDR ?*

Three Barriers in the Wi-Fi sensing research: incapable hardware

Ideally, SDR is thought to be the BFG for Wi-Fi sensing…

Actually, SDR is rarely used in the Wi-Fi sensing research...

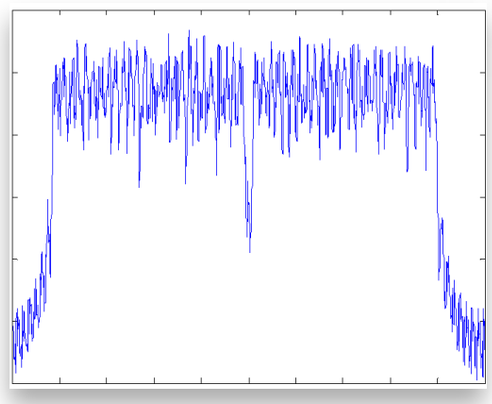
Three Barriers in the Wi-Fi sensing research: incapable hardware

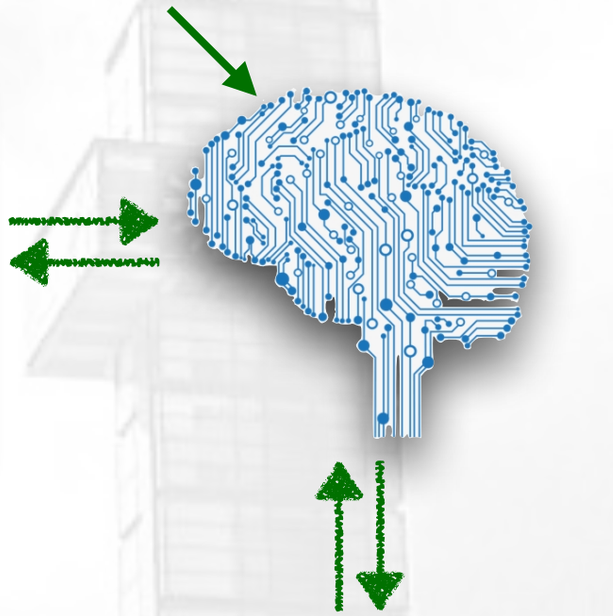
COTS NIC too good to abandon?

or, SDR hardware too expensive?

USRP B210, $ 1,472.00 USD

The missing Wi-Fi baseband DSP for SDR





01010010000100 ...

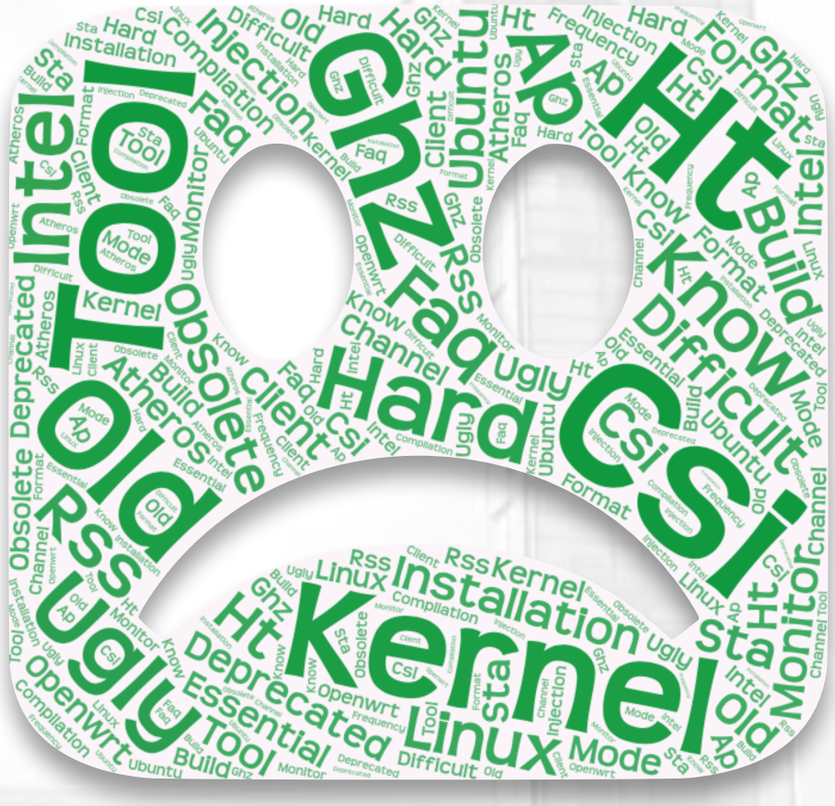
There is ***no***
publicly available Wi-Fi
baseband implementations.

https://ps.zpj.io

Three Barriers in the Wi-Fi sensing research: incapable software

After 10 years, the software is still very primitive ...

- No multi-NIC control
- No integrated Tx/Rx
- No MPDU payload
- No hardware control
- Deprecated kernel/OS
- No collaborative measurement
- hard to use, far from maturity
- No technical support



Introducing PicoScenes

Wi-Fi Sensing Middleware

https://ps.zpj.io

What is PicoScenes?

- Still a CSI tool, but a powerful, modern, easy-to-use, and well-documented one, trying to overcome the hardware and software barriers of Wi-Fi sensing research.
- First and temporarily the only Wi-Fi 6E-ready CSI tool
- A flexible Wi-Fi sensing middleware supporting plugin mechanism
- Being developed for 5 years, released in Apr. 2021, being used by 58 university / research institutes, 31 in China and 27 oversea
- Warmly appreciated by users*
  - "*Thanks again for this software, and I'm looking forward to seeing it develop even further!*" — Gleen Forbes (Robert Gordon Univ., UK)
  - "*Thanks for providing an awesome tool*" — @wonwon
  - "*Thanks so much for your amazing work putting together PicoScenes! I just discovered it the other week but looks to be a really cool piece of kit, huge congratulations!*" — James Bache (Univ. of Exeter, UK)

* *Comments are excerpted from PicoScenes Issue Tracker.*

Visit here!

https://ps.zpj.io

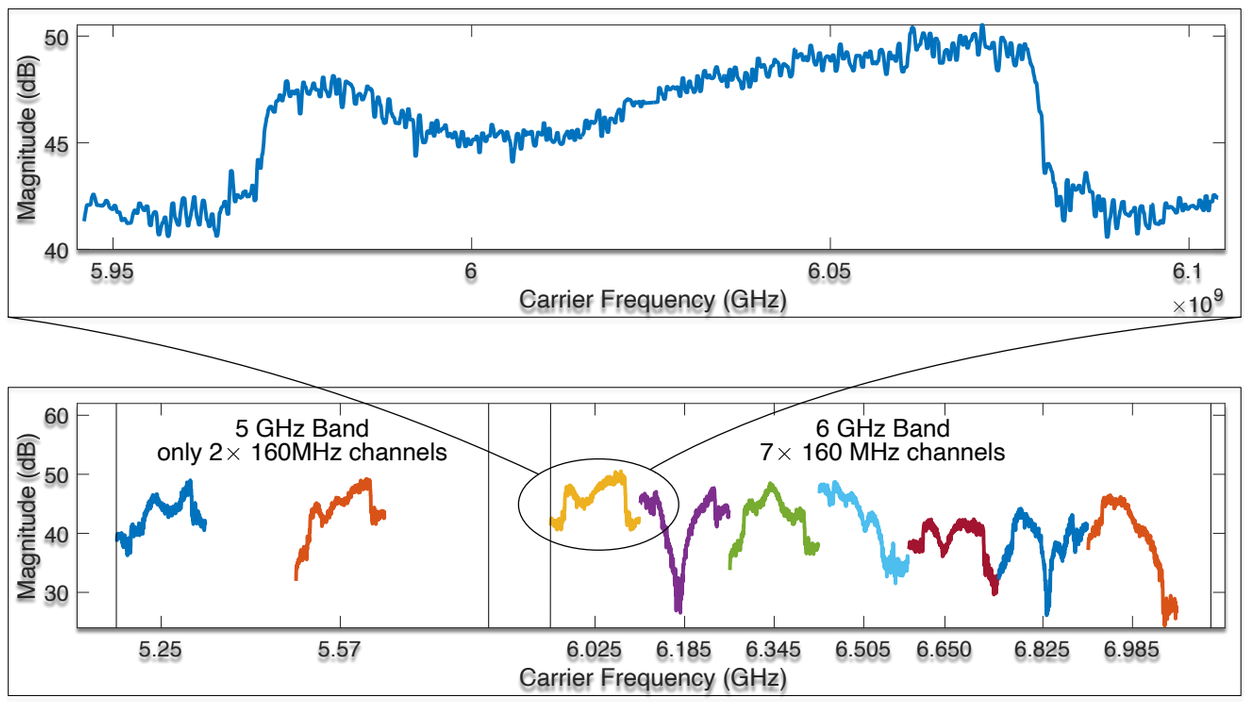
What is PicoScenes?

An non-exhaustive list of highlights of PicoScenes…

In the hardware aspect…

- Supports the most CSI-extractable hardware
  - Intel AX200 series (AX200/210/211)
  - Intel 5300 NIC (IWL5300)
  - Qualcomm Atheros 9300 (QCA9300)
  - *more* to support …
- Supports SDR into Wi-Fi sensing frontend
  - USRP series
  - HackRF One (just over 100$!)
  - *more* to support …
- The first and only Wi-Fi 6E-ready CSI Tool
  - using AX210/211 NIC
  - 802.11ax + 160-MHz BW + 6-GHz band
  - 7 continuous 160-MHz channels (5945-7125 MHz)
  - up to 8192 subcarriers per packet

Supported HW, including 5 Wi-Fi NICs and SDRs



Wi-Fi 6E CSI measurement by AX210

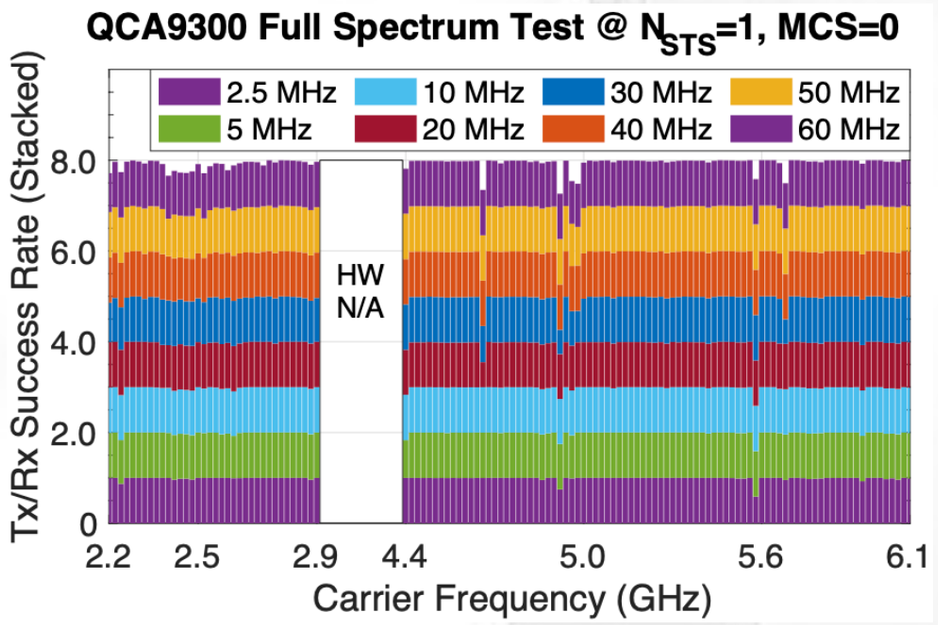
https://ps.zpj.io

What is PicoScenes?

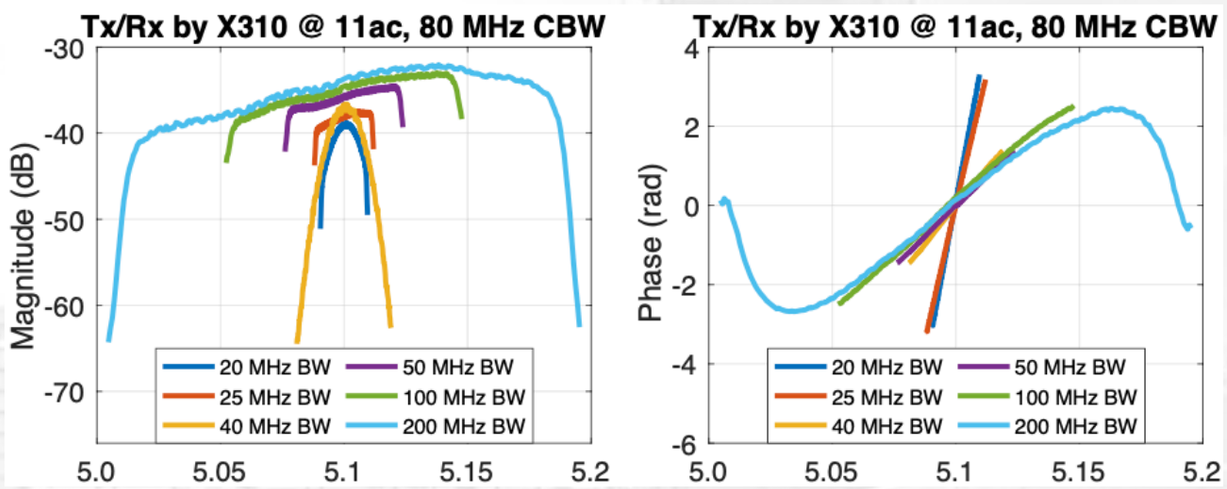
An non-exhaustive list of highlights of PicoScenes…

In the hardware aspect…

- Supports the most CSI-extractable hardware
  - Intel AX200 series (AX200/210/211)
  - Intel 5300 NIC (IWL5300)
  - Qualcomm Atheros 9300 (QCA9300)
  - *more* to support …
- Supports SDR into Wi-Fi sensing frontend
  - USRP series
  - HackRF One (just over 100$!)
  - *more* to support …
- The first and only Wi-Fi 6E-ready CSI Tool
  - using AX210/211 NIC
  - 802.11ax + 160-MHz BW + 6-GHz band
  - 7 continuous 160-MHz channels (5945-7125 MHz)
  - up to 8192 subcarriers per packet
- Freely turning RF in 2.4-GHz wide spectrum
  - using QCA9300 NIC
  - Tunable in 2.2-2.9 and 4.4-6.1 GHz ranges
- Up to 200-MHz BW CSI measurement
  - using USRP X310 SDR
  - live packet injection & CSI measurement



QCA9300 supports 2.4-GHz spectrum w/ variable BW

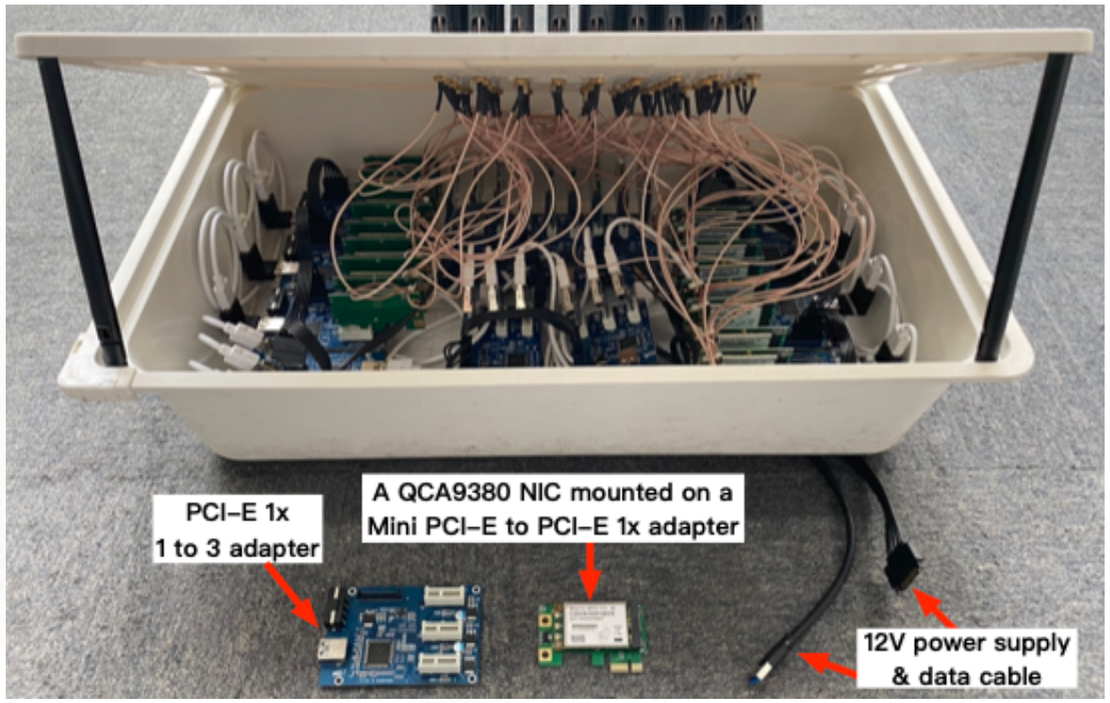


20-200 MHz BW CSI measurement by X310

https://ps.zpj.io

What is PicoScenes?

