Networking on WARP

Chris Hunter & Patrick Murphy Rice University

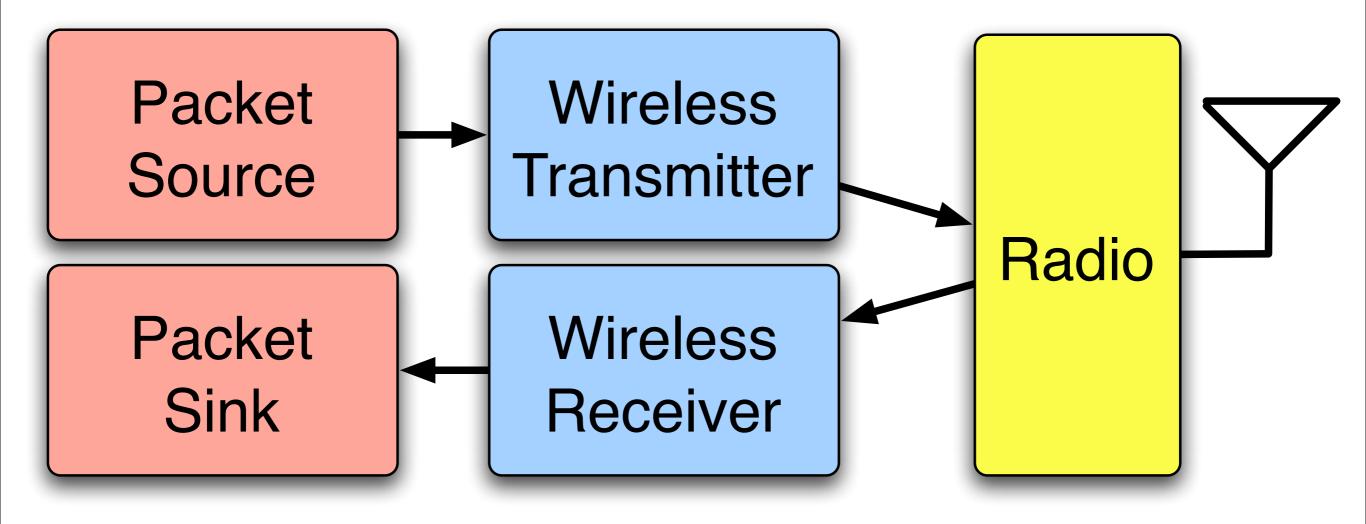
> WARP Workshop December 2, 2007



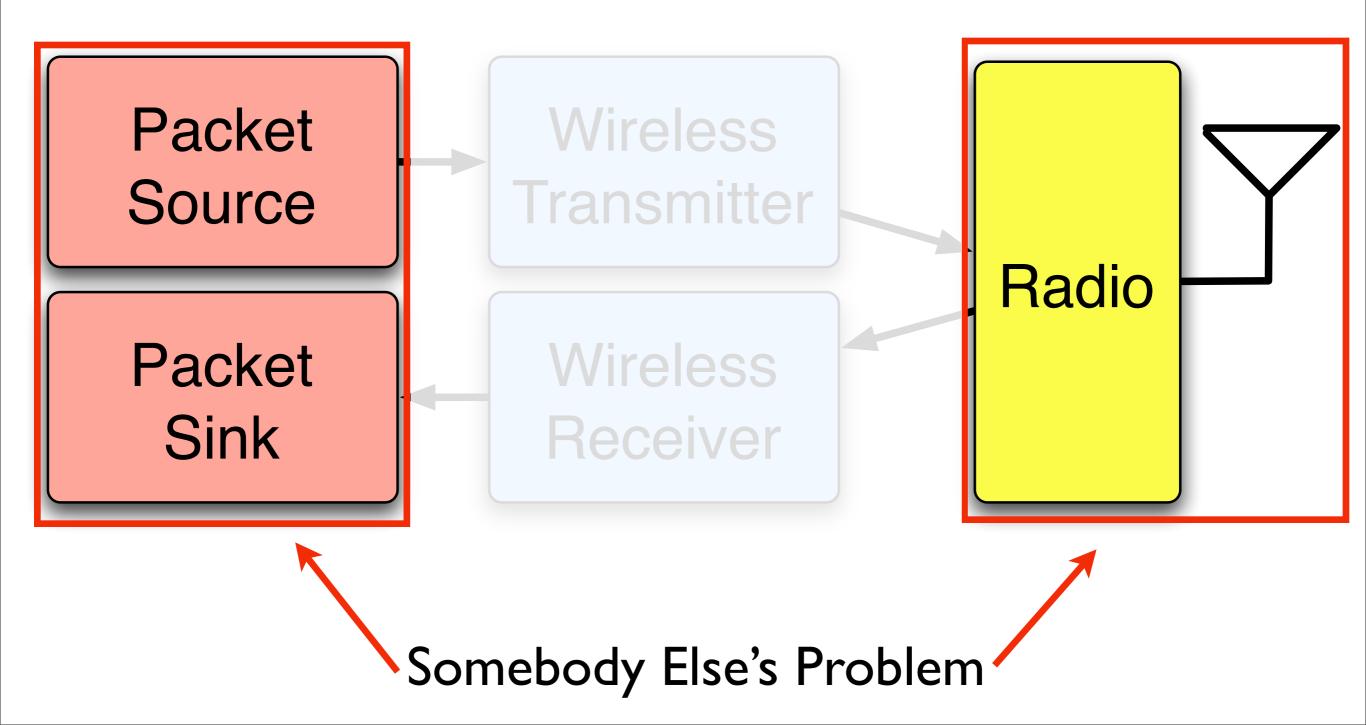
Today's Agenda

- Questions from yesterday?
- Networking on WARP lecture
- Lab 4 Simple "MAC" layer
- Lab 5 Unidirectional MAC
- Lab 6 Channel-hopping MAC
- Workshop wrap-up