An non-exhaustive list of highlights of PicoScenes…



PicoScenes drives a 27-NIC array

SDR Rx Rate vs. N_{STS} vs. N_{Ant} @ BW=20 MHz, 500 Hz Tx by QCA9300

MCS	T1R1	T1R2	T2R2	T1R3	T2R3	T3R3	T1R4	T2R4	T3R4
0	1	1	1	0.914	0.905	0.888	0.75	0.76	0.674
1	1	1	1	0.889	0.912	0.893	0.759	0.716	0.759
2	1	1	1	0.943	0.948	0.847	0.752	0.768	0.749
3	1	1	1	0.957	0.92	0.938	0.778	0.769	0.745
4	1	1	1	0.977	0.95	0.889	0.791	0.753	0.71
5	1	1	1	0.876	0.869	0.818	0.742	0.689	0.664
6	1	1	1	0.83	0.871	0.796	0.706	0.716	0.651
7	1	1	1	0.878	0.811	0.818	0.728	0.676	0.688

Tx N_{STS}(T) and Rx N_{Ant}(R) combination

High-perf. Wi-Fi baseband codec. w/ MIMO support

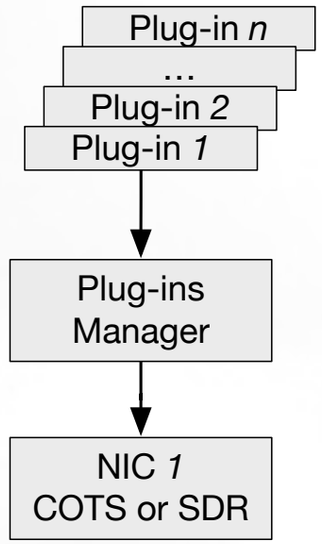
In the software aspect…

- Supports multi-frontend CSI measurement
  - concurrent CSI measurement from heterogeneous HW
  - Intuitive device naming
- Wi-Fi baseband implementation
  - all-format (a/g/n/ac/ax), up to 4x4 MIMO
  - Live packet capture & CSI measurement
  - Signal recording & replay
- Packet Injection for all supported frontend
  - Supports both Wi-Fi NICs and SDR

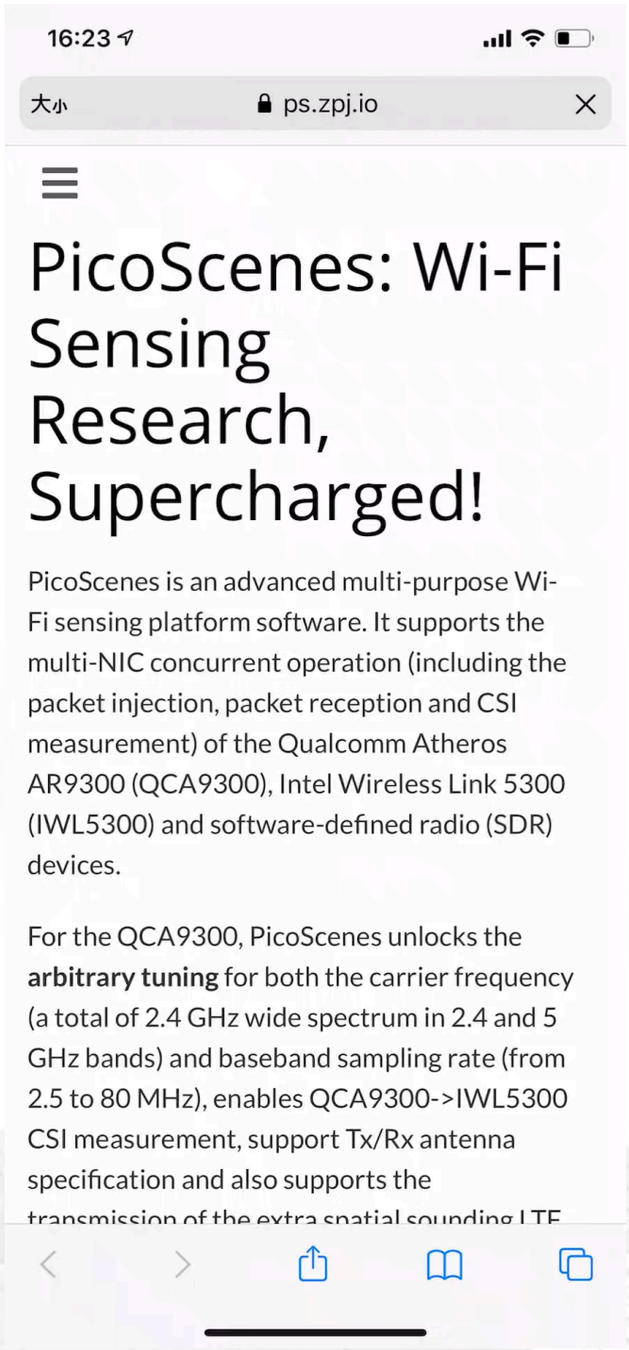
https://ps.zpj.io

What is PicoScenes?

An non-exhaustive list of highlights of PicoScenes…



PicoScenes supports multi-frontend, and each runs its own plugins



Rich documentation

In the software aspect…

- Supports multi-frontend CSI measurement
  - concurrent CSI measurement from heterogeneous HW
  - Intuitive device naming
- Wi-Fi baseband implementation
  - all-format (a/g/n/ac/ax), up to 4x4 MIMO
  - Live packet capture & CSI measurement
  - Signal recording & replay
- Packet Injection for all supported frontend
  - Supports both Wi-Fi NICs and SDR
- Plugin mechanism
  - Driver + Platform + Plugins architecture
  - plugins executes all the end-use functionalities
  - Rapid development for complex & realtime measurement
- Provide official MATLAB/Python Toolbox
- Much easier to use
  - *apt*-based installation & upgrade on Ubuntu 20.04
  - Fresh new installation in 10 mins
  - The richest documentation

https://ps.zpj.io

What is PicoScenes?

An non-exhaustive list of highlights of PicoScenes…

In the hardware aspect…

- Supports the most CSI-extractable hardware
  - Intel AX200 series (AX200/210/211)
  - Intel 5300 NIC (IWL5300)
  - Qualcomm Atheros 9300 (QCA9300)
  - *more* to support …
- Supports SDR into Wi-Fi sensing frontend
  - USRP series
  - HackRF One (just over 100$!)
  - *more* to support …
- The first and only Wi-Fi 6E-ready CSI Tool
  - using AX210/211 NIC
  - 802.11ax + 160-MHz BW + 6-GHz band
  - 7 continuous 160-MHz channels (5945-7125 MHz)
  - up to 8192 subcarriers per packet
- Freely turning RF in 2.3-GHz wide spectrum
  - using QCA9300 NIC
  - Tunable in 2.2-2.9 and 4.4-6.1 GHz ranges
- Up to 200-MHz BW CSI measurement
  - using USRP X310 SDR
  - live packet injection & CSI measurement

In the software aspect…

- Supports multi-frontend CSI measurement
  - concurrent CSI measurement from heterogeneous HW
  - Intuitive device naming
- Wi-Fi baseband implementation
  - all-format (a/g/n/ac/ax), up to 4x4 MIMO
  - Live packet capture & CSI measurement
  - Signal recording & replay
- Packet Injection for all supported frontend
  - Supports both Wi-Fi NICs and SDR
- Plugin mechanism
  - Driver + Platform + Plugins architecture
  - plugins executes all the end-use functionalities
  - Rapid development for complex & realtime measurement
- Provide official MATLAB/Python Toolbox
- Much easier to use
  - *apt*-based installation & upgrade on Ubuntu 20.04
  - Fresh new installation in 10 mins
  - The richest documentation

https://ps.zpj.io

Demo Videos

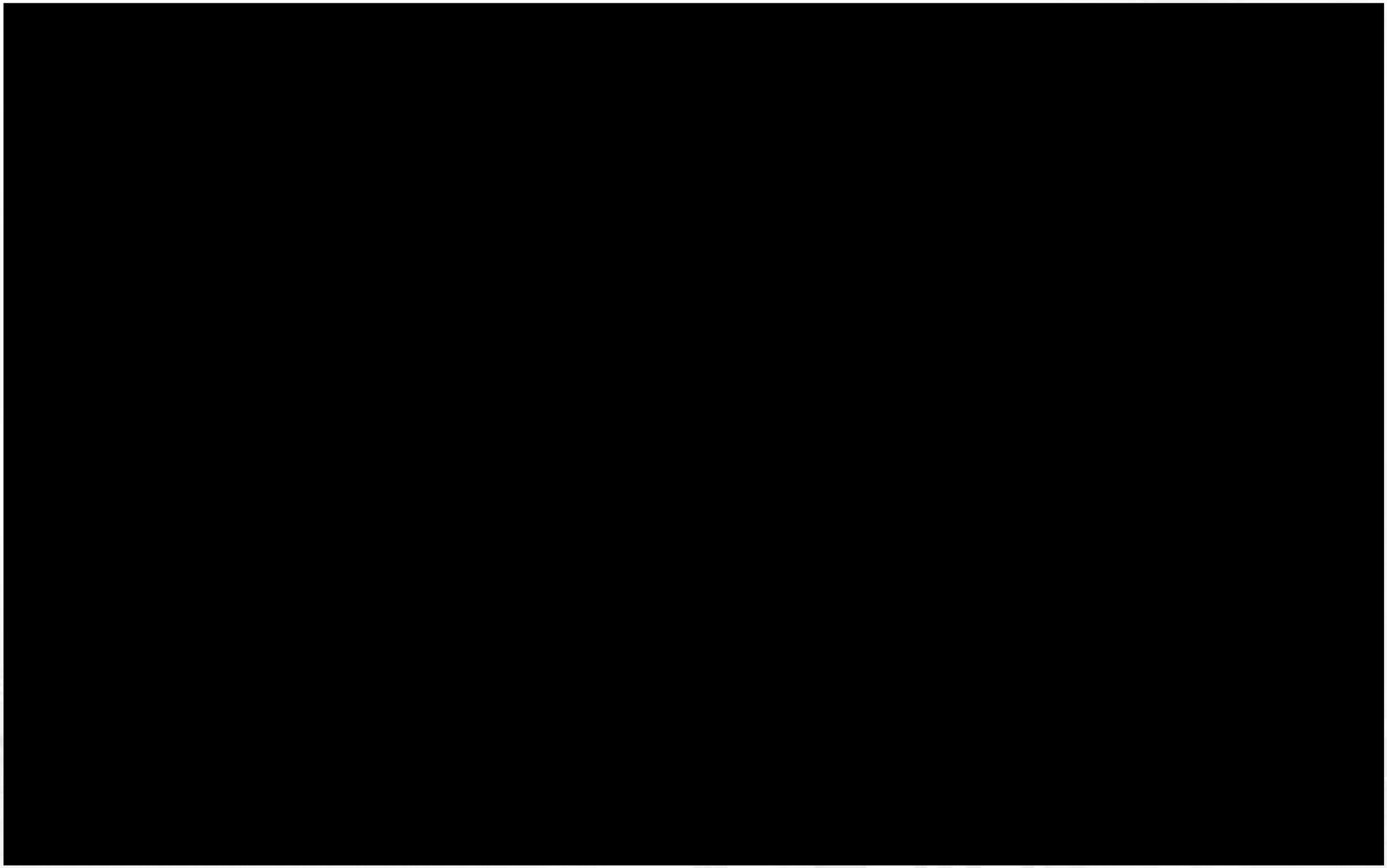
- CSI measurement between AP and IWL5300/AX200/210 NICs
- Fully passive CSI measurement by AX200/AX210 NIC
- Packet Injection & CSI measurement between two Wi-Fi NICs
- Packet Injection & CSI measurement between two SDRs
- 6-GHz spectrum scanning with two AX210 NICs
- Installation & Use of PicoScenes MATLAB Toolbox

Demo 1: CSI Measurement between AP and AX210 NIC

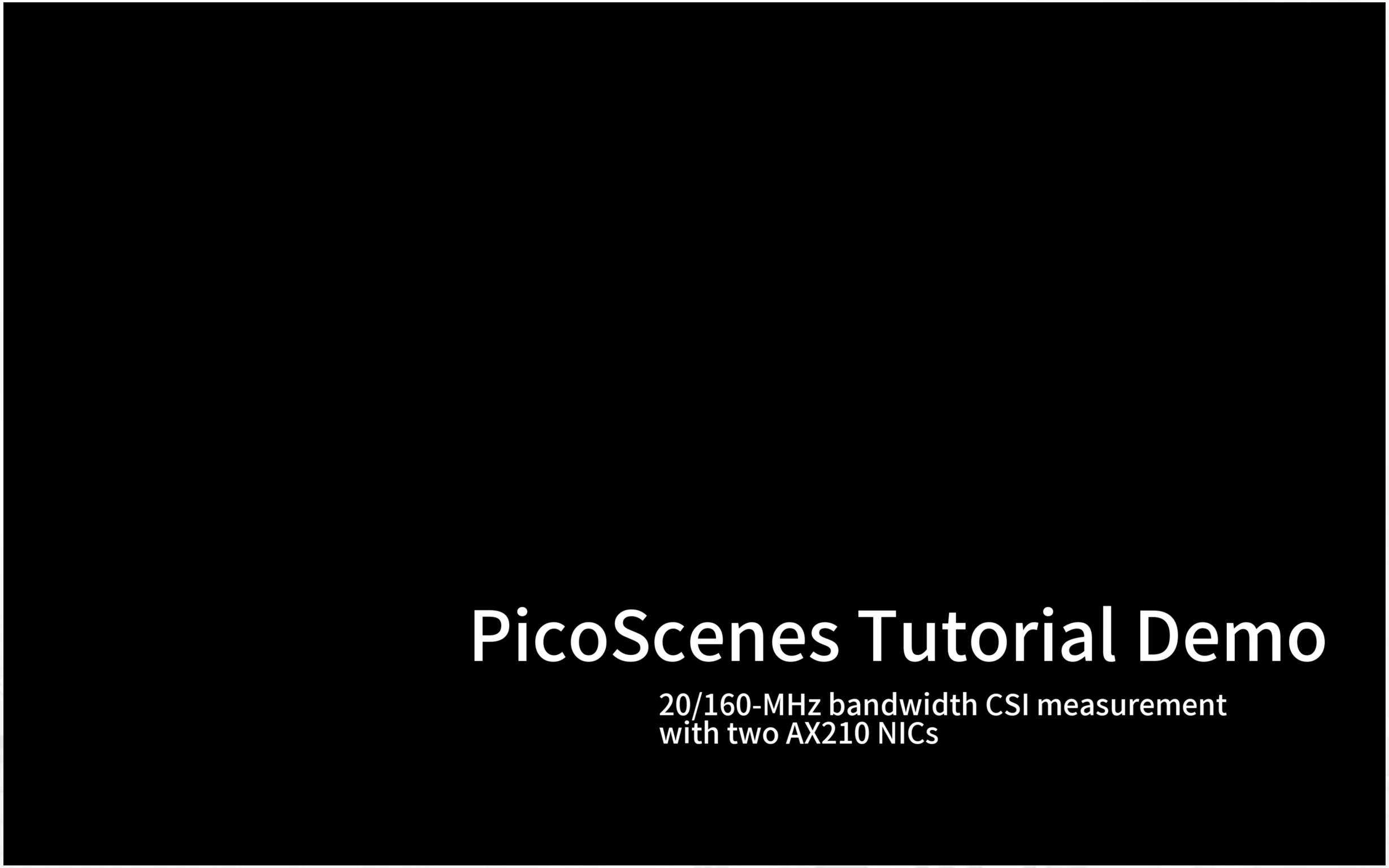
PicoScenes Tutorial Demo

CSI measurement between AP and AX210 NIC

Demo 2: Fully Passive CSI Measurement by AX210 NIC

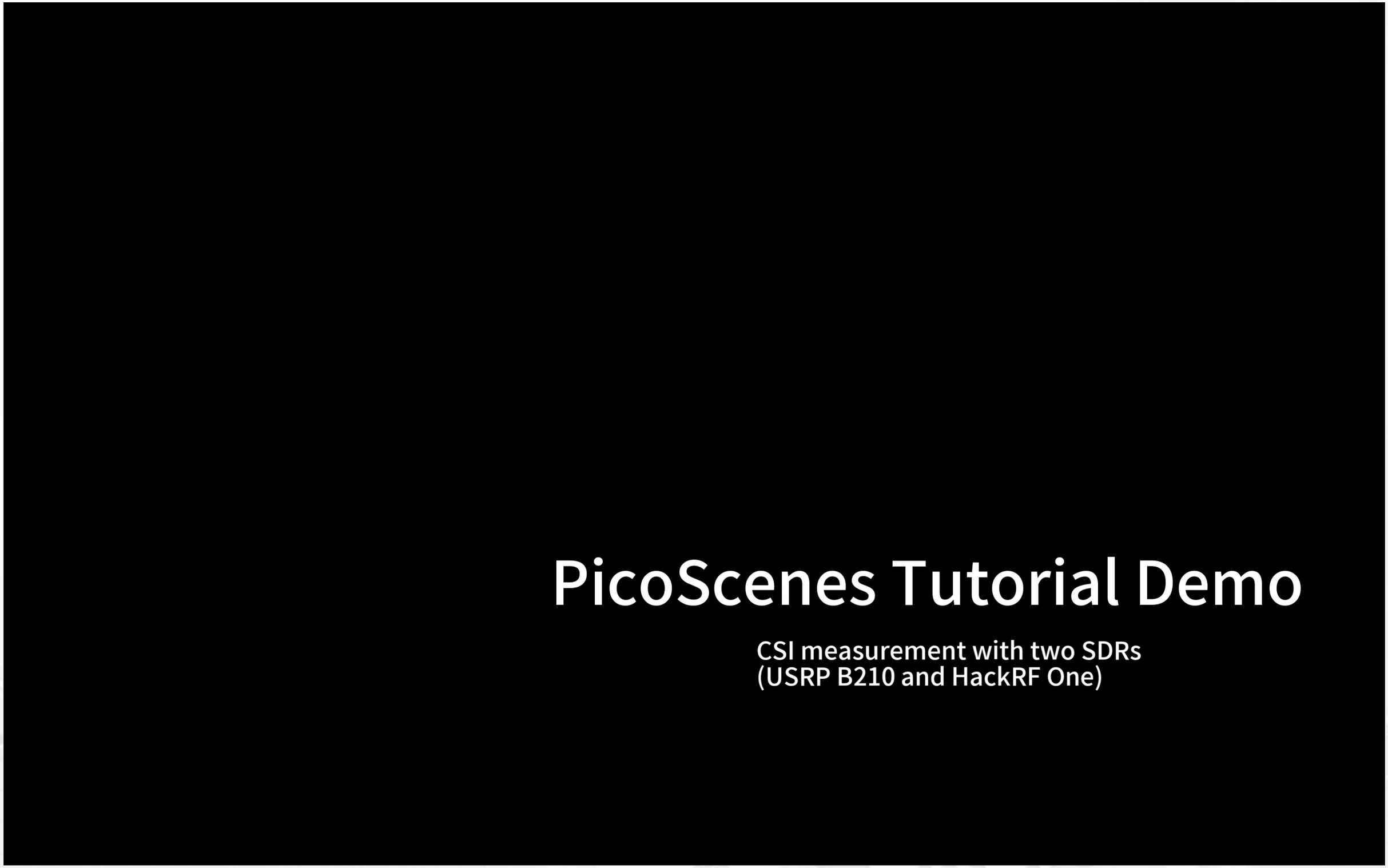


Demo 3: 20/160-MHz BW CSI Measurement between two AX210 NICs



https://ps.zpj.io

Demo 4: CSI Measurement between two SDRs (USRP B210 and HackRF One)