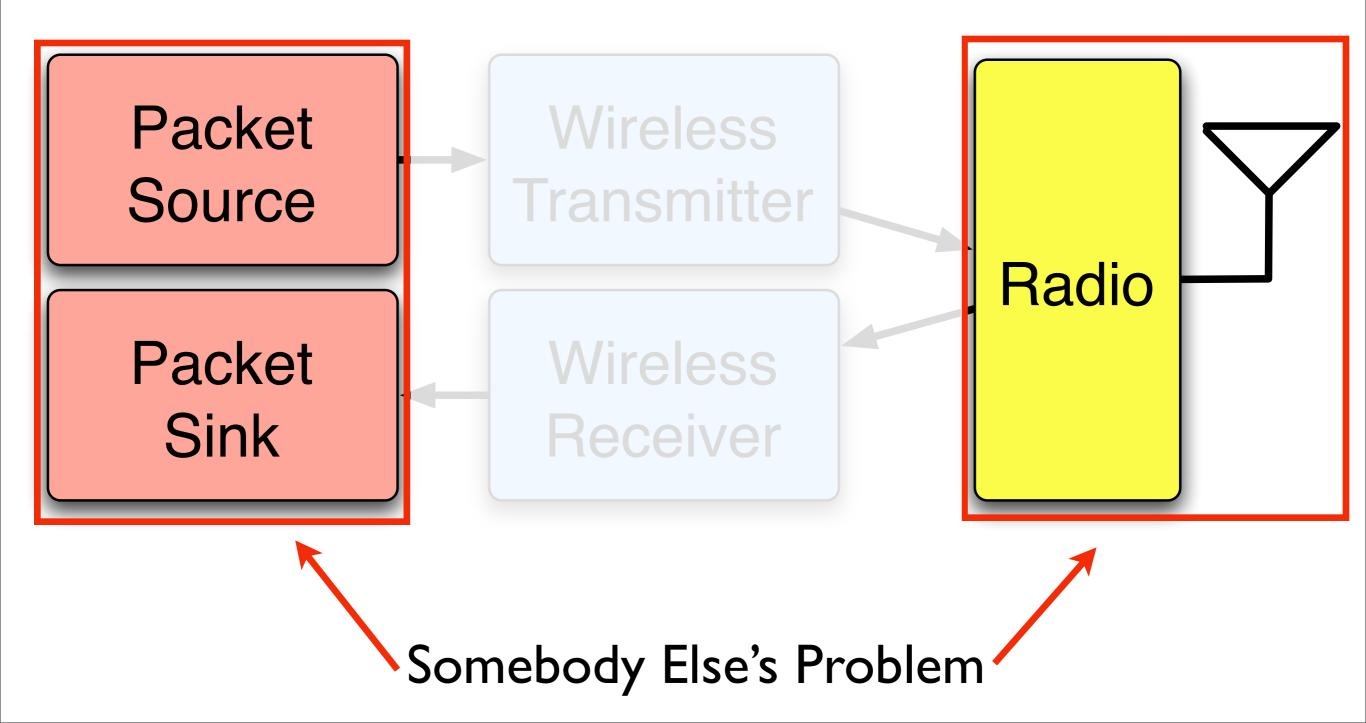
Physical Layer Basics



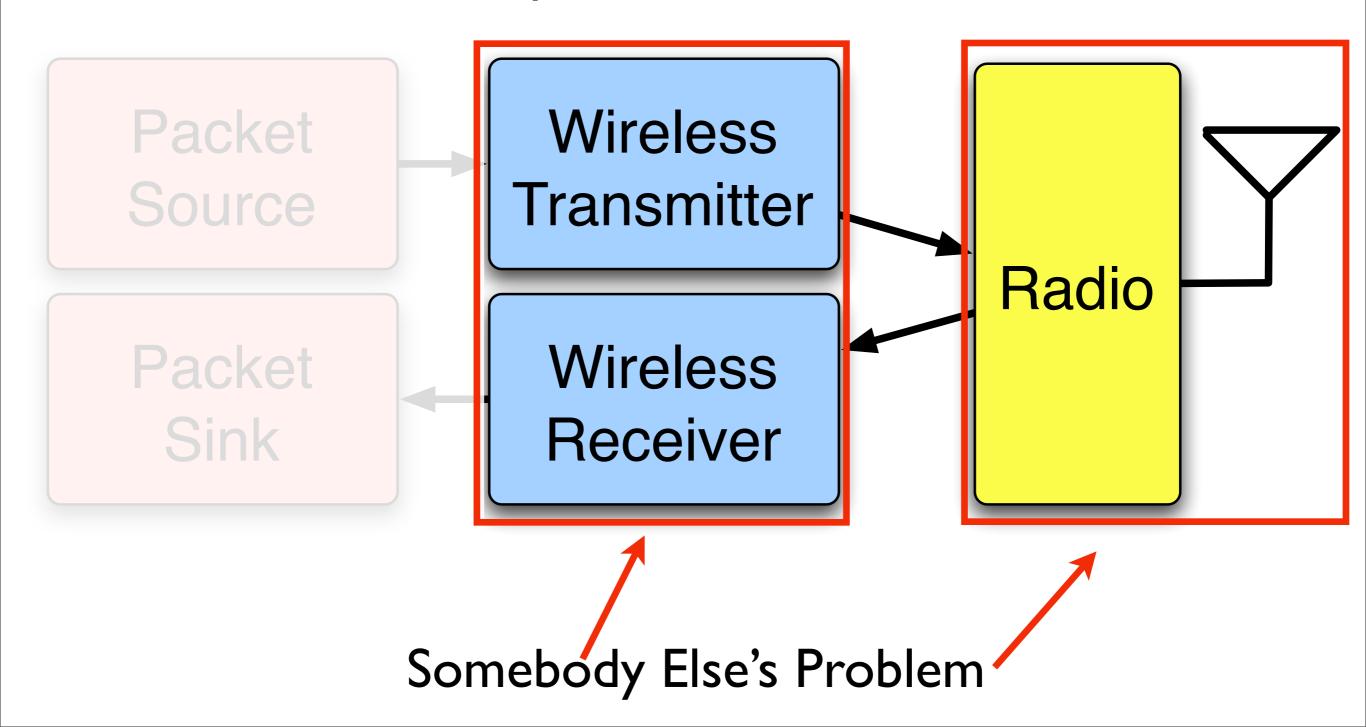
Physical Layer Basics



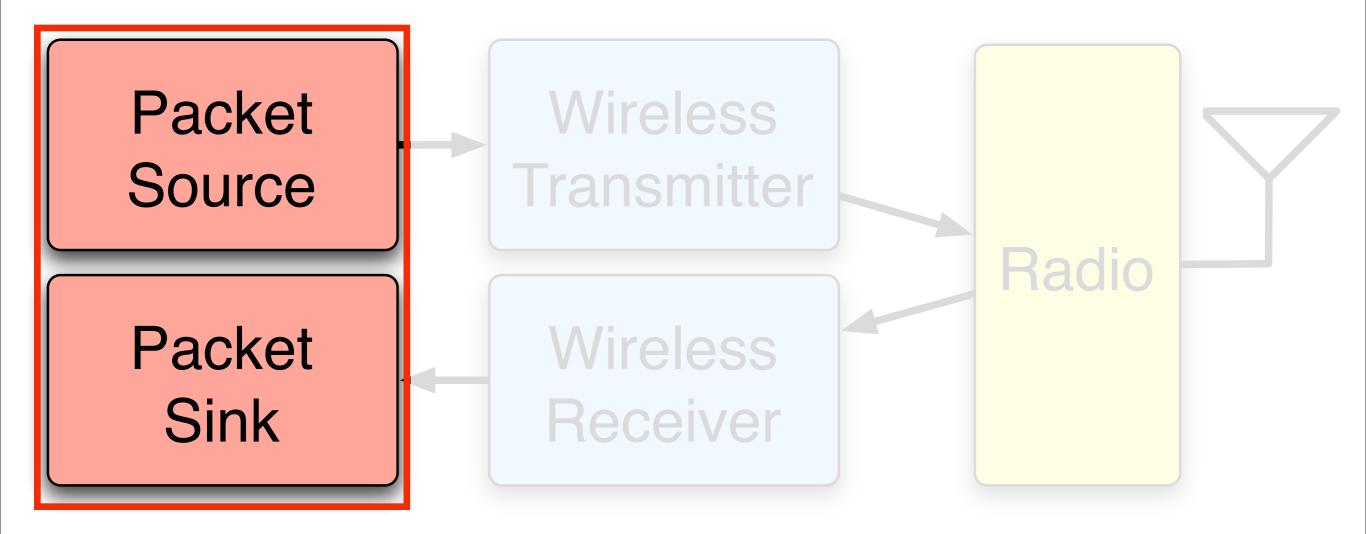
Network Layer Basics



Network Layer Basics

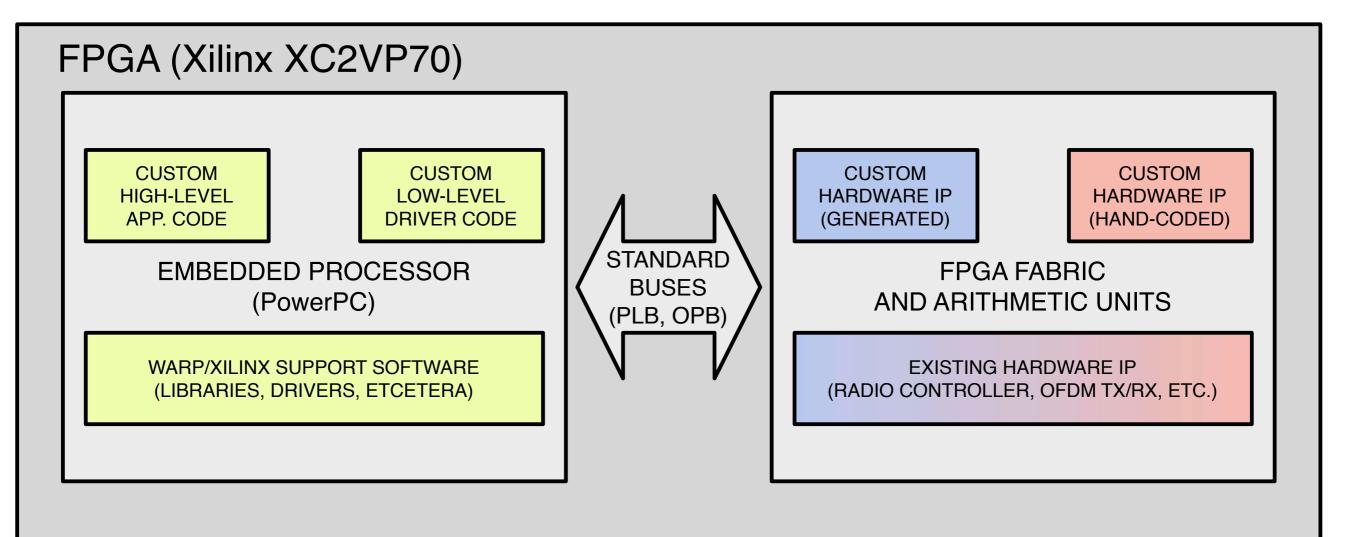


Network Layer Basics



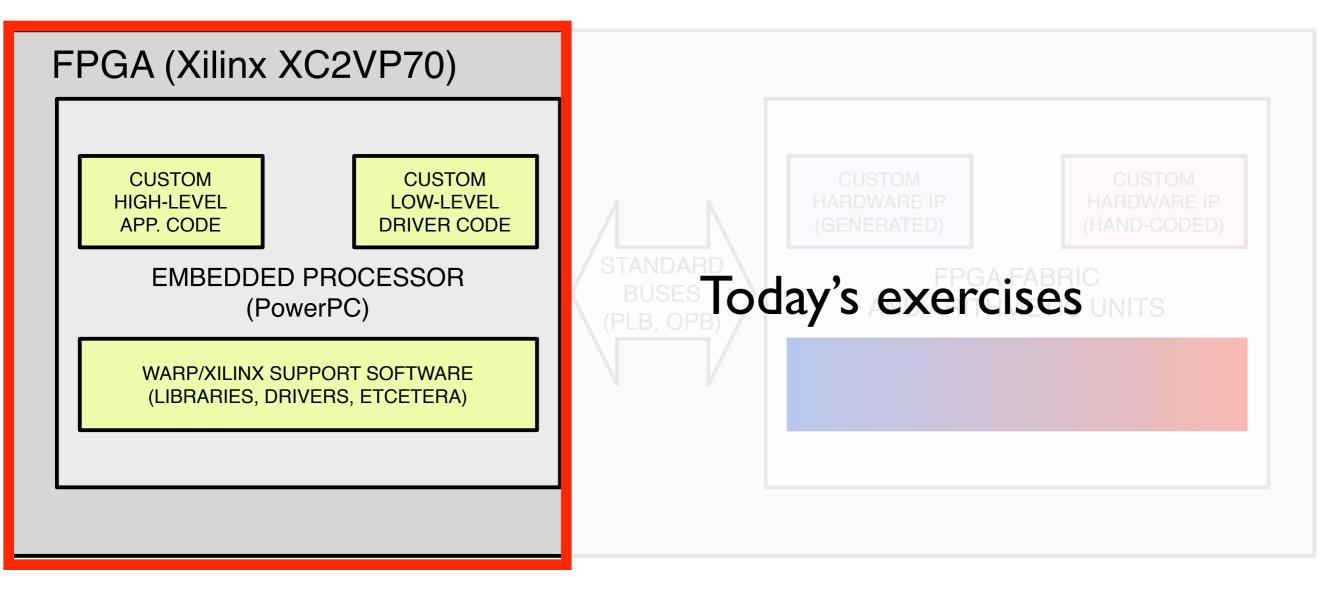
Targeting WARP Hardware

(Understanding the Development Environment)

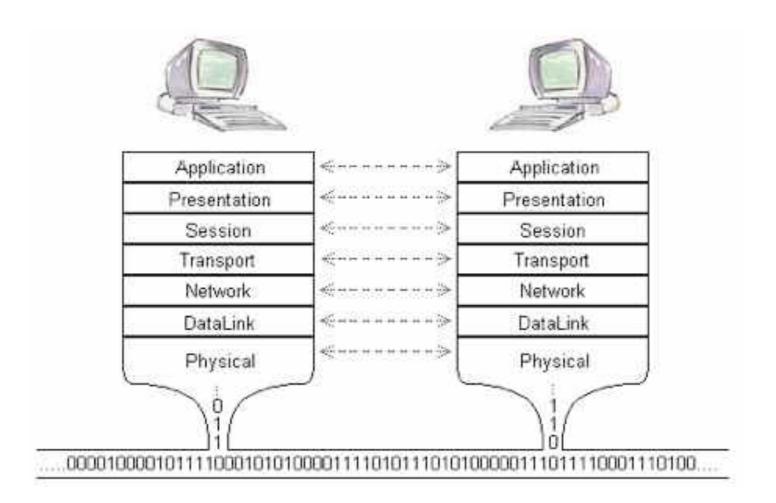


Targeting WARP Hardware

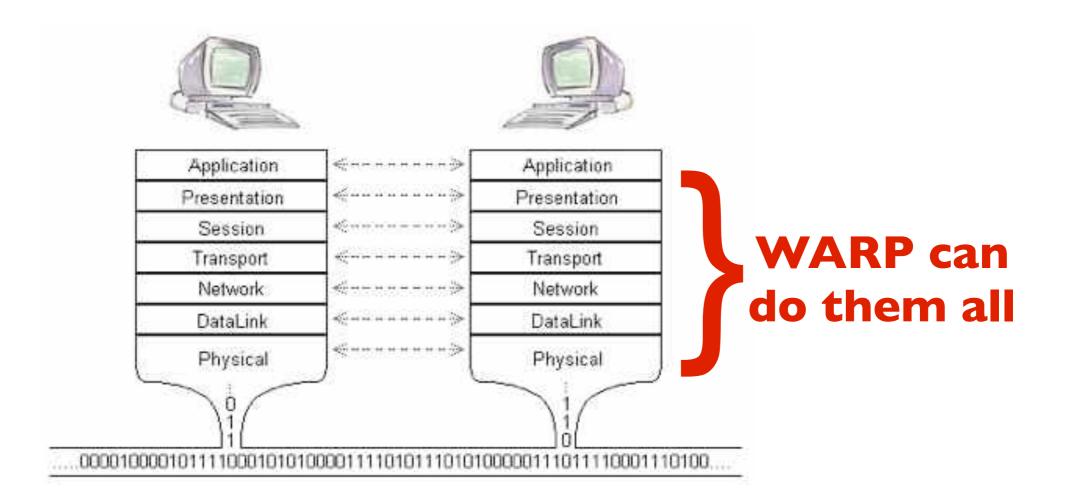
(Understanding the Development Environment)



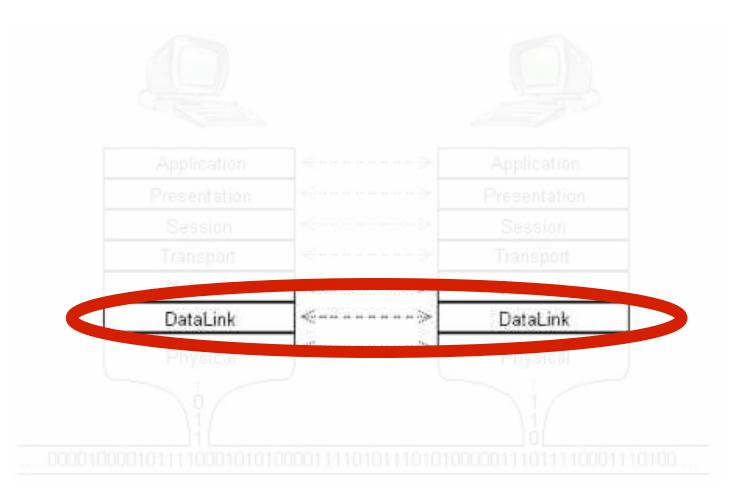
Some Perspective - The OSI Model



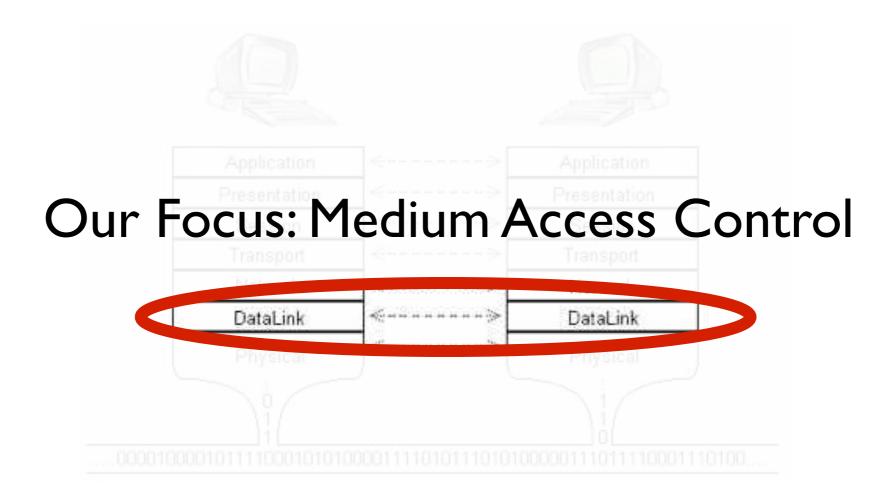
Some Perspective - The OSI Model



The OSI Model



The OSI Model



The OSI Model

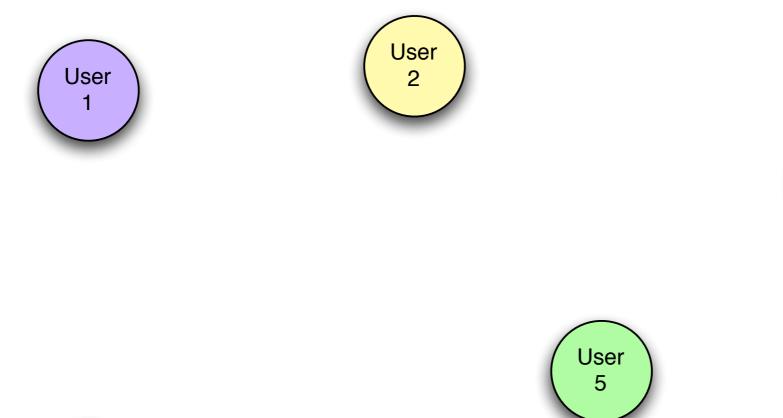


- Many interesting research problems: mesh networks, adaptive rate, cross-layer gains, etc.
- All commercial 802.11 chipsets are closed

Outline

- Overview of Medium Access Control
- Design Realization
- WARPMAC Framework
- Detailed Example
- Lab Exercises