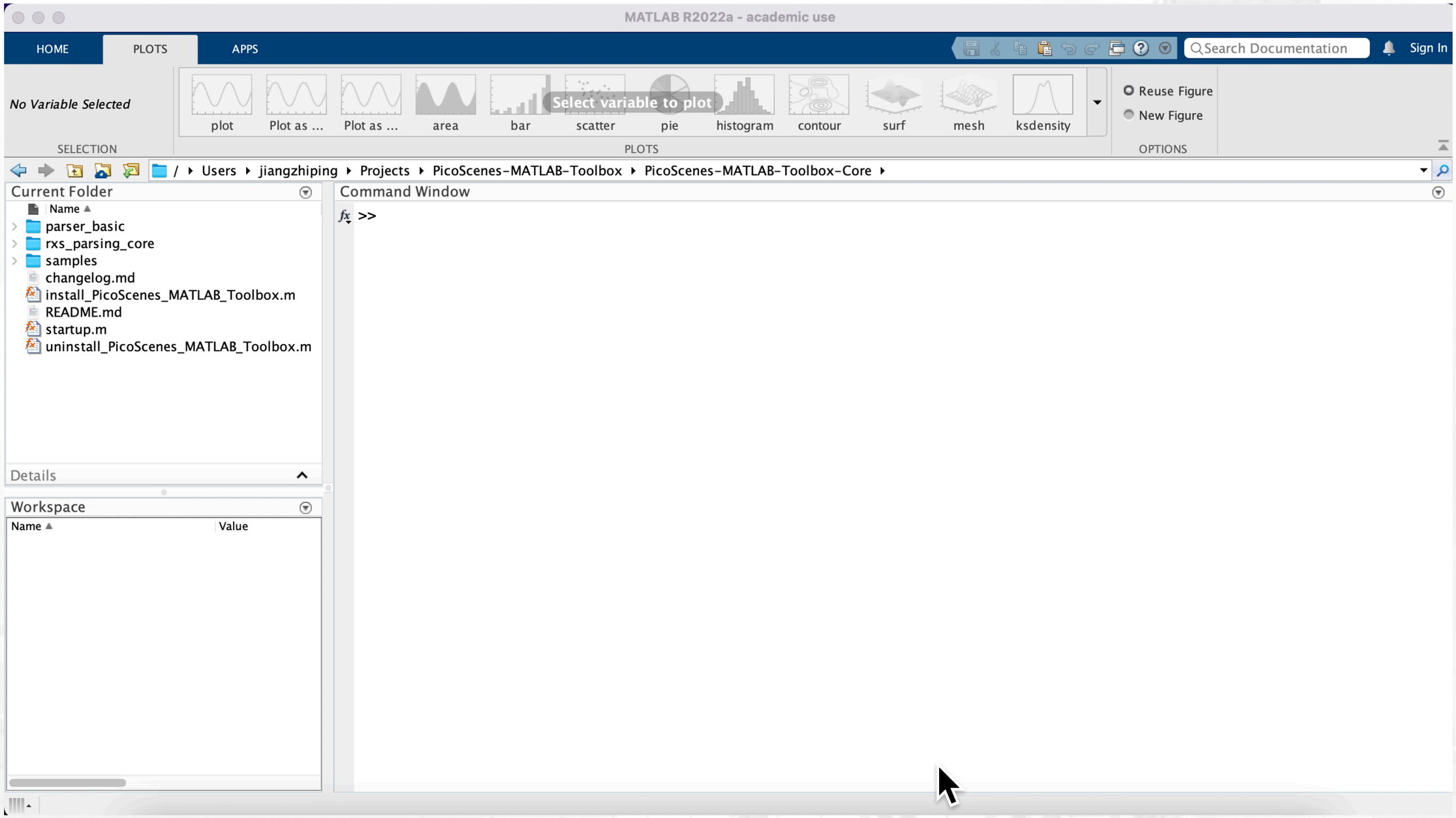


Demo 5: CSI Measurement between two SDRs (USRP B210 and HackRF One)

Demo 6: CSI measurement while freq-hopping in 6-GHz band using two AX210

Demo 7: Installation & Use
PicoScenes MATLAB Toolbox

PicoScenes MATLAB Toolbox (PMT) provides consistent data parsing and drag'n'drop style .csi file decoding.



https://ps.zpj.io

Powerful & Agile CSI Measurement via PicoScenes Plugin Mechanism

https://ps.zpj.io

Why plugins?

Without plugin architecture,

event the simplest CSI measurement tool is a hell,

for researchers!

Why plugins?

Taking *random_packets*, from Intel 5300 CSI Tool, as an example …

random_packets:
transmit 802.11n-format packets to the special 00:16:ea:12:34:56 MAC address

- Total of 190 LoC, 6 sections:
  1. headers & constants (20 LoC)
  2. packet data structure definition (10 LoC)
  3. Program option handling (30 LoC)
  4. Packet generation (70 LoC)
  5. Initialize packet injection library (20 LoC)
  6. Transmit packets (40 LoC)

2

```c
struct lorcon_packet
{
        __le16  fc;
        __le16  dur;
        u_char  addr1[6];
        u_char  addr2[6];
        u_char  addr3[6];
        __le16  seq;
        u_char  payload[0];
} __attribute__ ((packed));
```

3

```c
uint32_t num_packets;
uint32_t packet_size;
struct lorcon_packet *packet;
uint32_t i;
int32_t ret;
uint32_t mode;
uint32_t delay_us;
struct timespec start, now;
int32_t diff;

/* Parse arguments */
if (argc > 5) {
        printf("Usage: random_packets <number> <length> <mode: 0=my MAC, 1=injection MAC> <delay in us>\n");
        return 1;
}
if (argc < 5 || (1 != sscanf(argv[4], "%u", &delay_us))) {
        delay_us = 0;
}
if (argc < 4 || (1 != sscanf(argv[3], "%u", &mode))) {
        mode = 0;
        printf("Usage: random_packets <number> <length> <mode: 0=my MAC, 1=injection MAC> <delay in us>\n");
} else if (mode > 1) {
        printf("Usage: random_packets <number> <length> <mode: 0=my MAC, 1=injection MAC> <delay in us>\n");
        return 1;
}
if (argc < 3 || (1 != sscanf(argv[2], "%u", &packet_size)))
        packet_size = 2200;
if (argc < 2 || (1 != sscanf(argv[1], "%u", &num_packets)))
        num_packets = 10000;
```

4

```c
/* Generate packet payloads */
printf("Generating packet payloads \n");
payload_buffer = malloc(PAYLOAD_SIZE);
if (payload_buffer == NULL) {
        perror("malloc payload buffer");
        exit(1);
}
generate_payloads(payload_buffer, PAYLOAD_SIZE);

/* Setup the interface for lorcon */
printf("Initializing LORCON\n");
init_lorcon();

/* Allocate packet */
packet = malloc(sizeof(*packet) + packet_size);
if (!packet) {
        perror("malloc packet");
        exit(1);
}
packet->fc = (0x08 /* Data frame */
                | (0x0 << 8) /* Not To-DS */);
packet->dur = 0xffff;
if (mode == 0) {
        memcpy(packet->addr1, "\x00\x16\xea\x12\x34\x56", 6);
        get_mac_address(packet->addr2, "mon0");
        memcpy(packet->addr3, "\x00\x16\xea\x12\x34\x56", 6);
} else if (mode == 1) {
        memcpy(packet->addr1, "\x00\x16\xea\x12\x34\x56", 6);
        memcpy(packet->addr2, "\x00\x16\xea\x12\x34\x56", 6);
        memcpy(packet->addr3, "\xff\xff\xff\xff\xff\xff", 6);
}
packet->seq = 0;
tx_packet.packet = (uint8_t *)packet;
tx_packet.plen = sizeof(*packet) + packet_size;
```

```c
static inline uint32_t advance_lfsr(uint32_t lfsr)
{
        return (lfsr << 1) | (((lfsr >> 31) ^ (lfsr >> 29) ^ (lfsr >> 25) ^
                               (lfsr >> 24)) & 1);
}

void generate_payloads(uint8_t *buffer, size_t buffer_size)
{
        uint32_t lfsr = 0x1f3d5b79;
        uint32_t i;
        for (i = 0; i < buffer_size; ++i) {
                lfsr = advance_lfsr(lfsr);
                lfsr = advance_lfsr(lfsr);
                lfsr = advance_lfsr(lfsr);
                lfsr = advance_lfsr(lfsr);
                lfsr = advance_lfsr(lfsr);
                lfsr = advance_lfsr(lfsr);
                lfsr = advance_lfsr(lfsr);
                lfsr = advance_lfsr(lfsr);
                buffer[i] = lfsr & 0xff;
        }
}

int get_mac_address(uint8_t *buf, const char *ifname)
{
        int fd;
        struct ifreq ifr;

        /* Open generic socket */
        fd = socket(PF_INET, SOCK_PACKET, htons(ETH_P_ALL));
        if (fd == -1) {
                fprintf(stderr, "Error opening socket on %s to get MAC.\n",
                        ifname);
                return 1;
        }

        /* Store interface name */
        strcpy(ifr.ifr_name, ifname);

        /* Get Hardware Address (i.e., MAC) */
        if (-1 == ioctl(fd, SIOCGIFHWADDR, &ifr)) {
                fprintf(stderr, "Error calling SIOCGIFHWADDR to get MAC.\n");
                return 1;
        }

        /* Store it in the return buffer */
        memcpy(buf, ifr.ifr_hwaddr.sa_data, 6);

        /* Close the socket and return success */
        close(fd);
        return 0;
}
```

5

```c
static void init_lorcon()
{
        /* Parameters for LORCON */
        int drivertype = tx80211_resolvecard("iwlwifi");

        /* Initialize LORCON tx struct */
        if (tx80211_init(&tx, "mon0", drivertype) < 0) {
                fprintf(stderr, "Error initializing LORCON: %s\n",
                                tx80211_geterrstr(&tx));
                exit(1);
        }
        if (tx80211_open(&tx) < 0 ) {
                fprintf(stderr, "Error opening LORCON interface\n");
                exit(1);
        }

        /* Set up rate selection packet */
        tx80211_initpacket(&tx_packet);
}
```

6

```c
/* Send packets */
printf("Sending %u packets of size %u (. every thousand)\n", num_packets, packet_size);
if (delay_us) {
        /* Get start time */
        clock_gettime(CLOCK_MONOTONIC, &start);
}
for (i = 0; i < num_packets; ++i) {
        payload_memcpy(packet->payload, packet_size,
                       (i*packet_size) % PAYLOAD_SIZE);

        if (delay_us) {
                clock_gettime(CLOCK_MONOTONIC, &now);
                diff = (now.tv_sec - start.tv_sec) * 1000000 +
                       (now.tv_nsec - start.tv_nsec + 500) / 1000;
                diff = delay_us*i - diff;
                if (diff > 0 && diff < delay_us)
                        usleep(diff);
        }

        ret = tx80211_txpacket(&tx, &tx_packet);
        if (ret < 0) {
                fprintf(stderr, "Unable to transmit packet: %s\n",
                                tx.errstr);
                exit(1);
        }

        if (((i+1) % 1000) == 0) {
                printf(".");
                fflush(stdout);
        }
        if (((i+1) % 50000) == 0) {
                printf("%dk\n", (i+1)/1000);
                fflush(stdout);
        }
}
```

random_packets https://github.com/dhalperi/linux-80211n-csitool-supplementary/blob/master/injection/

Why plugin?

Taking *log_to_file*, from Intel 5300 CSI Tool, as an example …

log_to_file:
receive CSI data from driver and dump to disk

- Total of 150 LoC, 4 sections:
  1. headers & constants (30 LoC)
  2. Connect to driver via NetLink (30 LoC)
  3. Receive data & dump (50 LoC)

2

```c
/* Local variables */
struct sockaddr_nl proc_addr, kern_addr;       // addrs for recv, send, bind
struct cn_msg *cmsg;
char buf[4096];
int ret;
unsigned short l, l2;
int count = 0;

/* Make sure usage is correct */
check_usage(argc, argv);

/* Open and check log file */
out = open_file(argv[1], "w");

/* Setup the socket */
sock_fd = socket(PF_NETLINK, SOCK_DGRAM, NETLINK_CONNECTOR);
if (sock_fd == -1)
        exit_program_err(-1, "socket");

/* Initialize the address structs */
memset(&proc_addr, 0, sizeof(struct sockaddr_nl));
proc_addr.nl_family = AF_NETLINK;
proc_addr.nl_pid = getpid();                   // this process' PID
proc_addr.nl_groups = CN_IDX_IWLAGN;
memset(&kern_addr, 0, sizeof(struct sockaddr_nl));
kern_addr.nl_family = AF_NETLINK;
kern_addr.nl_pid = 0;                                  // kernel
kern_addr.nl_groups = CN_IDX_IWLAGN;

/* Now bind the socket */
if (bind(sock_fd, (struct sockaddr *)&proc_addr, sizeof(struct sockaddr_nl)) == -1)
        exit_program_err(-1, "bind");

/* And subscribe to netlink group */
{
        int on = proc_addr.nl_groups;
        ret = setsockopt(sock_fd, 270, NETLINK_ADD_MEMBERSHIP, &on, sizeof(on));
        if (ret)
                exit_program_err(-1, "setsockopt");
}
```

3

```c
/* Poll socket forever waiting for a message */
while (1)
{
        /* Receive from socket with infinite timeout */
        ret = recv(sock_fd, buf, sizeof(buf), 0);
        if (ret == -1)
                exit_program_err(-1, "recv");
        /* Pull out the message portion and print some stats */
        cmsg = NLMSG_DATA(buf);
        if (count % SLOW_MSG_CNT == 0)
                printf("received %d bytes: id: %d val: %d seq: %d clen: %d\n", cmsg-
        /* Log the data to file */
        l = (unsigned short) cmsg->len;
        l2 = htons(l);
        fwrite(&l2, 1, sizeof(unsigned short), out);
        ret = fwrite(cmsg->data, 1, l, out);
        if (count % 100 == 0)
                printf("wrote %d bytes [msgcnt=%u]\n", ret, count);
        ++count;
        if (ret != l)
                exit_program_err(1, "fwrite");
}

exit_program(0);
return 0;
```

```c
FILE* open_file(char* filename, char* spec)
{
        FILE* fp = fopen(filename, spec);
        if (!fp)
        {
                perror("fopen");
                exit_program(1);
        }
        return fp;
}

void caught_signal(int sig)
{
        fprintf(stderr, "Caught signal %d\n", sig);
        exit_program(0);
}

void exit_program(int code)
{
        if (out)
        {
                fclose(out);
                out = NULL;
        }
        if (sock_fd != -1)
        {
                close(sock_fd);
                sock_fd = -1;
        }
        exit(code);
}

void exit_program_err(int code, char* func)
{
        perror(func);
        exit_program(code);
```

log_to_file https://github.com/dhalperi/linux-80211n-csitool-supplementary/blob/master/netlink/log_to_file.c