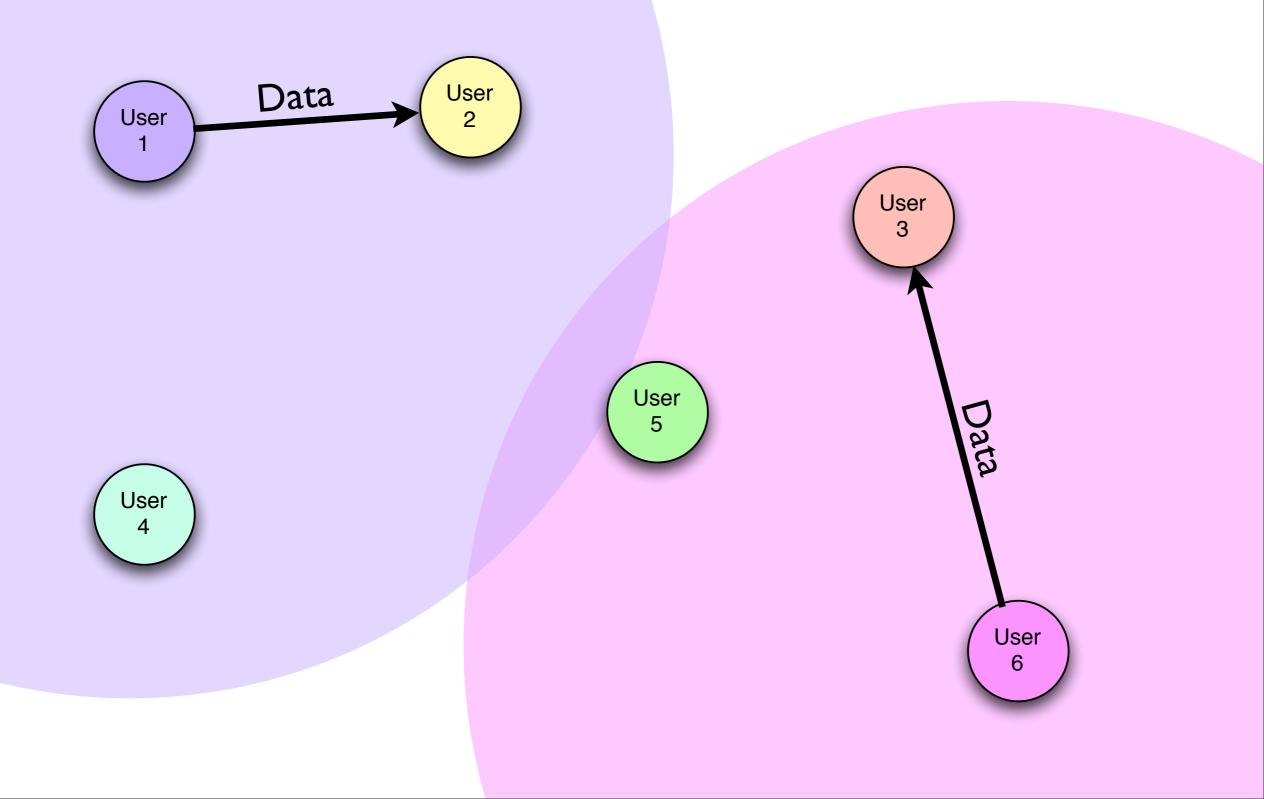
Medium Access Control Overview

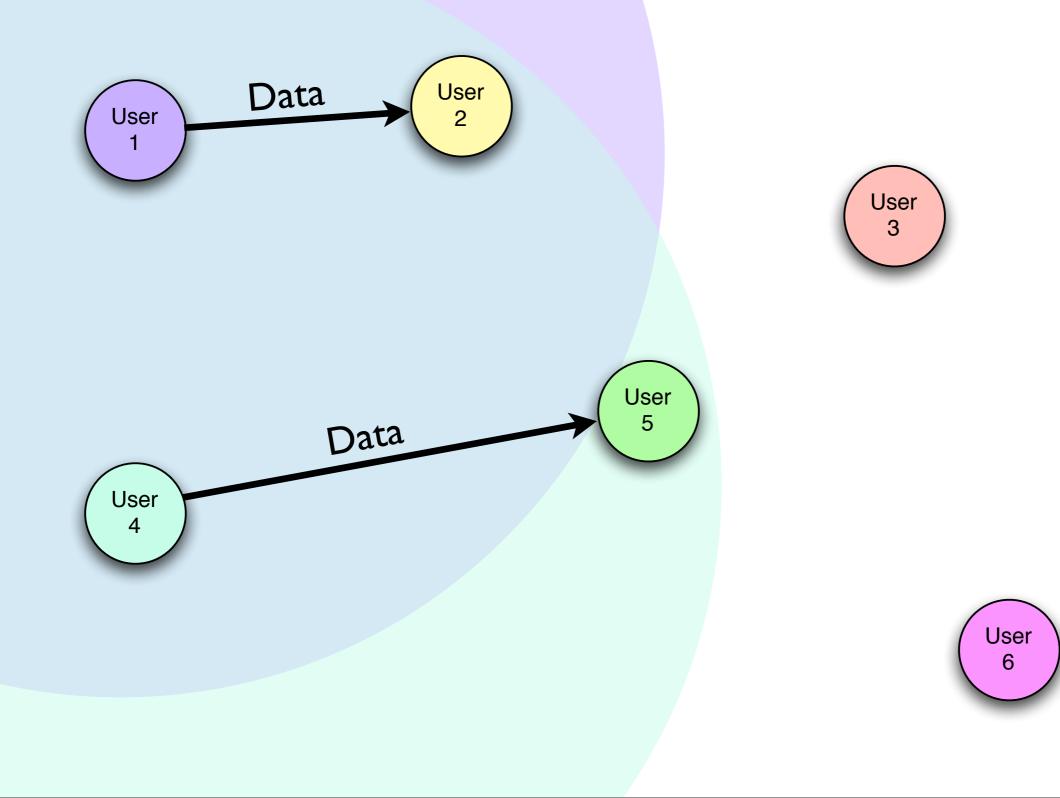


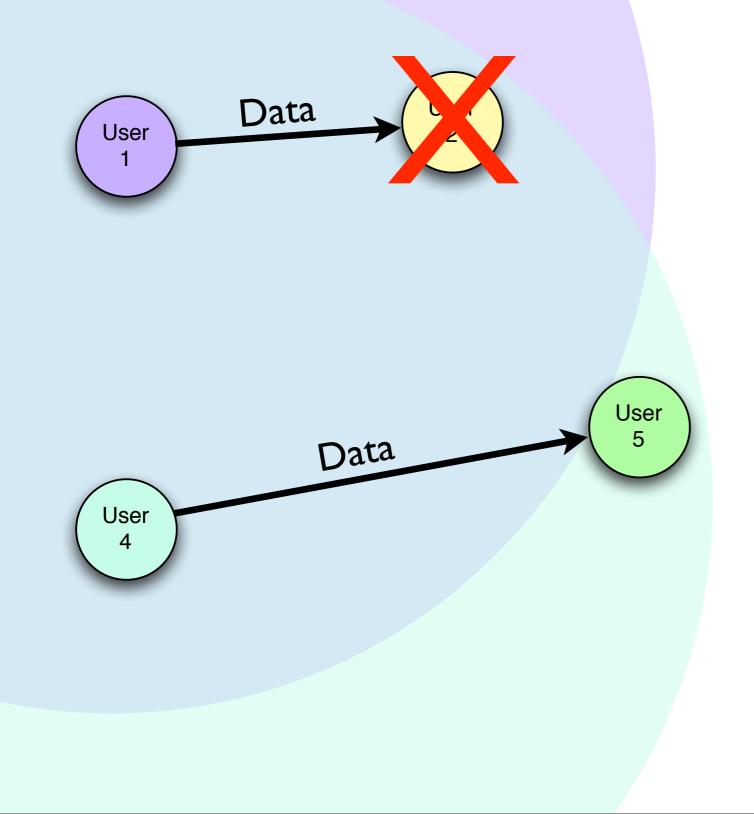


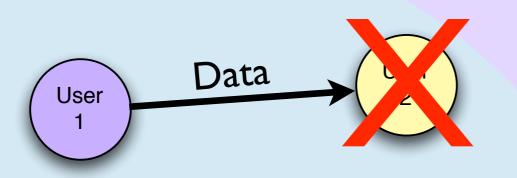


User 3

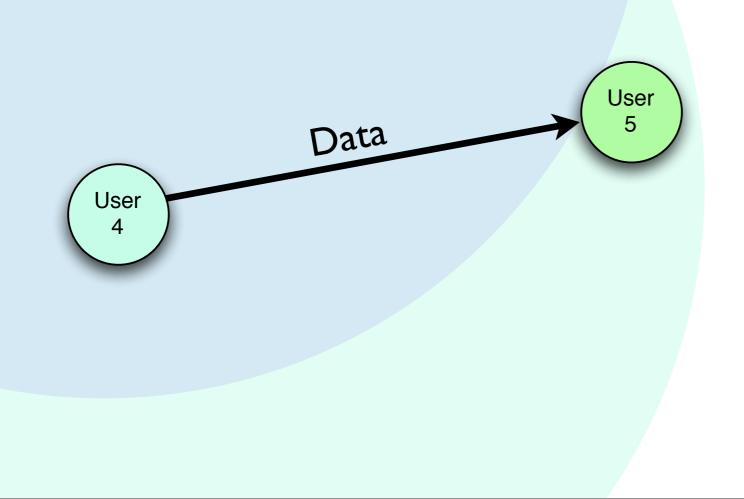








Received a jumbled packet... infer a packet collision