Powerful & Agile CSI Measurement via PicoScenes Plugin Mechanism

Taking *"—mode logger"*, from PicoScenes EchoProbe plugin, as a comparison …

—mode logger of EchoProbe plugin:
receive CSI data from driver and dump to disk

- **Essentially, Just 2 LoC**
  - Grab the frame's buffer
  - Dump the buffer to disk via RXSDumper
- **Rest of the work, ALL DONE by PicoScenes middleware !**
  - Communicating with kernel
  - Receiving the raw buffer
  - Package the buffer to ModularPicoScenesRxFrame format
  - Pass the frame to EchoProbe plugin
  - etc…

Enabling Researchers to prototype their plugins focusing on their CSI measurement task!

```
void EchoProbeResponder::handle(const ModularPicoScenesRxFrame &rxframe) {
    if (parameters.workingMode == MODE_Injector || parameters.workingMode == MODE_EchoProbeInitiator)
        return;

    if (parameters.workingMode == MODE_Logger) {
        auto buffer = rxframe.toBuffer();
        if (!parameters.outputFileName)
            RXSDumper::getInstance("rx_" + nic->getReferredInterfaceName()).dumpRXS(buffer.data(), buffer.size());
        else
            RXSDumper::getInstance(*parameters.outputFileName).dumpRXS(buffer.data(), buffer.size());
        return;
    }
```

EchoProbe plugin https://gitlab.com/wifisensing/PicoScenes-PDK

Powerful & Agile CSI Measurement via PicoScenes Plugin Mechanism

With such simplicity, EchoProbe plugin goes far beyond just a CSI logger…

—mode injector of EchoProbe plugin:
packet injection using all support frontends

- **Essentially, Just 3 LoC**
  - Generate frame & specify Tx parameters
  - Transmit the frames and wait
  - Wait a while a go next frame
- **Rest of the work, ALL DONE by PicoScenes middleware !**
  - generate MPDU buffer,
  - initiate & transmit MPDU buffer via COTS NICs
  - or, generate signal transmit PPDU signal via SDR
  - *etc*…

1. Generate frame & Specify Tx parameters

```
std::shared_ptr<PicoScenesFrameBuilder> EchoProbeInitiator::buildBasicFrame(uint16_t taskId, const EchoProbePacketFra
    auto fp = std::make_shared<PicoScenesFrameBuilder>(nic);

    if (frameType == SimpleInjectionFrameType && parameters.injectorContent == EchoProbeInjectionContent::NDP) {
        fp->makeFrame_NDP();
        /**
         * @brief PicoScenes Platform CLI parser has *absorbed* the common Tx parameters.
         * The platform parser will parse the Tx parameters options and store the results in AbstractNIC.
         * Plugin developers now can access the parameters via a new method nic->getUserSpecifiedTxParameters().
         */
        fp->setTxParameters(nic->getUserSpecifiedTxParameters());
        return fp;
    } else {
        fp->makeFrame_HeaderOnly();
        /**
         * @brief PicoScenes Platform CLI parser has *absorbed* the common Tx parameters.
         * The platform parser will parse the Tx parameters options and store the results in AbstractNIC.
         * Plugin developers now can access the parameters via a new method nic->getUserSpecifiedTxParameters().
         */
        fp->setTxParameters(nic->getUserSpecifiedTxParameters());
        fp->setTaskId(taskId);
        fp->setPicoScenesFrameType(frameType);

        if (frameType == SimpleInjectionFrameType && parameters.injectorContent == EchoProbeInjectionContent::Full) {
            fp->addExtraInfo();
        }

        if (frameType == EchoProbeRequestFrameType)
            fp->addExtraInfo();
    }

    fp->setSourceAddress(PicoScenesFrameBuilder::magicIntel123456.data());
    fp->setDestinationAddress(PicoScenesFrameBuilder::magicIntel123456.data());
    fp->set3rdAddress(nic->getFrontEnd()->getMacAddressPhy().data());

    if (parameters.inj_for_intel5300.value_or(false)) {
        fp->setForceSounding(false);
        fp->setChannelCoding(ChannelCodingEnum::BCC); // IWL5300 doesn't support LDPC coding.
    }

    return fp;
}
```

2. Transmit

```
if (workingMode == MODE_Injector) {
    fp = buildBasicFrame(taskId, SimpleInjectionFrameType, sessionId);
    fp->transmitSync();
    tx_count++;
    total_tx_count++;
    if (LoggingService::localDisplayLevel == Trace)
        printDots(tx_count);
    SystemTools::Time::delay_periodic(parameters.tx_delay_us);
} else if (workingMode == MODE_EchoProbeInitiator) {
```

EchoProbe plugin https://gitlab.com/wifisensing/PicoScenes-PDK

Powerful & Agile CSI Measurement via PicoScenes Plugin Mechanism

With such simplicity, EchoProbe plugin goes far beyond just a CSI logger…

—mode responder of EchoProbe plugin:
receive packets & transmit them back to *initiator*

- **Essentially, Just 50 LoC**
  - Receive RxFrame (CSI and MPDU content)
  - Generate Reply frames (packet the RxFrame as payload)
  - Transmit the Reply frames
- **Rest of the work, ALL DONE by PicoScenes middleware !**
  - **Frame auto segmentation**
  - initiate & transmit MPDU buffer via COTS NICs
  - or, generate signal transmit PPDU signal via SDR
  - *etc*…

2. Transmit

```
if (rxframe.PicoScenesHeader->frameType == EchoProbeRequestFrameType) {
    auto replies = makeReplies(rxframe, epSegment.getEchoProbeRequest());
    for (auto i = 0; i < 1 + (rxframe.PicoScenesHeader->deviceType == PicoScenesDev
        for (auto &reply: replies) {
            reply.transmit();
        }
    }
}
```

1. Generate reply frames

```
std::vector<PicoScenesFrameBuilder> EchoProbeResponder::makeRepliesForEchoProbeRequest(con
    auto frameBuilder = PicoScenesFrameBuilder(nic);
    frameBuilder.makeFrame_HeaderOnly();

    EchoProbeReply reply;
    reply.sessionId = epReq.sessionId;
    if (epReq.replyStrategy == EchoProbeReplyStrategy::ReplyWithFullPayload) {
        frameBuilder.addExtraInfo();
        reply.replyStrategy = EchoProbeReplyStrategy::ReplyWithFullPayload;
        reply.payloadName = "EchoProbeReplyFull";
        frameBuilder.addSegment(std::make_shared<EchoProbeReplySegment>(reply));
        frameBuilder.addSegment(std::make_shared<PayloadSegment>(reply.payloadName, rxfram
    } else if (epReq.replyStrategy == EchoProbeReplyStrategy::ReplyWithCSI) {
        frameBuilder.addExtraInfo();
        reply.replyStrategy = EchoProbeReplyStrategy::ReplyWithCSI;
        reply.payloadName = "EchoProbeReplyCSI";
        frameBuilder.addSegment(std::make_shared<EchoProbeReplySegment>(reply));
        frameBuilder.addSegment(std::make_shared<PayloadSegment>(reply.payloadName, rxfram
    } else if (epReq.replyStrategy == EchoProbeReplyStrategy::ReplyWithExtraInfo) {
        frameBuilder.addExtraInfo();
        reply.replyStrategy = EchoProbeReplyStrategy::ReplyWithExtraInfo;
        frameBuilder.addSegment(std::make_shared<EchoProbeReplySegment>(reply));
    } else if (epReq.replyStrategy == EchoProbeReplyStrategy::ReplyOnlyHeader) {
        reply.replyStrategy = EchoProbeReplyStrategy::ReplyOnlyHeader;
        frameBuilder.addSegment(std::make_shared<EchoProbeReplySegment>(reply));
    }


    frameBuilder.setPicoScenesFrameType(EchoProbeReplyFrameType);
    frameBuilder.setTxParameters(nic->getUserSpecifiedTxParameters());
    frameBuilder.setSourceAddress(PicoScenesFrameBuilder::magicIntel123456.data());
    frameBuilder.setDestinationAddress(PicoScenesFrameBuilder::magicIntel123456.data());
    frameBuilder.set3rdAddress(nic->getFrontEnd()->getMacAddressPhy().data());

    if (parameters.inj_for_intel5300.value_or(false)) {
        frameBuilder.setForceSounding(false);
        frameBuilder.setChannelCoding(ChannelCodingEnum::BCC); // IWL5300 doesn't support
    }
    frameBuilder.setTaskId(rxframe.PicoScenesHeader->taskId);
    frameBuilder.setTxId(rxframe.PicoScenesHeader->txId);
    return std::vector<PicoScenesFrameBuilder>{frameBuilder};
}
```

EchoProbe plugin https://gitlab.com/wifisensing/PicoScenes-PDK

Powerful & Agile CSI Measurement via PicoScenes Plugin Mechanism

With such simplicity, EchoProbe plugin goes far beyond just a CSI logger…

—mode initiator of EchoProbe plugin:
initiate the roundtrip and spectrum scanning CSI measurement with the *responder* side

- **Totally, 420 LoC**
  - Handling various Tx/Rx options
  - EchoProbe Request / FreqChangeRequest generation
  - Initiator<->Responder ACK-Retry protocol
  - Frequency and bandwidth scanning protocol
  - Available Frequency and bandwidth enumeration
  - Dump the received frames
  - …
- **Rest of the work, ALL DONE by PicoScenes middleware !**
  - **Frame auto concatenation**
  - conditional Rx (waiting for Replay frames) and timeout
  - Communicate with heterogeneous frontend
  - *etc*…

EchoProbe plugin https://gitlab.com/wifisensing/PicoScenes-PDK

Powerful & Agile CSI Measurement via PicoScenes Plugin Mechanism

The official PicoScenes Plugin Development Kit (PicoScenes-PDK) opensources three plugins

PicoScenes-PDK project

- **Demo Plugin**
  - a rich-documented tutorial plugin
  - Describe how the plugin communicate with the platform, how to handle RxFrame,
- **EchoProbe Plugin**
  - 4 modes, injector/logger, initiator/responder
  - **Single/Round-trip CSI measurement**
  - **Spectrum scanning CSI measurement**
- **UDP-Forwarder Plugin**
  - Forward received frames to the remove via UDP
  - **Total 50 LoC, the core is just 1 LoC.**

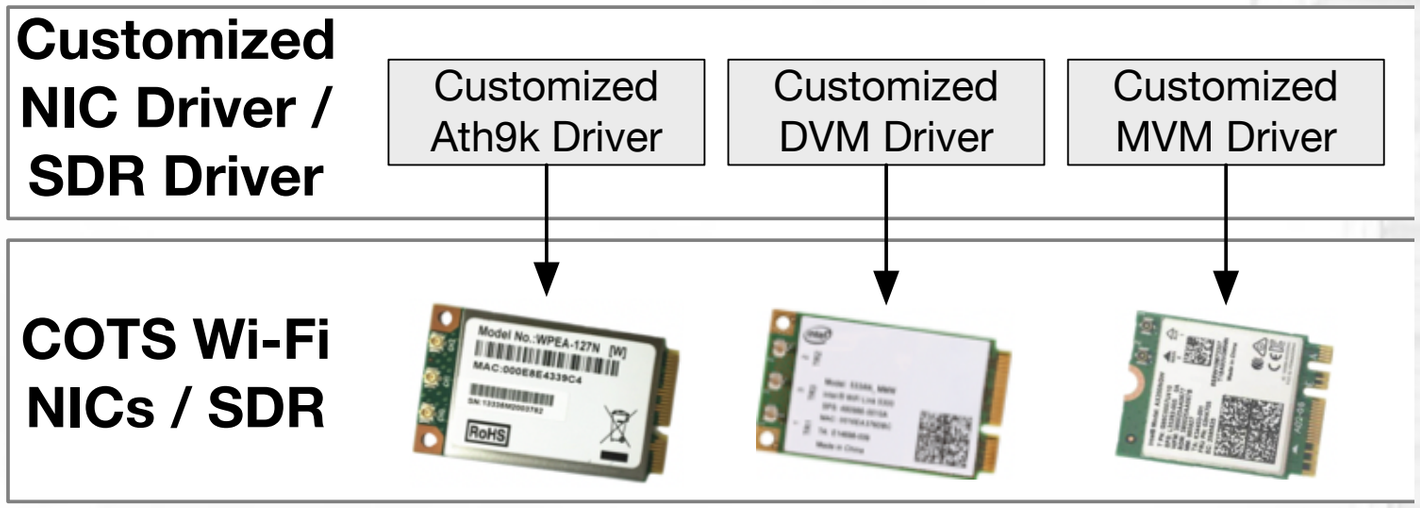
PicoScenes-PDK https://gitlab.com/wifisensing/PicoScenes-PDK

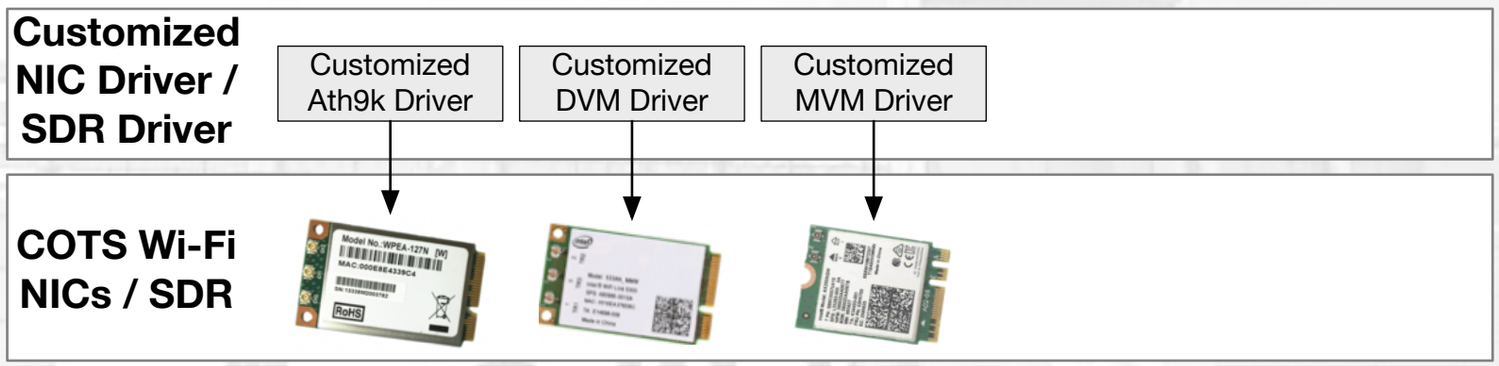
The Architecture of PicoScenes System

https://ps.zpj.io

PicoScenes Architecture

PicoScenes project roots and initiated upon *the customized drivers* for QCA9300, IWL5300, and AX200/210/211NICs.





https://ps.zpj.io

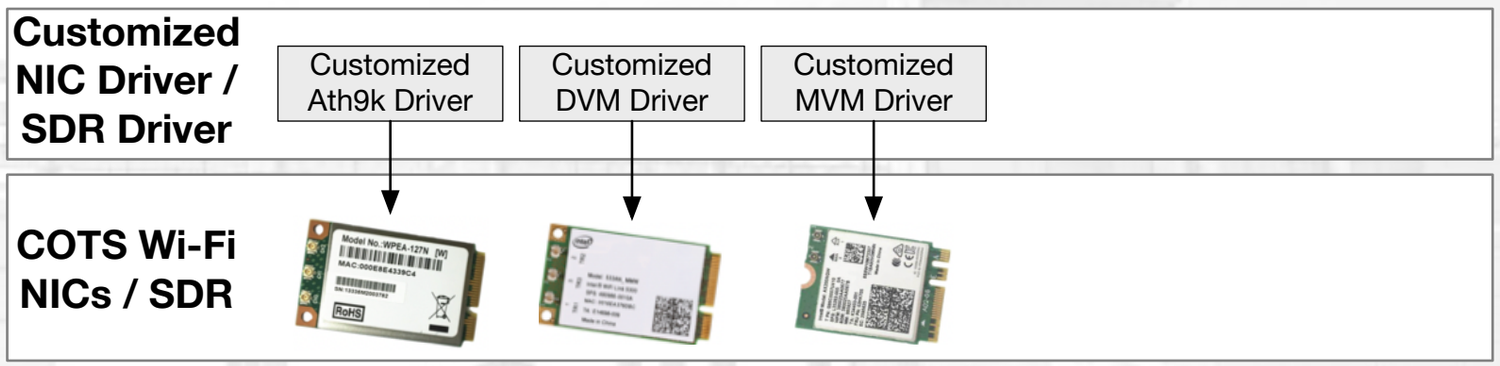
PicoScenes Architecture

PicoScenes project roots and initiated upon *the customized drivers* for QCA9300, IWL5300, and AX200/210/211NICs. Some highlights are …

for all supported NICs	
• **Simultaneous multi-NIC CSI measurement** • **Rich output, MPDU (packet) content + CSI + various HW status** • **Unified data format with backward compatibility** • Offer common HW settings, e.g., Tx/Rx radio chain selection, Tx power	
for AX200/210/211	**for QCA9300**

• **Wi-Fi 6E-ready CSI measurement** *1
  • Unlock the 6-GHz band for AX210/211
  • All-format CSI measurement (802.11a/g/n/ac/ax)
  • Up to 160-MHz bandwidth (1992 subcarriers per-stream)
• **Fully passive CSI measurement for all overheard frames** *2
• Packet injection in all format
	• **Arbitrary carrier freq. tuning** (2.2-2.9 and 4.4-6.1 GHz)
	• **Arbitrary sampling rate tuning** (2.5 to 80 MHz)
	• **Manual Rx-Gain**
	• Extra Spatial Sounding (ESS)

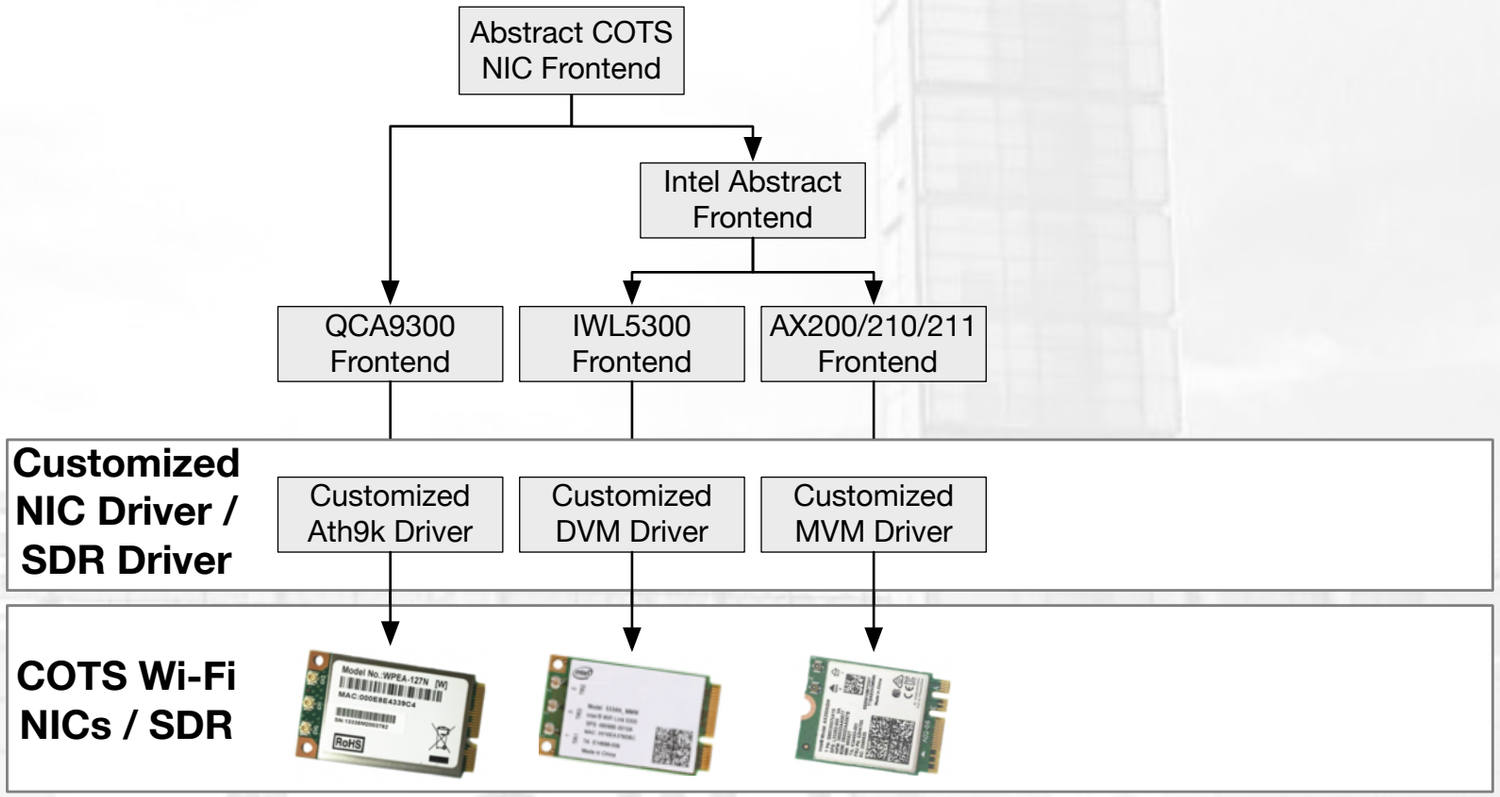
* *Wi-Fi 6E = 11ax + 160MHz BW + 6 GHz band*
* *CSI measurement in monitor mode supports all formats and all coding (BCC, LDPC)*



https://ps.zpj.io

PicoScenes Architecture

PicoScenes platform encapsulates the low-level controls into a hierarchy of AbstractFrontEnd (API).



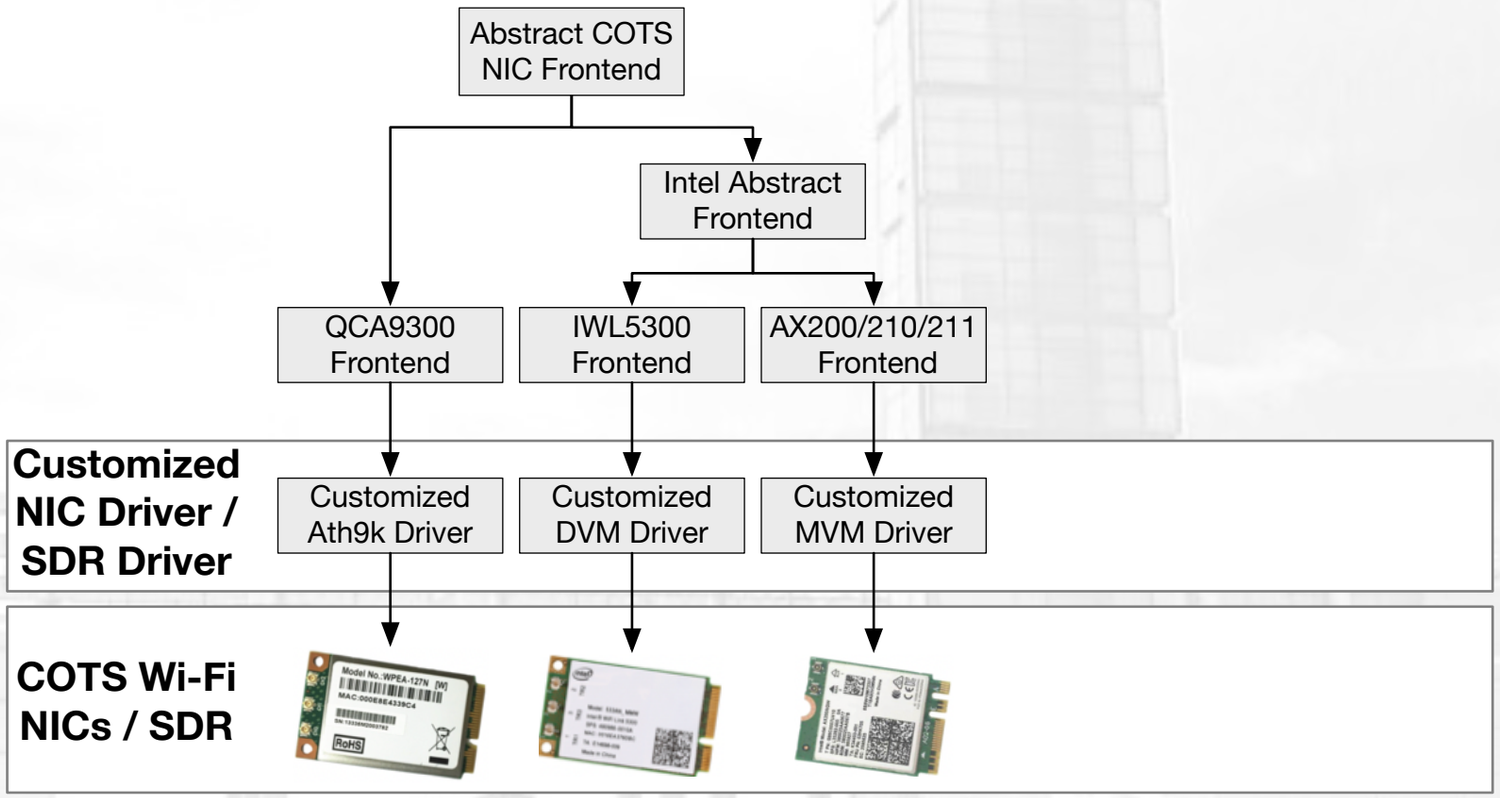
https://ps.zpj.io

PicoScenes Architecture

PicoScenes platform encapsulates the low-level controls into a hierarchy of AbstractFrontEnd (API). It offers …

HW-neutral APIs to

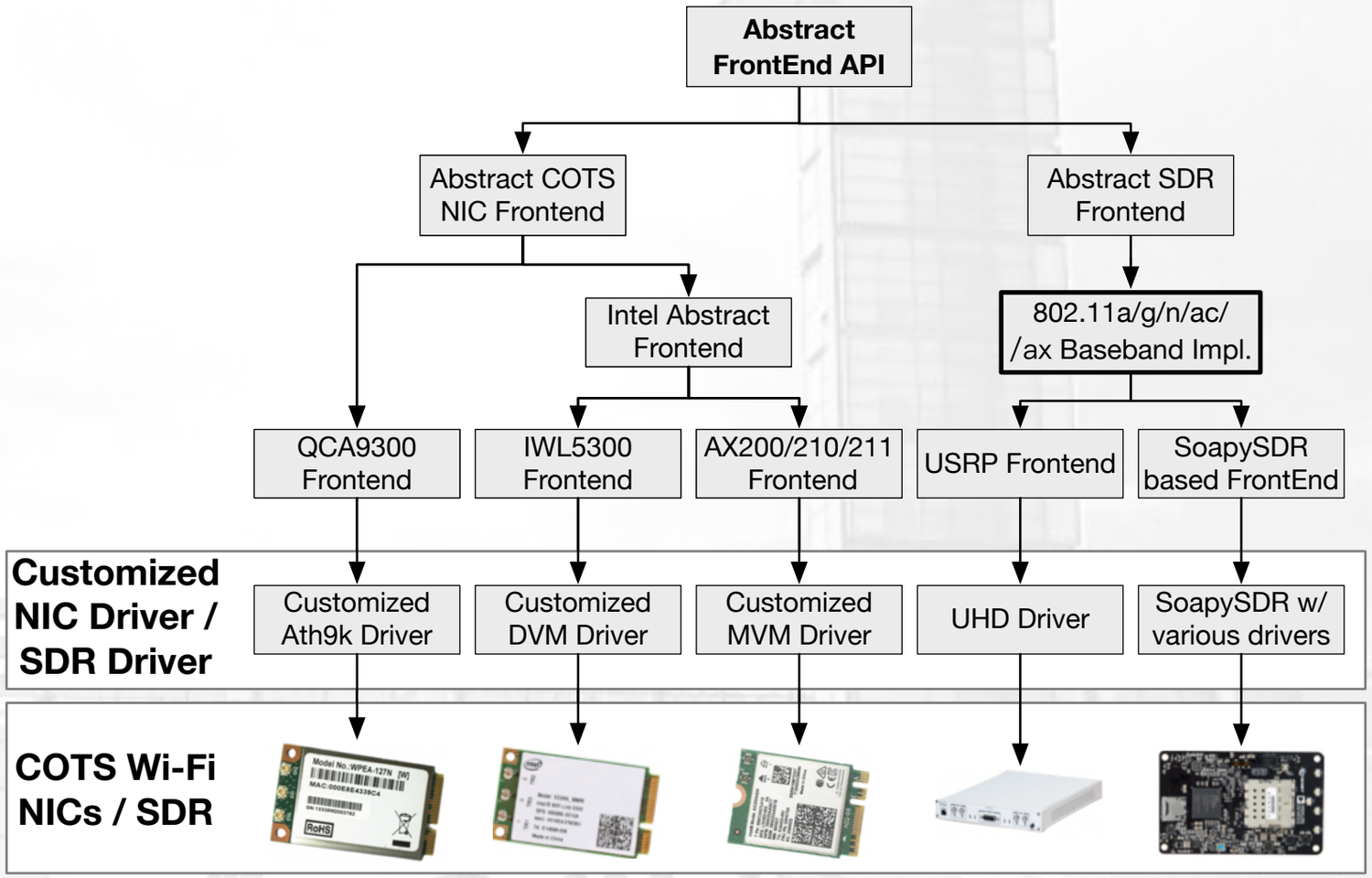
- **Transmit packets**
- **Receive packets**
- **Receive CSI measurement**
- **query HW status**
- **access various settings**
  - channel
  - bandwidth
  - carrier frequency
  - sampling rate
  - Tx parameters
  - Tx/Rx gain
  - Tx/Rx chain mask
  - …



https://ps.zpj.io

PicoScenes Architecture

Based on our Wi-Fi baseband implementation, PicoScenes supports various SDR frontend, and encapsulates them by AbstractSDRFrontEnd API.



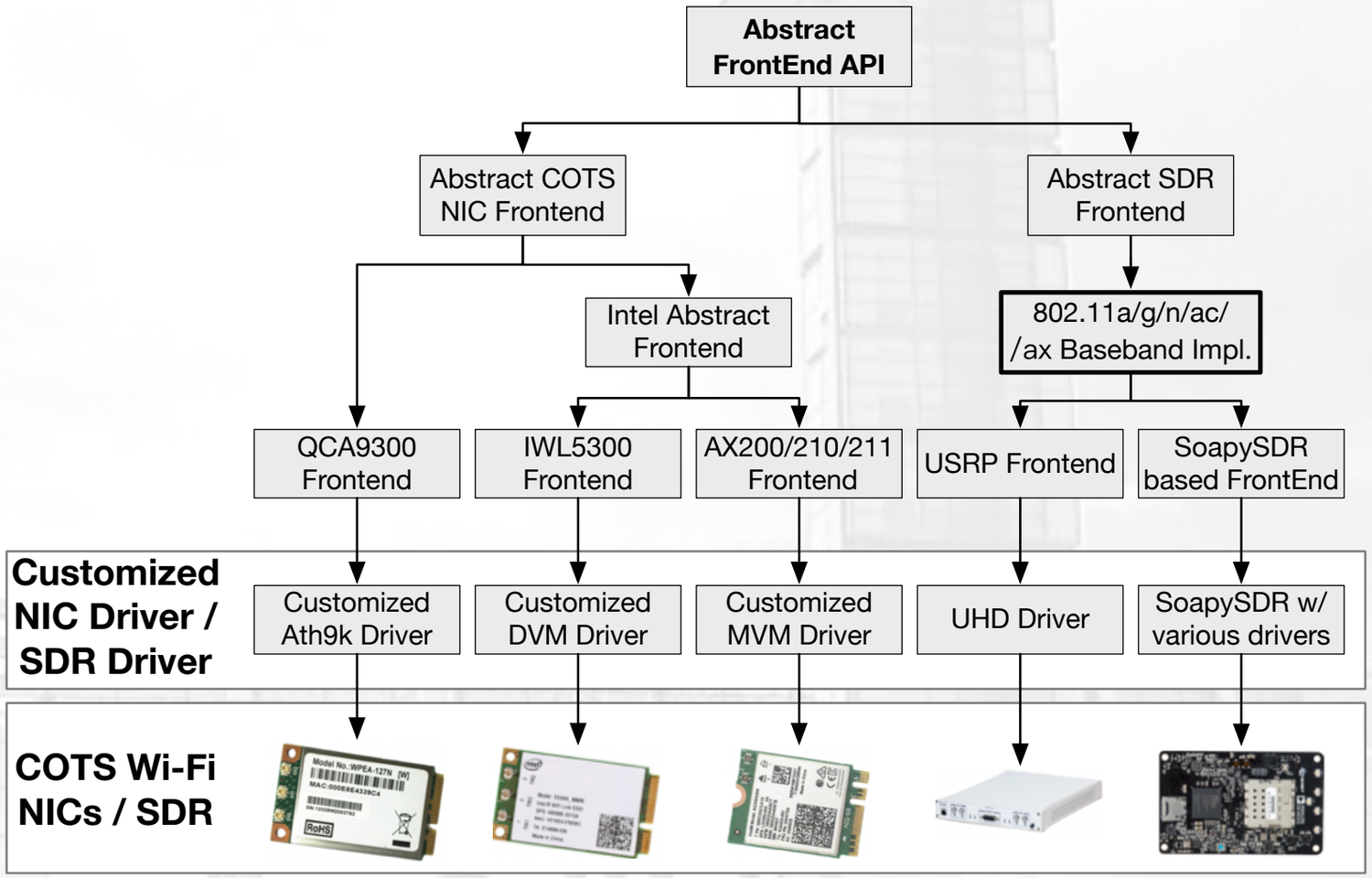
https://ps.zpj.io

PicoScenes Architecture

Based on our Wi-Fi baseband implementation, PicoScenes supports various SDR frontend, and encapsulates them by AbstractSDRFrontEnd API, which offers…