Received a jumbled packet... infer a packet collision

User 5

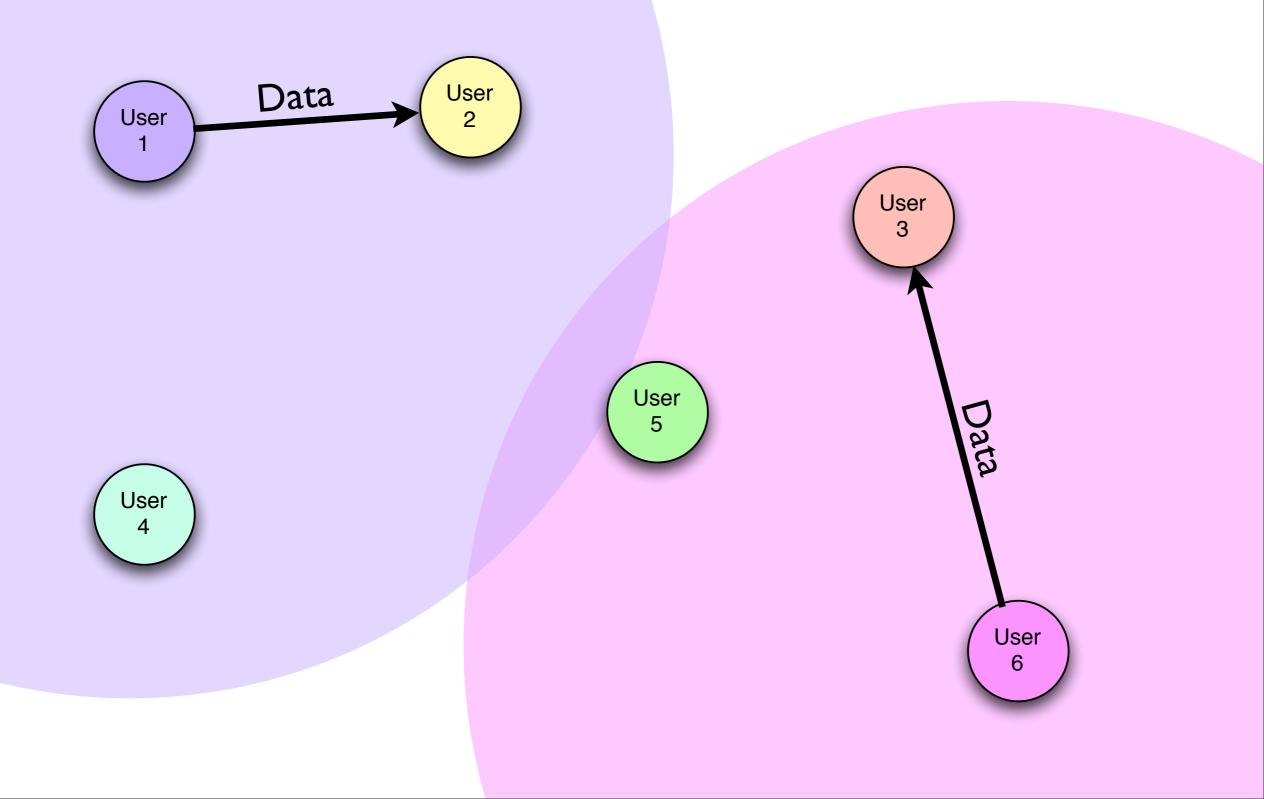
User 4

User

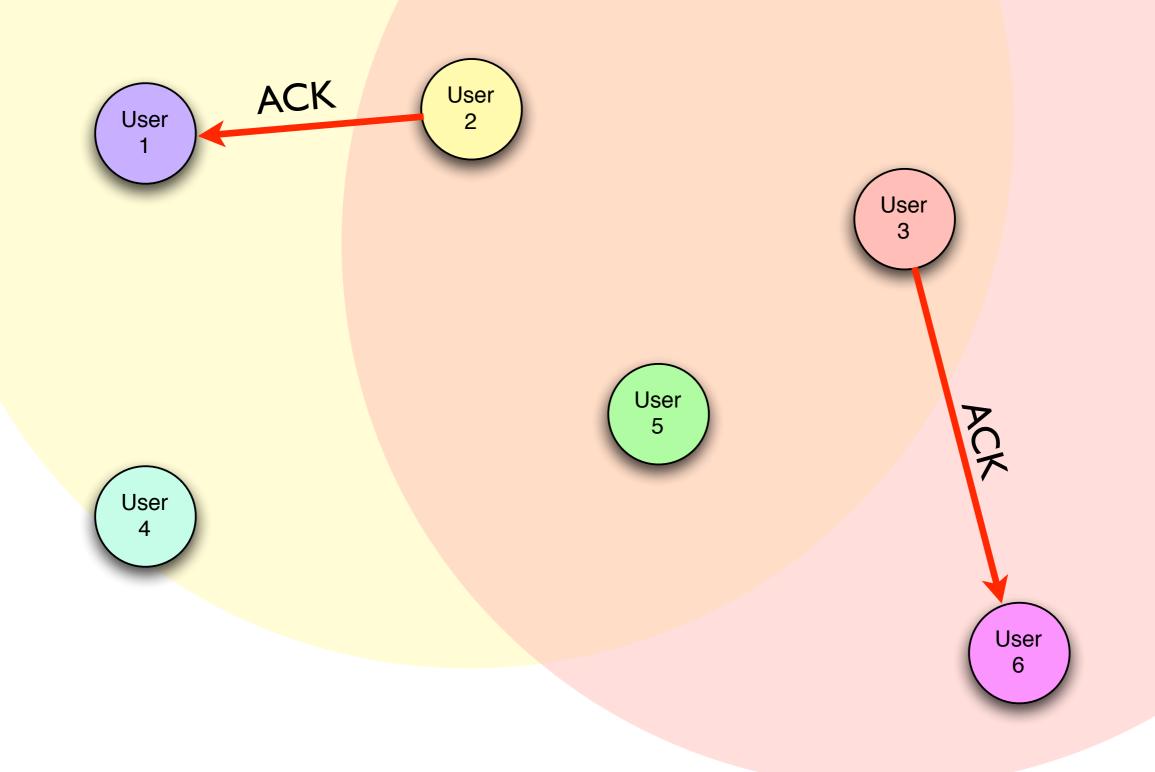
Data

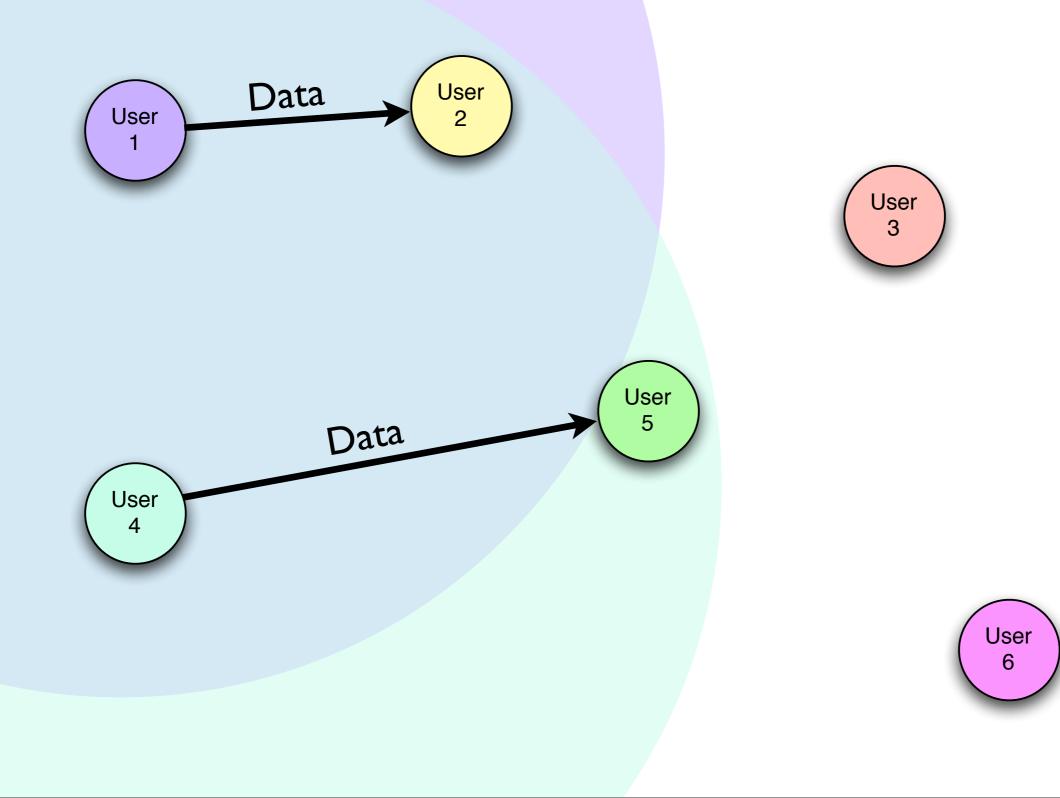
Data

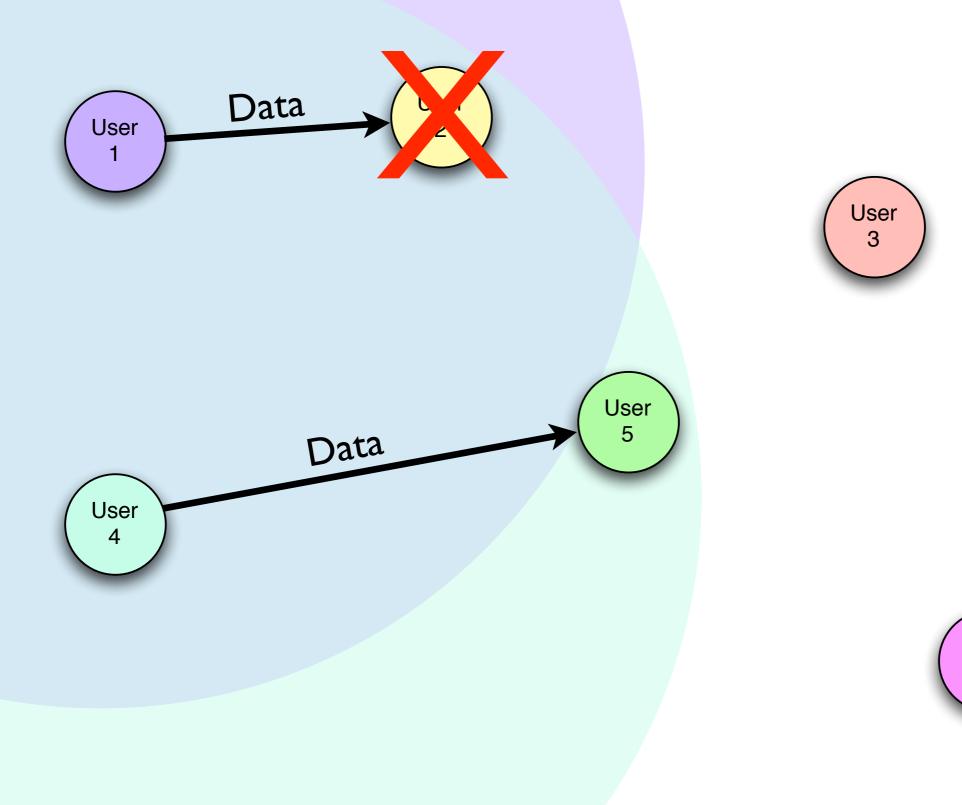
What if we ACK every transmit, and retransmit when we receive no ACK?