HW-neutral APIs to

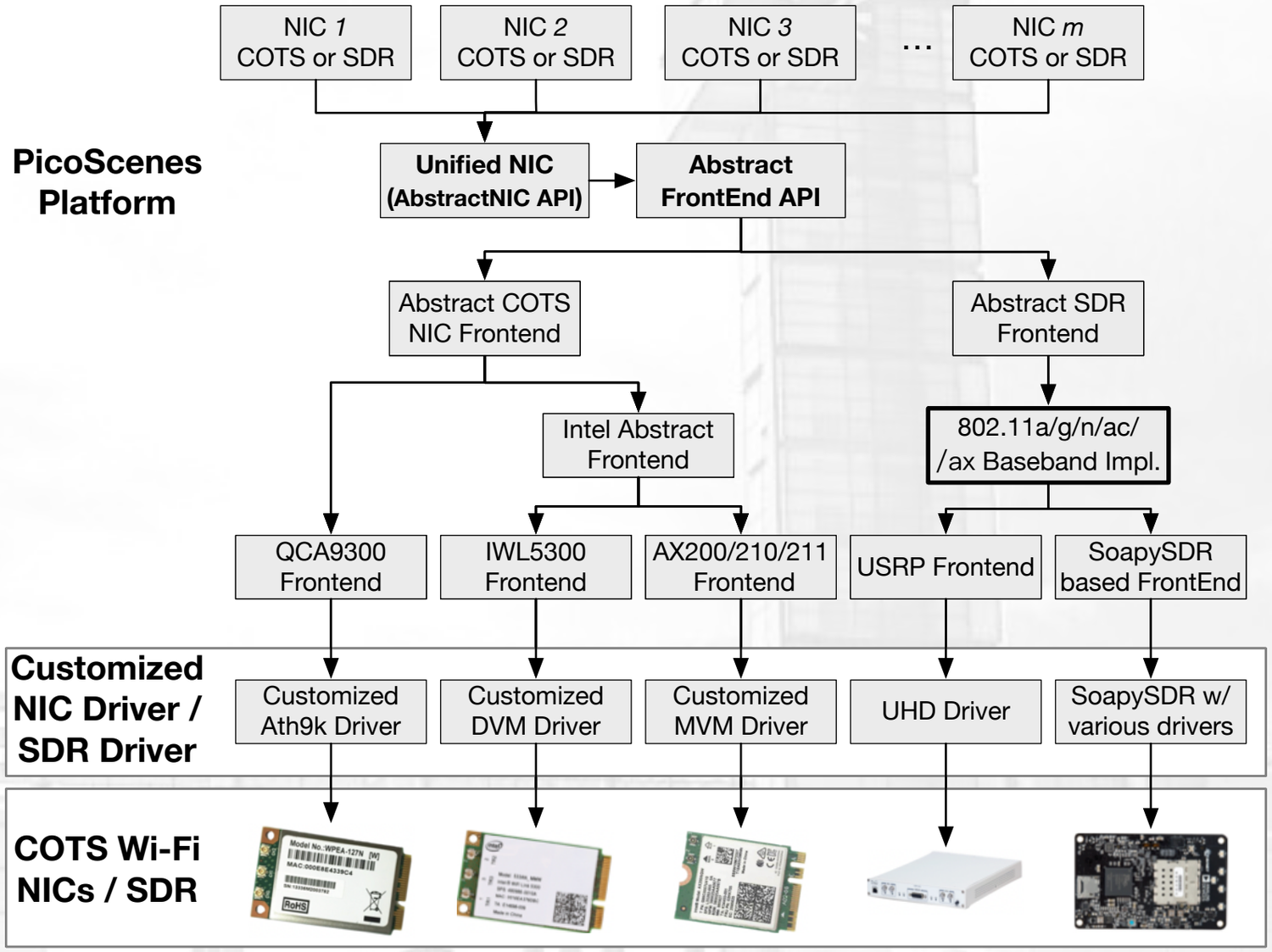
- **Transmit & receive packets**
- **Access Full PHY decoding results**
  - Multi-stage CSI
    - Legacy CSI (L-LTF)
    - CSI (HT/VHT/HE-LTF)
    - Pilot CSI (Pilot subcarriers per OFDM symbol)
  - CFO & SFO estimation
  - Raw baseband signal
  - *etc*…
- **Complete Control over Tx/Rx PHY**
  - Carrier frequency
  - Sampling rate
  - Antenna control
  - Tx/Rx gain
  - Tx/Rx chain mask
  - Time/Clock sync.
  - Precoding / beamforming
  - Over/down-sampling
  - *etc*…



https://ps.zpj.io

PicoScenes Architecture

We add another layer of abstraction, AbstractNIC.



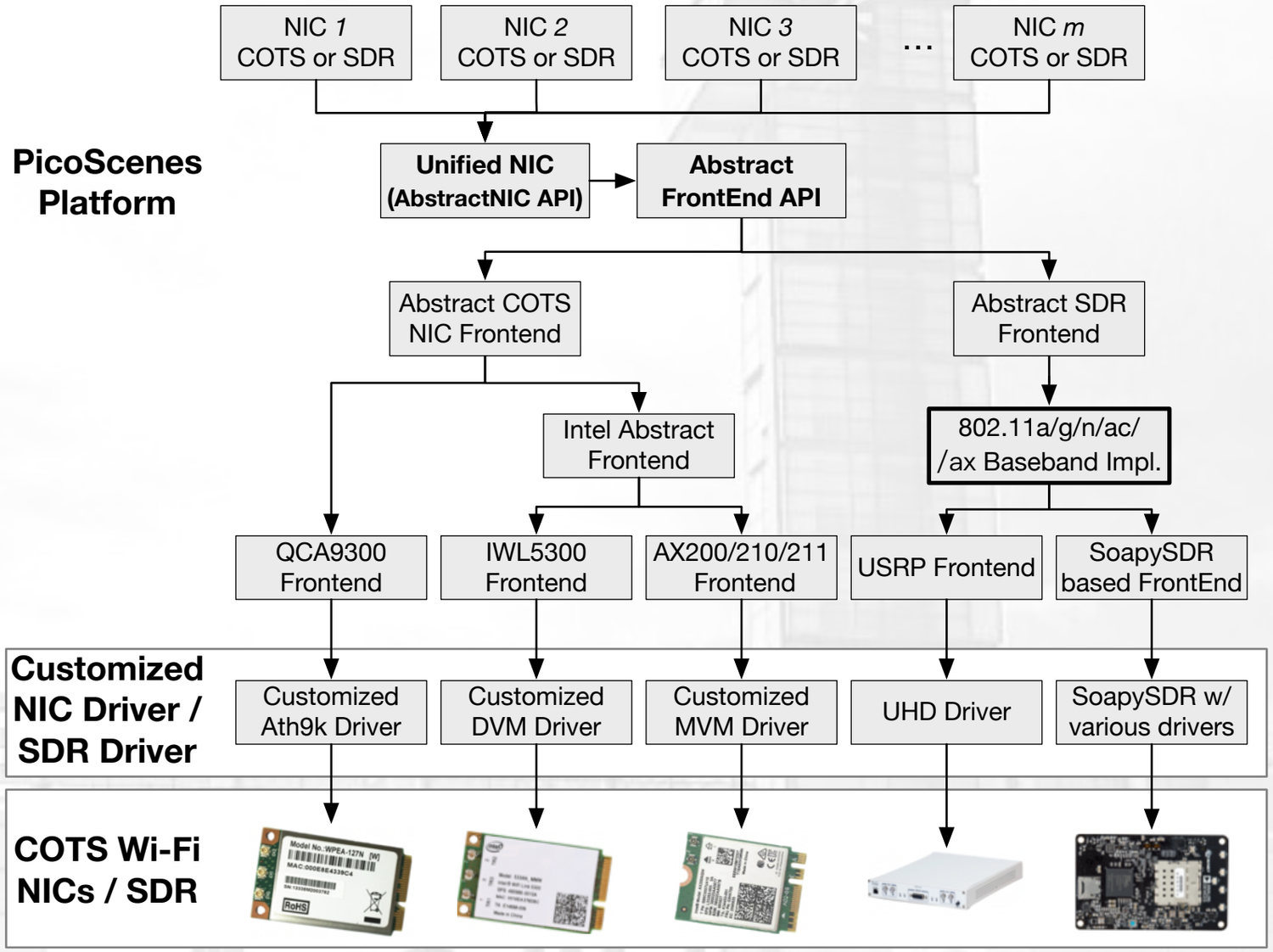
https://ps.zpj.io

PicoScenes Architecture

We add another layer of abstraction, AbstractNIC. It offers …

HW-neutral APIs to

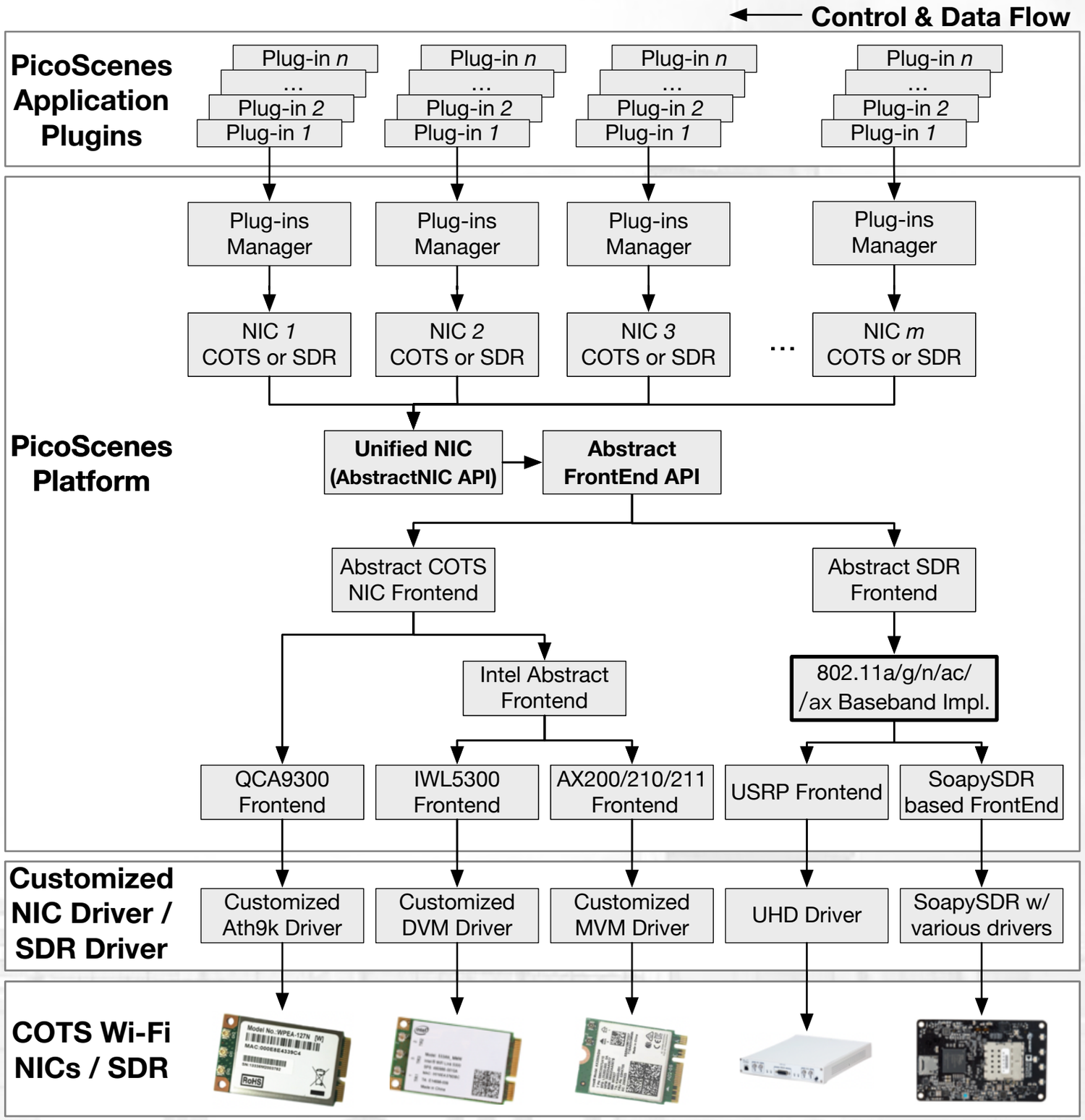
- **Multi-frontend concurrency**
- **Unification for heterogeneous HW**
- **Tx & Rx queue control**
- **Timed batch Tx**
- Signal generation
- **Conditional Rx for**
  - certain formats …
  - certain type …
  - arbitrary criteria
- CLI option parsing
- *etc*…



https://ps.zpj.io

PicoScenes Architecture

Each AbstractNIC has its own Plugin Manager, which hosts and executes the atop PicoScenes plugin layer.



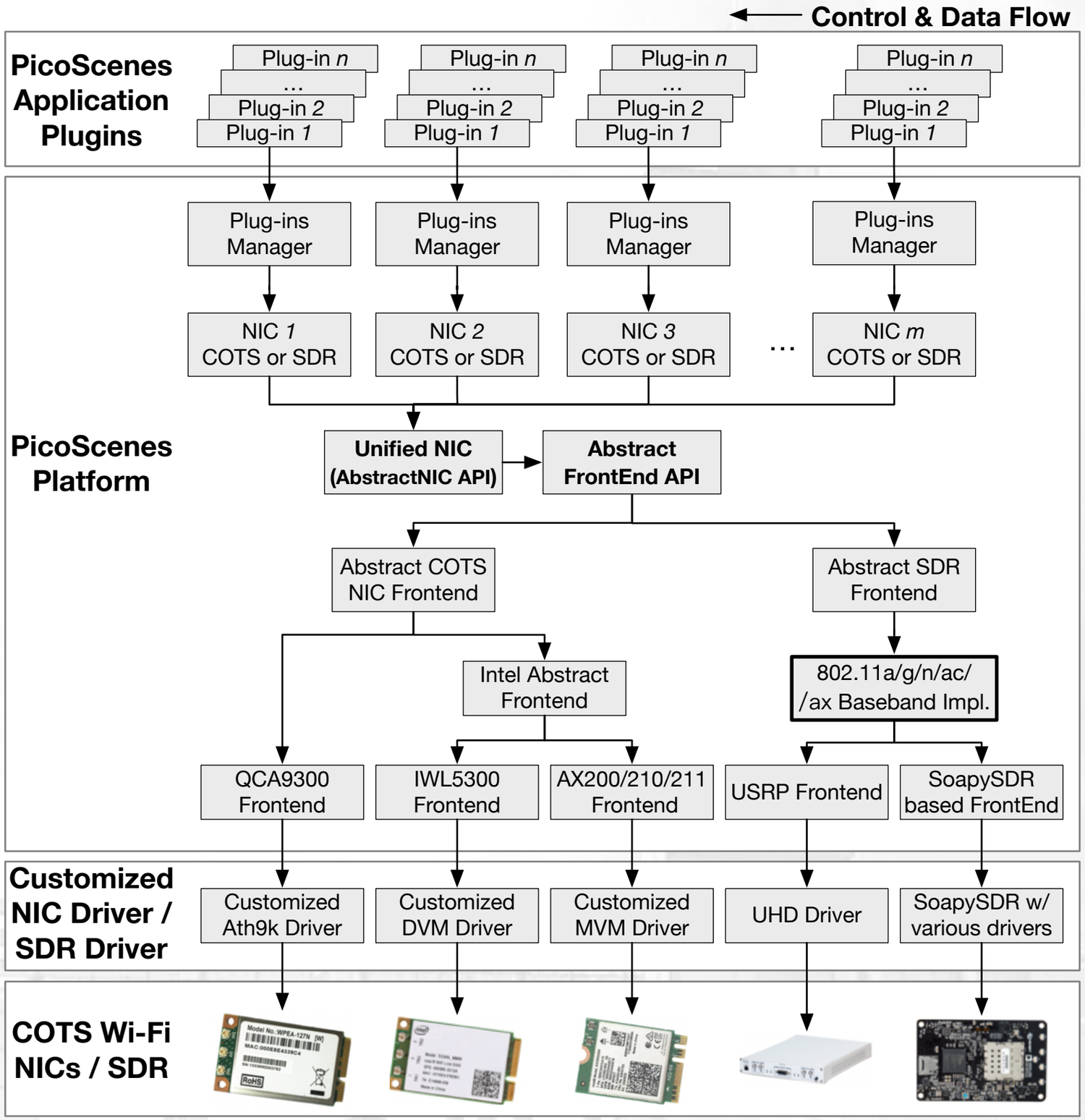
https://ps.zpj.io

PicoScenes Architecture

Each AbstractNIC has its own Plugin Manager, which hosts and executes the atop PicoScenes plugin layer. This architecture offers …

HW-neutral APIs to

- **CSI measurement middleware mechanism**
- Extensibility & scalability
- Plugin isolation & concurrency
- **Implementing enduser functionalities**
  - CSI dump to disk
  - packet injection
  - CSI data UDP forwarding
  - Round-trip CSI measurement
  - Freq. hopping CSI measurement
  - *etc*…



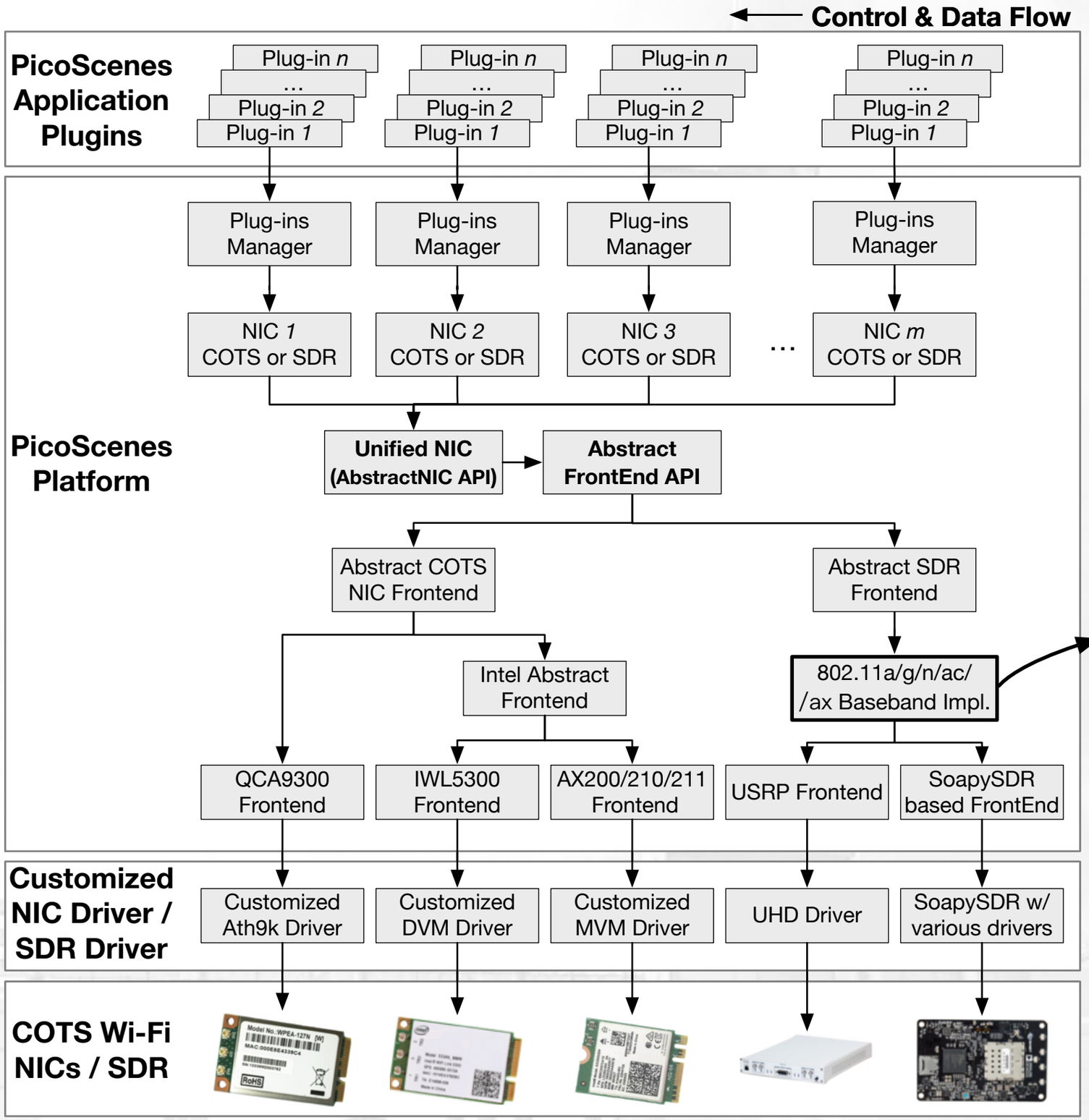
https://ps.zpj.io

PicoScenes Architecture

Each AbstractNIC has its own Plugin Manager, which hosts and executes the atop PicoScenes plugin layer. This architecture offers …

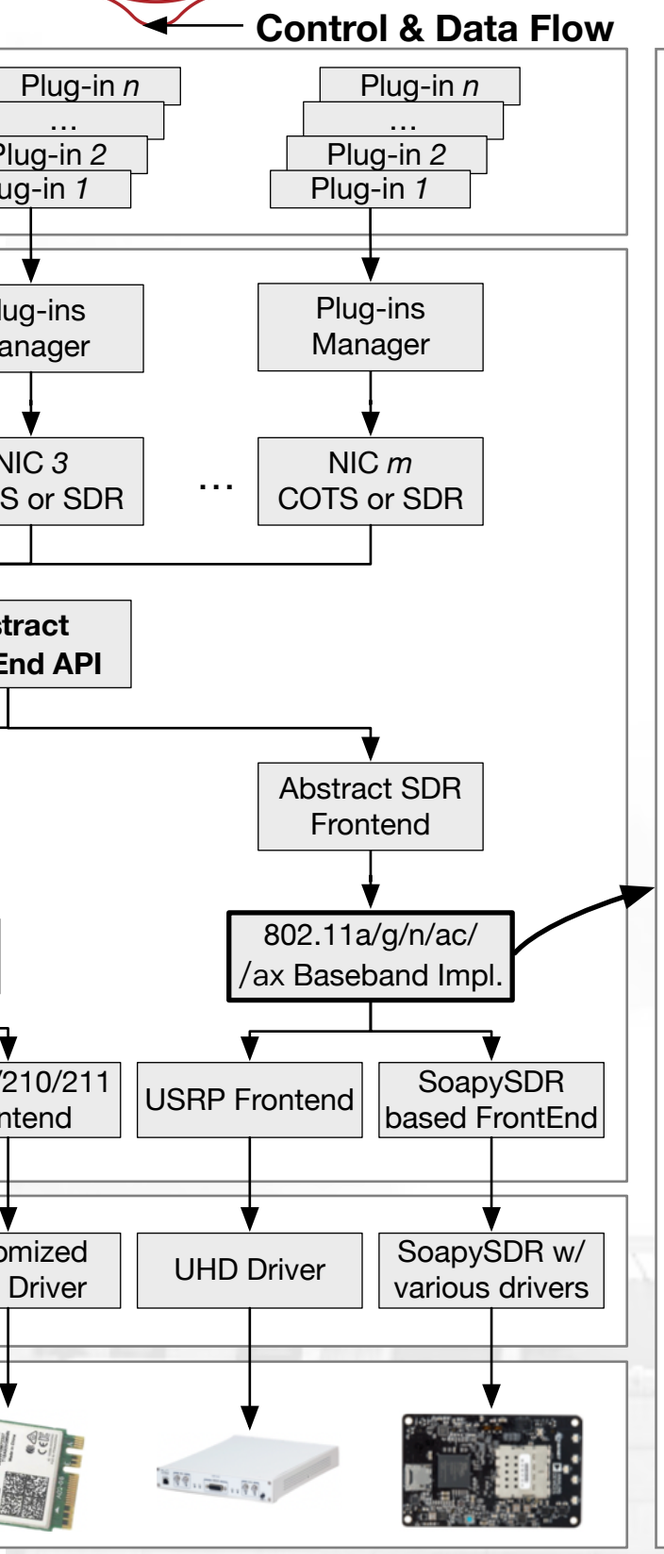
HW-neutral APIs to

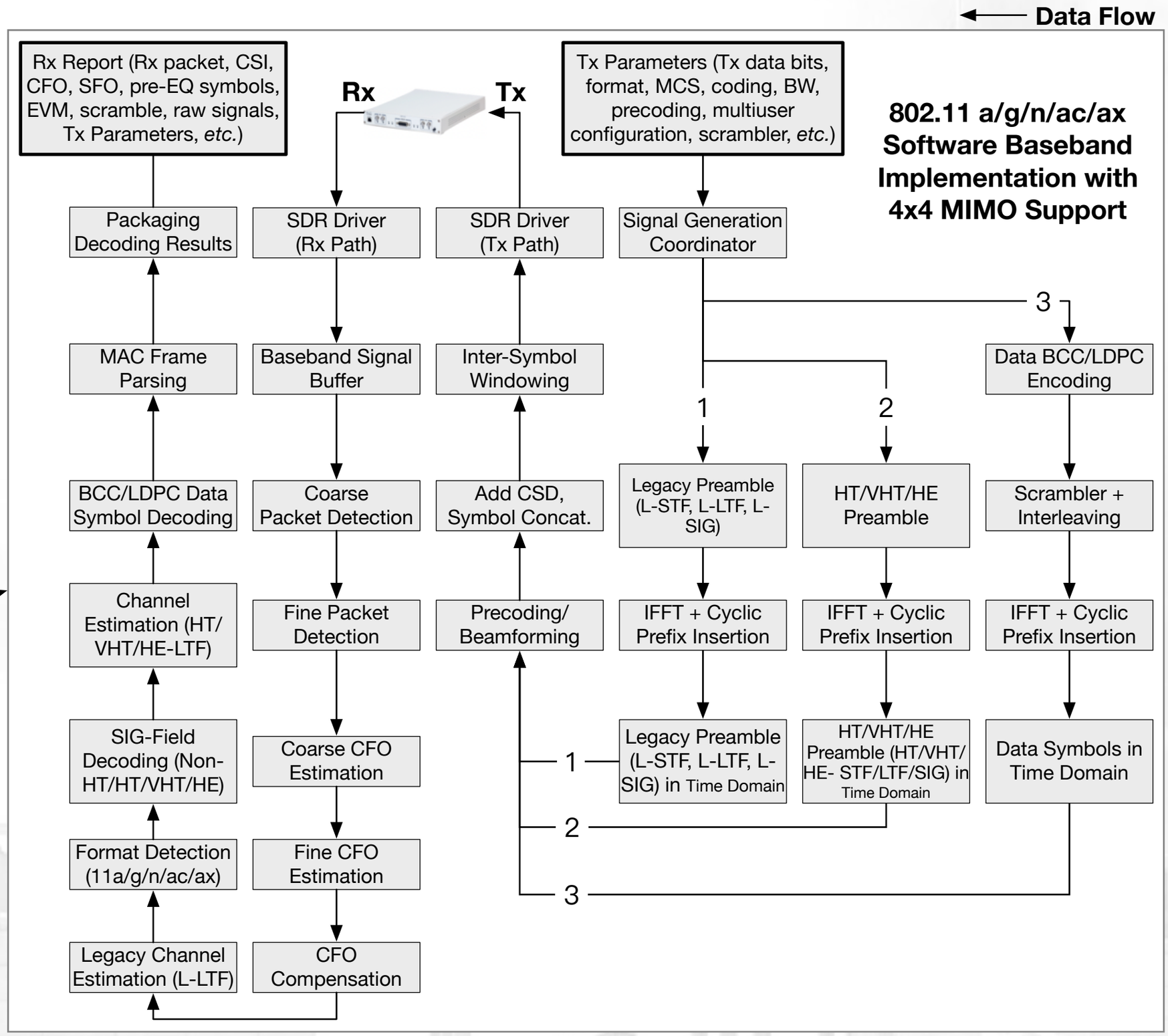
- **CSI measurement middleware mechanism**
- Extensibility & scalability
- Plugin isolation & concurrency
- **Implementing enduser functionalities**
  - CSI dump to disk
  - packet injection
  - CSI data UDP forwarding
  - Round-trip CSI measurement
  - Freq. hopping CSI measurement
  - *etc*…



https://ps.zpj.io

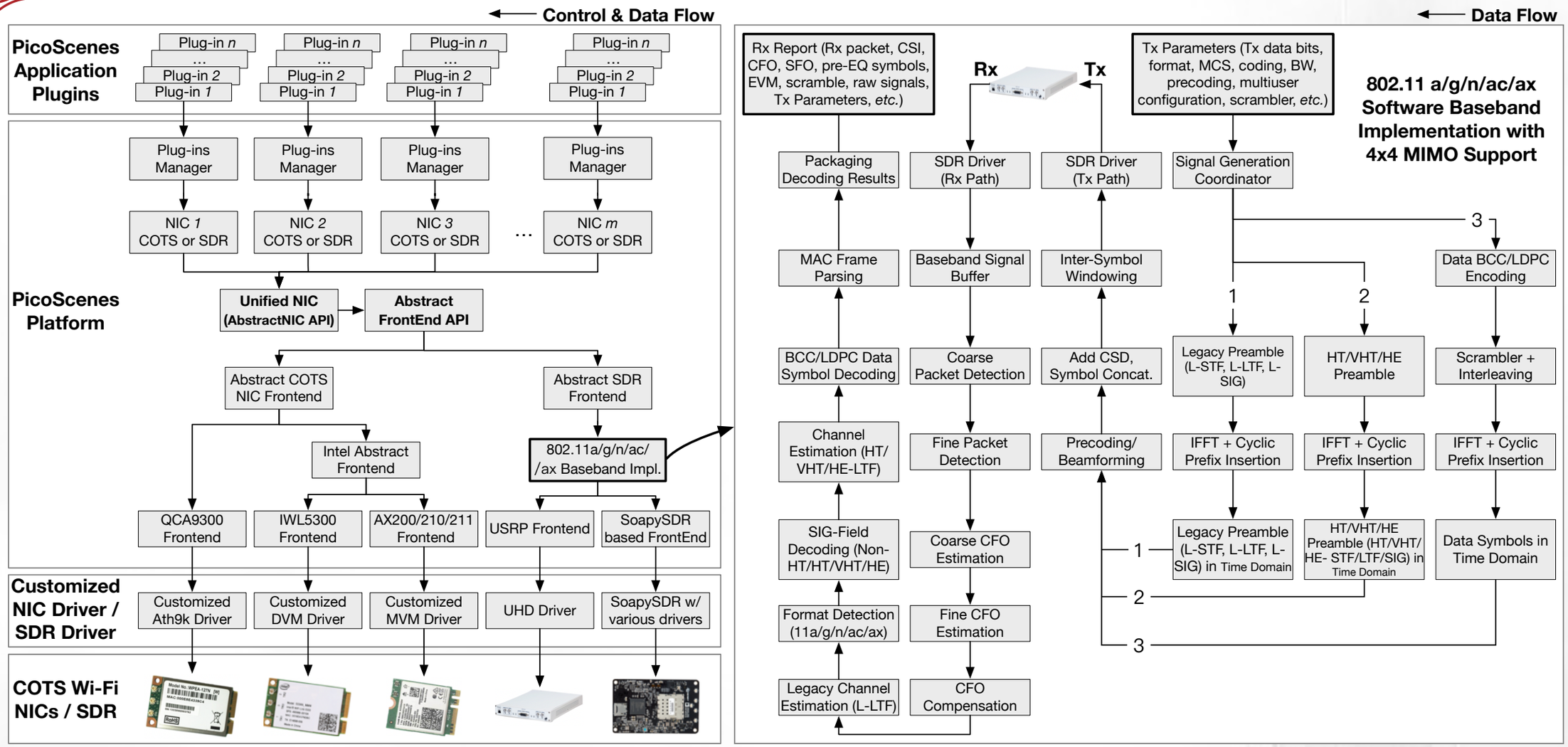
PicoScenes Architecture





https://ps.zpj.io

PicoScenes Architecture



PicoScnees Wi-Fi baseband Implementation

- **Transform SDR into a Wi-Fi NIC, live decoding & packet injection**
- **Full compliance with 802.11a/g/n/ac/ax standard**
- **Cover most advanced features,** *e.g.*, all-format, all-bandwidth (20/40/80/160-MHz), all-codec (BCC/LDPC), all-MCS (0-11), precoding, beamforming, and up to 4x4 MIMO.
- Full control & access to PHY
- High-performance, decoding in realtime with 0 packet loss in 20/40-MHz bandwidth

https://ps.zpj.io

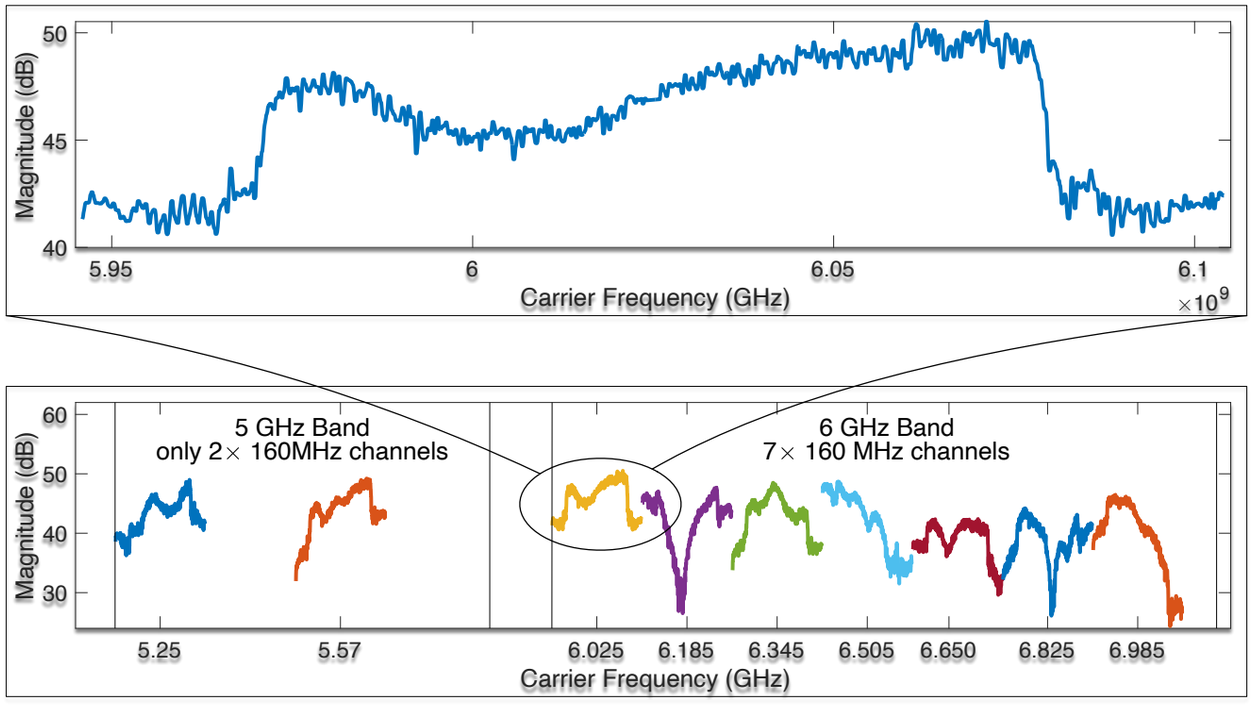
How PicoScenes can supercharge your next Wi-Fi sensing research?