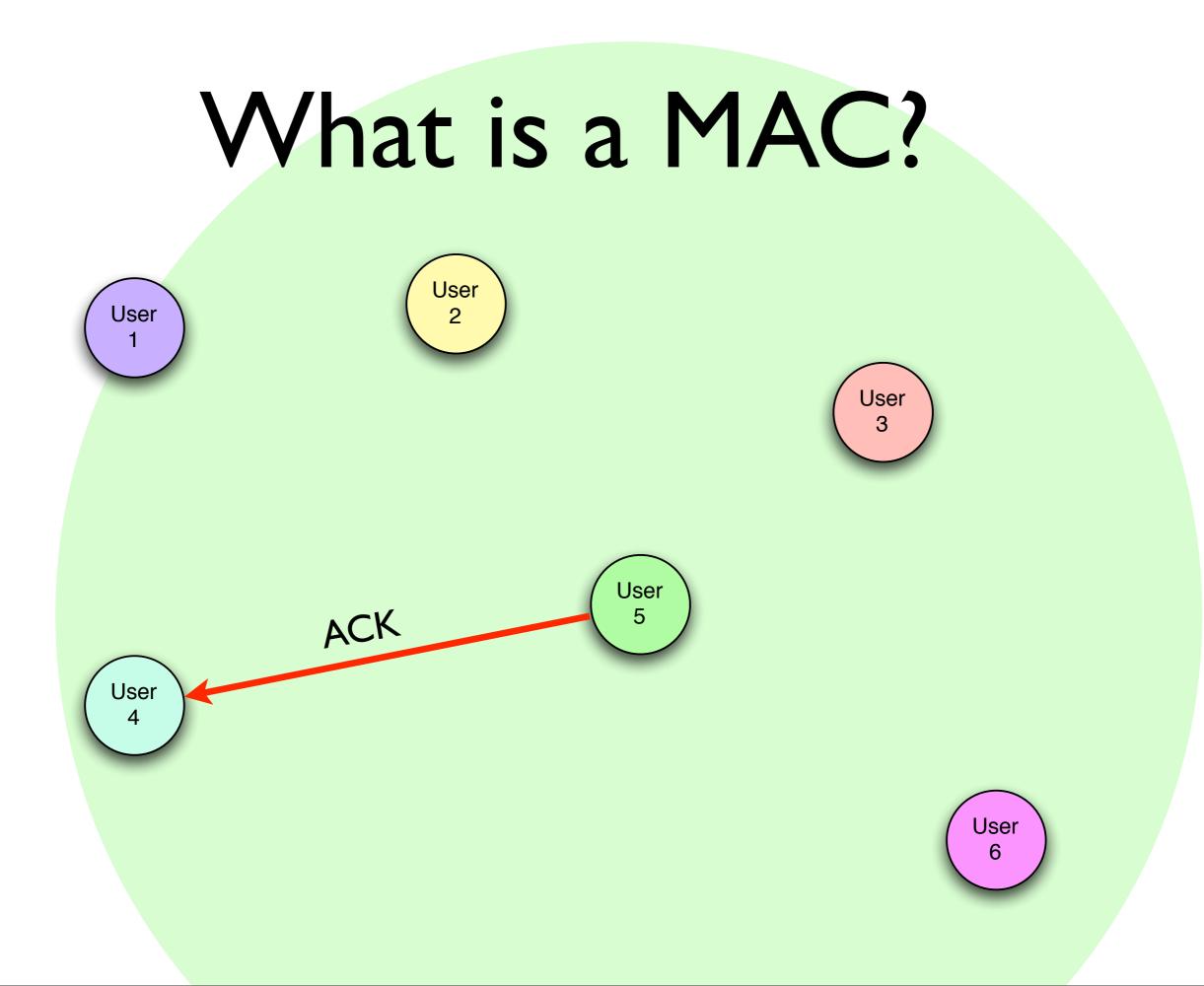


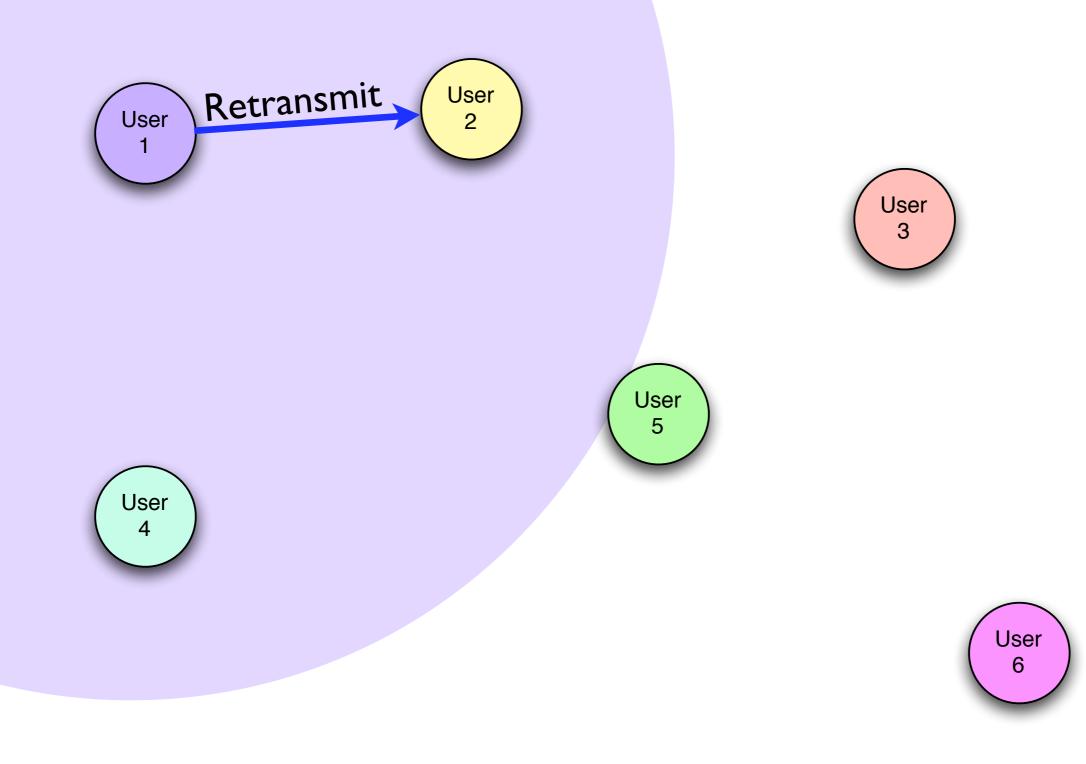


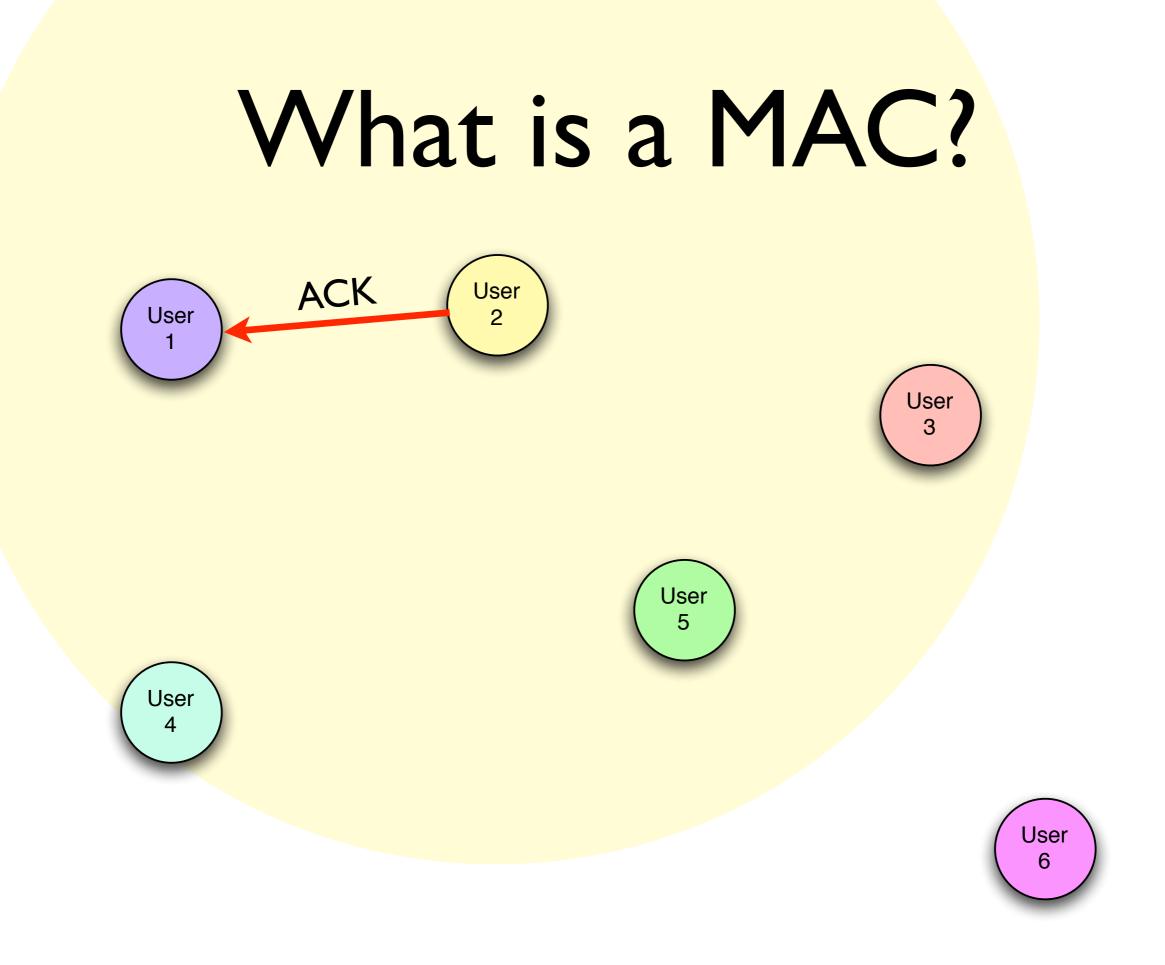




User 6





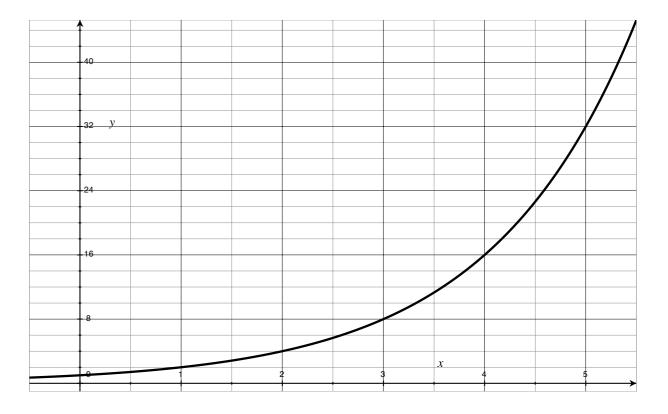


Random Backoffs

• **PROBLEM:**

Retransmissions can collide *ad infinitum!*

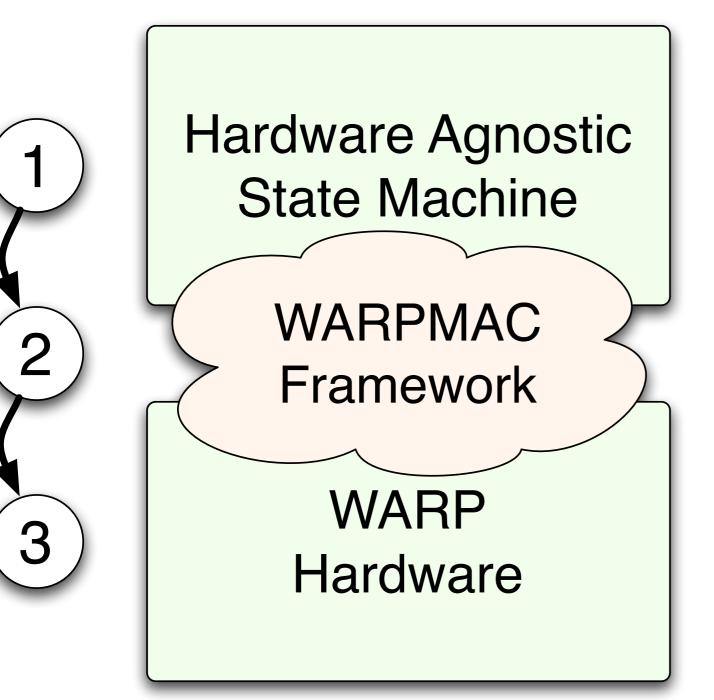
• **SOLUTION:** Wait a random amount of time before a retransmit



Contention Window increases over time

Other Important Details

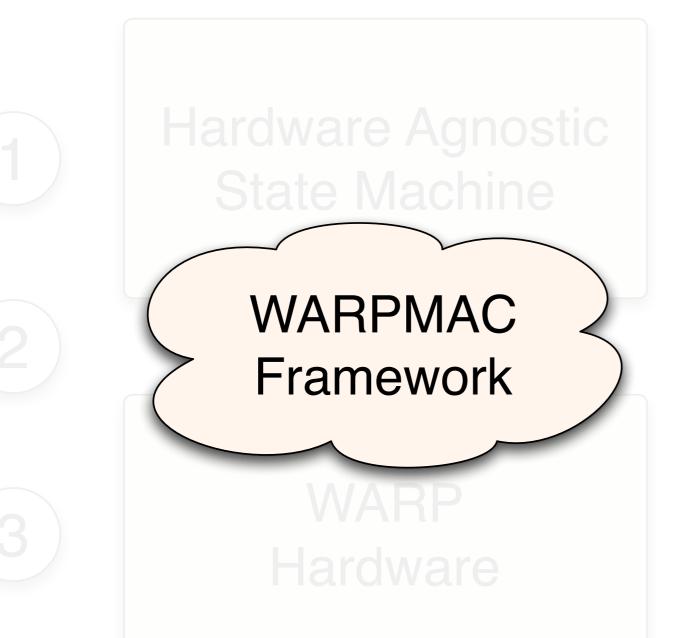
- Carrier Sense Multiple Access (CSMA)
 - Listen to the medium before sending
- Request to Send / Clear to Send (RTS/CTS)
 - "Reserve" the medium with a short packet before sending a long one



- Program high-level MAC behavior independent of hardware
- Use the WARPMAC
 framework to
 stitch the MAC to
 hardware



- "Driver" analogy is not entirely accurate
- No way to "lock" the framework and have it support all possible future MAC layers