https://ps.zpj.io

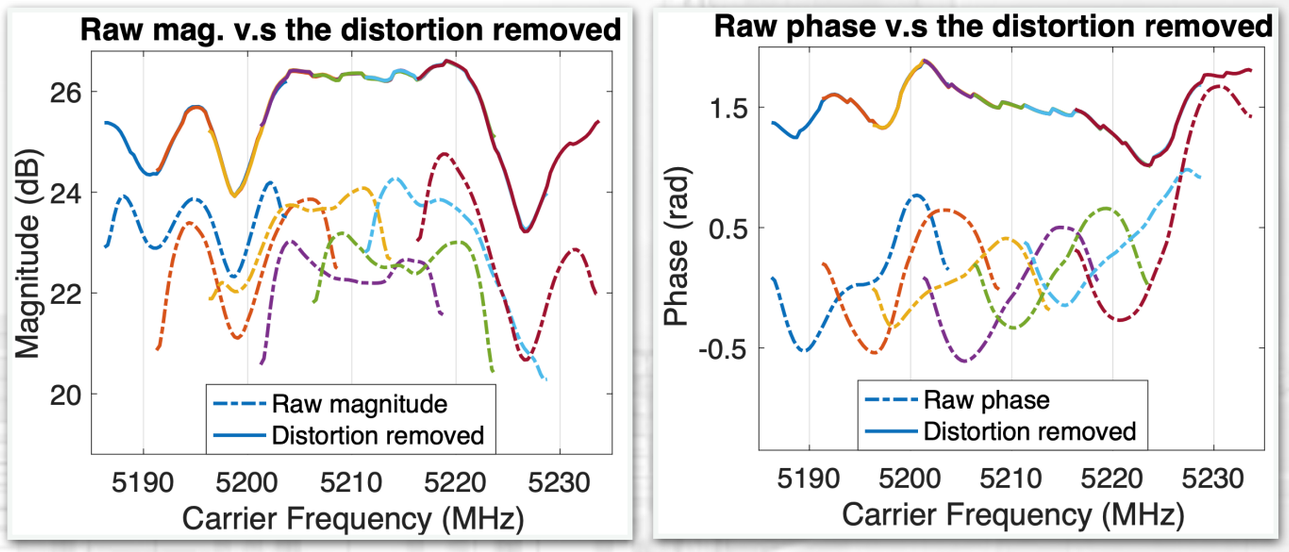
New Opportunities Provided by PicoScenes

In the hardware aspect…

- "Wi-Fi 6E" titled Wi-Fi sensing research
  - Ultra-bandwidth (UWB) on Wi-Fi
    - 160-MHz BW + 6-GHz band (7 x 160-MHz channels)
    - Challenges: Channel stitching?
  - 11ax-format, up to 8192 subcarriers per measurement
- Fully passive Wi-Fi sensing, enhanced by Wi-Fi 6E
  - A game changing paradigm for device-free Wi-Fi sensing
    - Transforming surrounding signals into excitations!
  - Challenges: transient signals (spatial and temporal)
- Wi-Fi sensing in unlicensed spectrum
  - Powered by QCA9300 and SDR
    - QCA9300: 2.2-2.9 and 4.4-6.1 GHz
    - USRP: 10-6000 MHz
    - HackRF One 10-7250 MHz
  - Much wider and clean spectrum



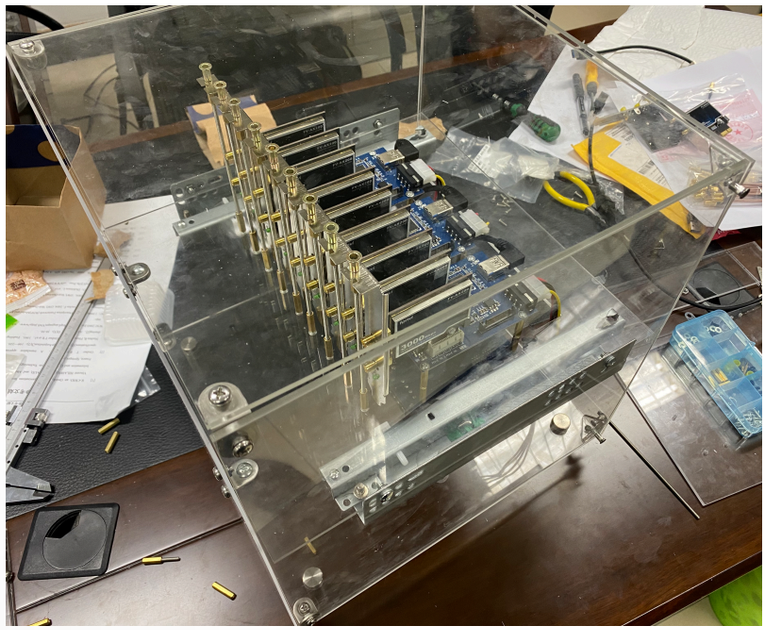
All 9 160-MHz BW channels in 5&6-GHz



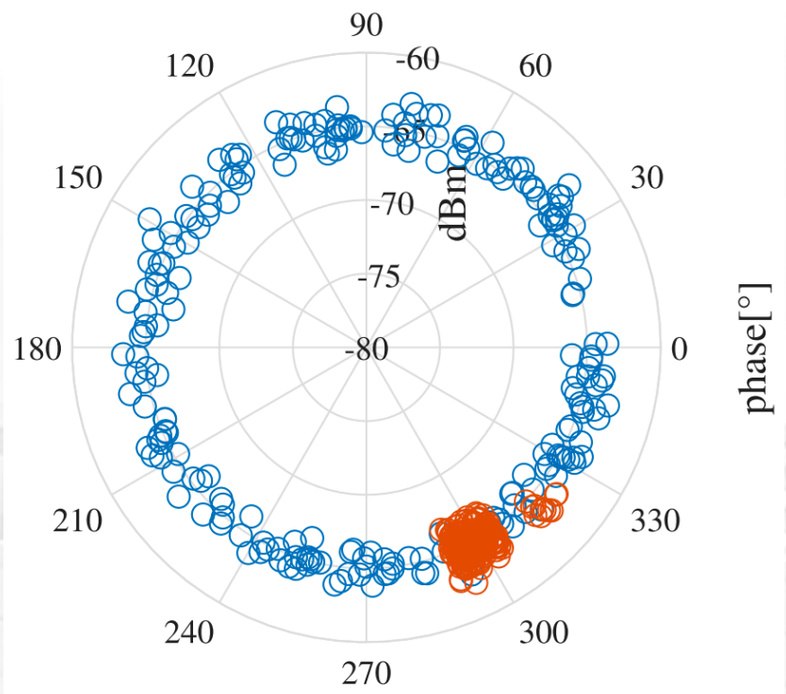
Spectrum stitching after CSI distortion removal

https://ps.zpj.io

New Opportunities Provided by PicoScenes



AX210-based Wi-Fi NIC array



Wire-free CFO estimation & elimination

In the software aspect…

- Concurrent multi-frontend CSI measurement
  - Unlock the path to phased array
- High-Wi-Fi baseband implementation & integration
  - Signal generation
  - Fully controlled Tx/Rx PHY
- Integrated packet injection w/ all-format support
  - Arbitrary content injection
    - Security research
- Precise Timing (both Tx and Rx)
  - precisely timed batch Tx (by SDR)
  - ADC-level Rx timestamp (AX200/210 and SDR)
- Much easier & larger-scale CSI measurement in spatial, temporal, and spectrum.
- Complex & interactive CSI measurement now possible!
  - Example code, EchoProbe initiator <-> responder

New Opportunities Provided by PicoScenes

In the hardware aspect…	In the software aspect…

• "Wi-Fi 6E" titled Wi-Fi sensing research
  • Ultra-bandwidth (UWB) on Wi-Fi
  • 160-MHz BW + 6-GHz band (7 x 160-MHz channels)
  • Challenges: Channel stitching?
  • 11ax-format, up to 8192 subcarriers per measurement
• Fully passive Wi-Fi sensing, enhanced by Wi-Fi 6E
  • A game changing paradigm for device-free Wi-Fi sensing
  • Transforming surrounding signals into excitations!
  • Challenges: transient signals (spatial and temporal)
• Wi-Fi sensing in unlicensed spectrum
  • Powered by QCA9300 and SDR
  • QCA9300: 2.2-2.9 and 4.4-6.1 GHz
  • USRP: 10-6000 MHz
  • HackRF One 10-7250 MHz
  • Much wider and clean spectrum
	• Concurrent multi-frontend CSI measurement
	  • Unlock the path to phased array
	• High-Wi-Fi baseband implementation & integration
	  • Signal generation
	  • Fully controlled Tx/Rx PHY
	• Integrated packet injection w/ all-format support
	  • Arbitrary content injection
	  • Security research
	• Precise Timing (both Tx and Rx)
	  • precisely timed batch Tx (by SDR)
	  • ADC-level Rx timestamp (AX200/210 and SDR)
	• Much easier & larger-scale CSI measurement in spatial, temporal, and spectrum.
	• Complex & interactive CSI measurement now possible!
	  • Example code, EchoProbe initiator <-> responder

Resources

- “Eliminating the Barriers: Demystifying Wi-Fi Baseband Design and Introducing the PicoScenes Wi-Fi Sensing Platform”, Zhiping Jiang, et.al, *IEEE Internet of Things Journal,* 2021, https://doi.org/10.1109/JIOT.2021.3104666
- PicoScenes software documents: https://ps.zpj.io
- PicoScenes Issue Tracker (all issues go here!) https://gitlab.com/wifisensing/picoscenes-issue-tracker
- **Opensource subprojects**
  - RXS-Parsing-Core Library https://gitlab.com/wifisensing/rxs_parsing_core
  - PicoScenes Plugin Development Kit (PDK) https://gitlab.com/wifisensing/PicoScenes-PDK
  - PicoScenes MATLAB Toolbox Core (PMT-Core) https://gitlab.com/wifisensing/PicoScenes-MATLAB-Toolbox-Core
  - PicoScenes Python Toolbox Core (PPT-Core) https://gitlab.com/wifisensing/PicoScenes-Python-Toolbox

Thanks!

https://ps.zpj.io

Extra slides

https://ps.zpj.io

Wi-Fi Sensing CSI Tools (pre PicoScenes)

CSI Tools based on COTS Wi-Fi NICs

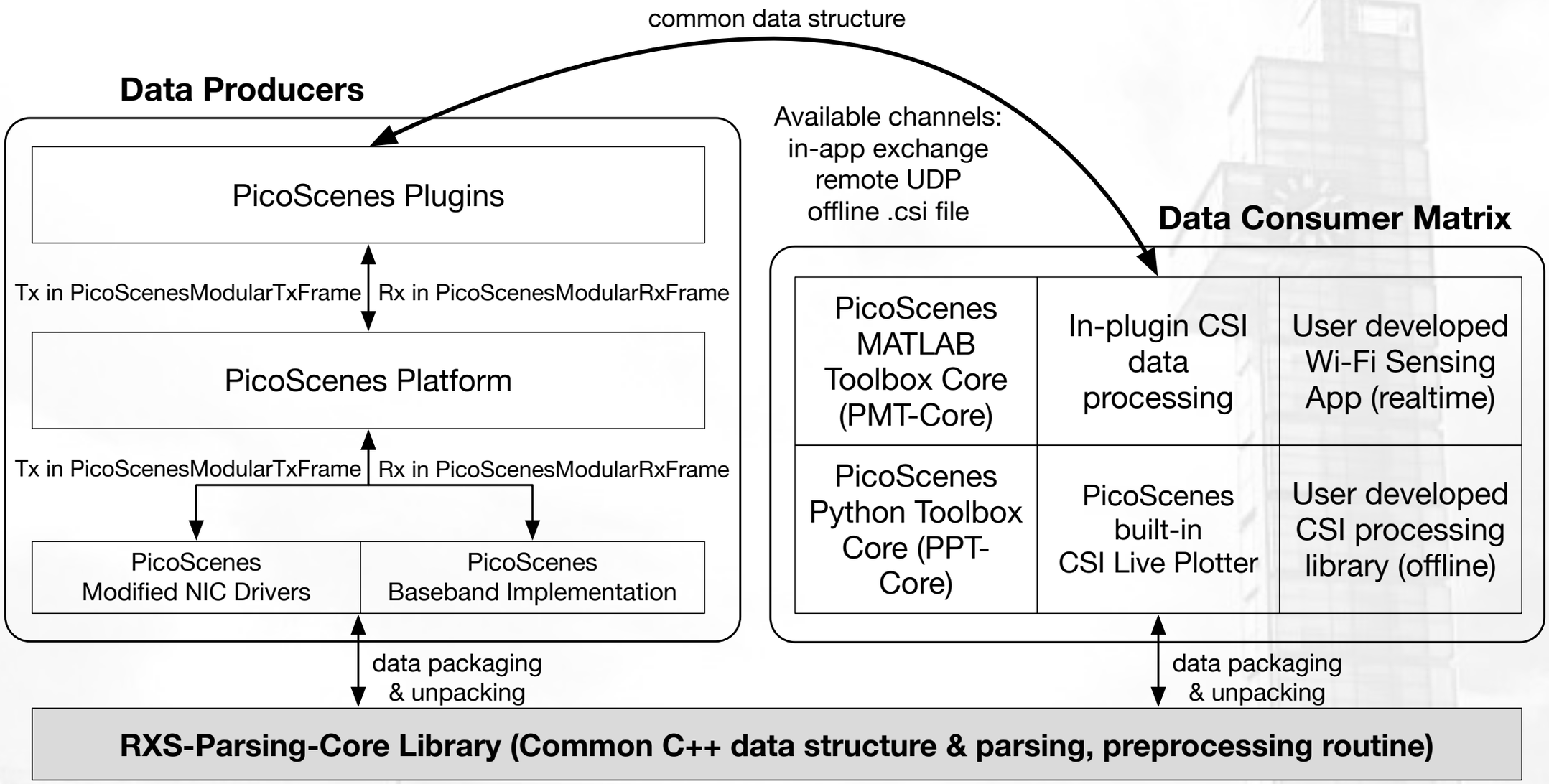
Hardware	Release Year	Released Tool	Highest Standard	Highlight Features	Citation Count
Intel 5300 NIC	2011	Intel 5300 CSI Tool	802.11n	**First CSI tool**, 3T3R, 8-bit ADC	**>1000**
Atheros 9300 NIC	2015	Atheros CSI Tool	802.11n	10-bit ADC, 3T3R, full subcarriers (56/114)	**>100**
Broadcom 43xx Chip	2019	Nexmon CSI Extractor	802.11ac	**First 11ac CSI tool**, up to 4T4R on modified router	<100
ESP 32	2020	Wi-ESP	802.11n	**Ultra low-cost SoC, standalone,** 1T1R	>10

CSI Tools based on General SDR / ZynQ

Hardware	Release Year	Baseband Implementation	Highlight Features	Cost	Citation Count
USRP	2008	none is public available	**widely accepted by academic,** easy for synchronization	$1100 - 25000	**>100**
WARP v3	2012	PHY/MAC on ZynQ	**Full PHY + MAC layer**	Enquiry	>10
Open Wi-Fi	2019	PHY/MAC on ZynQ	**Full opensource**, FPGA-based Wi-Fi NIC	$500 - 10000	~10

PicoScenes Data Production & Consumption

CSI data production & consumption in PicoScenes ecosystem



RXS-Parsing-Core Library https://gitlab.com/wifisensing/rxs_parsing_core
PicoScenes Plugin Development Kit (PDK) https://gitlab.com/wifisensing/PicoScenes-PDK
PicoScenes MATLAB Toolbox Core (PMT-Core) https://gitlab.com/wifisensing/PicoScenes-MATLAB-Toolbox-Core
PicoScenes Python Toolbox Core (PPT-Core) https://gitlab.com/wifisensing/PicoScenes-Python-Toolbox