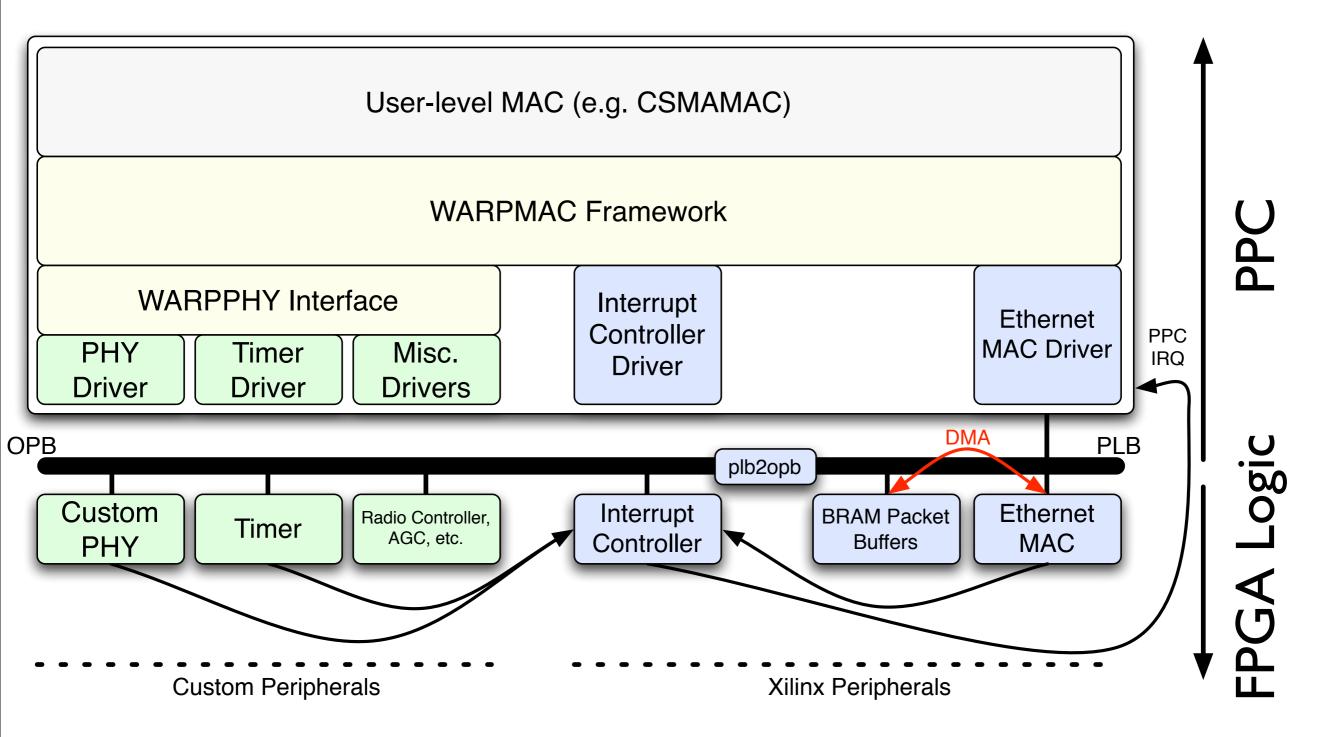


- "Driver" analogy is not entirely accurate
- No way to "lock" the framework and have it support all possible future MAC layers

Solution: WARPMAC must grow with new algorithms

WARPMAC Framework

System Diagram



WARPMAC

WARPPHY

WARPMAC

WARPPHY

WARPMAC

WARPPHY

PHY Driver:

- Configure very low-level parameters
 - Correlation thresholds
 - FFT scaling parameters
 - Filter coefficients
 - Etc.

User Code WARPMAC WARPPHY Drivers

WARPMAC

WARPPHY

WARPMAC

WARPPHY

Radio Controller Driver:

- Set center frequency
- Switch from Rx to Tx mode and vice versa

User Code

WARPMAC

WARPPHY

WARPMAC

WARPPHY

WARPMAC

WARPPHY

PHY Control:

- Provides control over PHY commonalities
 - General initialization command
 - Configure constellation order
 - "Start" and "Stop" the PHY



WARPMAC

WARPPHY

WARPMAC

WARPPHY

Mostly PHY agnostic

User Code

WARPMAC

WARPPHY

Drivers

Completely PHY dependent

WARPMAC

WARPPHY

MAC Control:

- Provides control over MAC commonalities
 - Timers for timeouts, backoffs, etc.
 - Carrier-sensing functions
 - Register user callbacks to ISRs
 - Etc.

User Code

WARPMAC

WARPPHY

WARPMAC

WARPPHY

WARPMAC

WARPPHY

User-level MAC Algorithms:

- High-level MAC algorithms
- Some examples so far:
 - Aloha
 - Carrier-sensing MAC
 - Opportunistic Auto-Rate (OAR)
 - MAC Workshop Exercises

User Code WARPMAC WARPPHY Drivers

WARPMAC

WARPPHY

- Simple, usable MAC
- Serves as a foundation for a large class of other random access protocols
- The algorithm is simple:

- Simple, usable MAC
- Serves as a foundation for a large class of other random access protocols
- The algorithm is simple:

Packet to send? Just send it

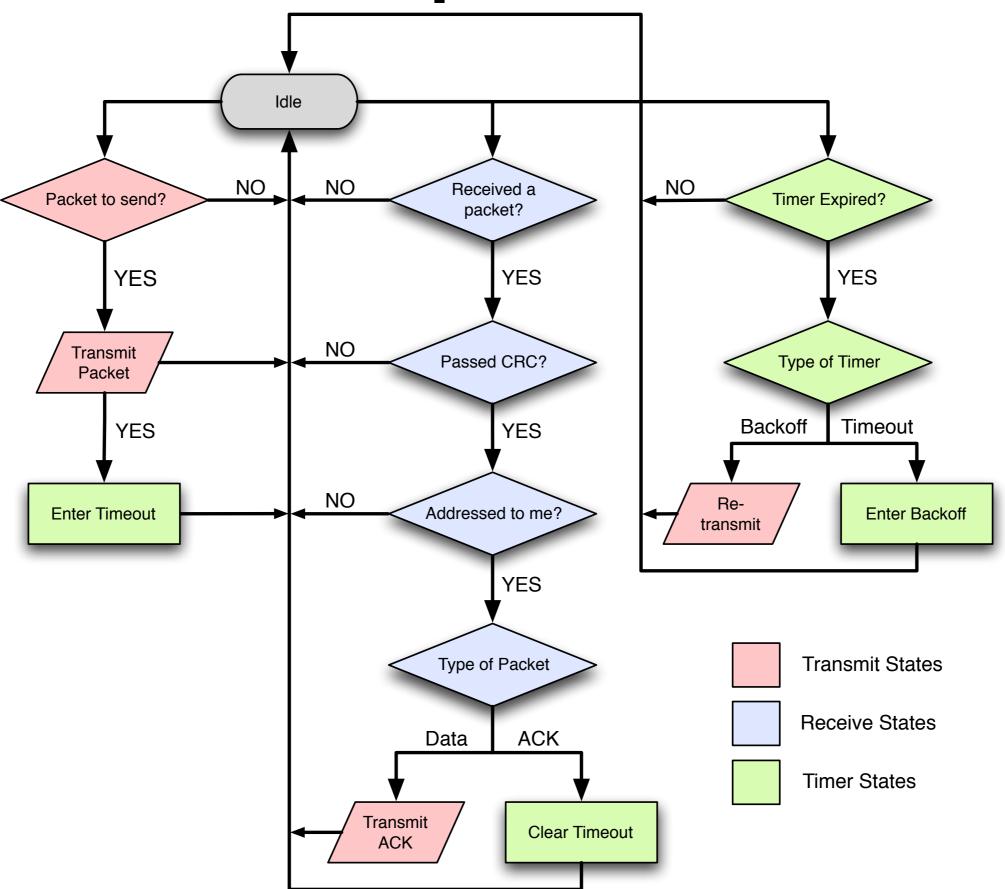
- Simple, usable MAC
- Serves as a foundation for a large class of other random access protocols
- The algorithm is simple:
 - Packet to send? Just send it

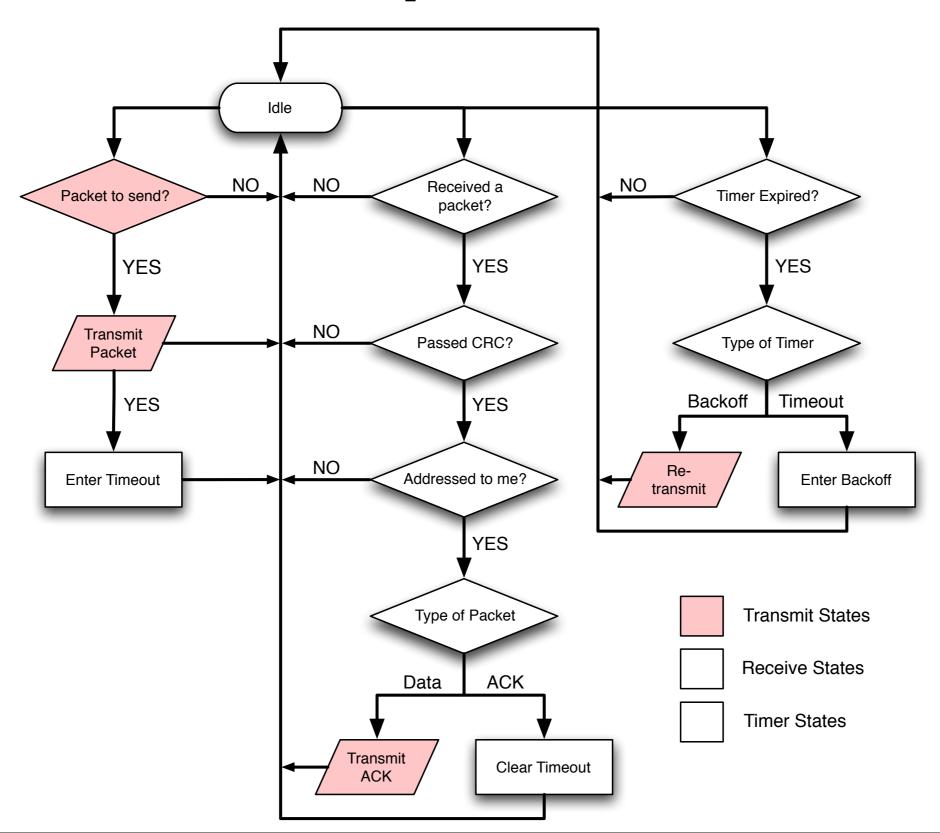
Received a packet? Send an ACK

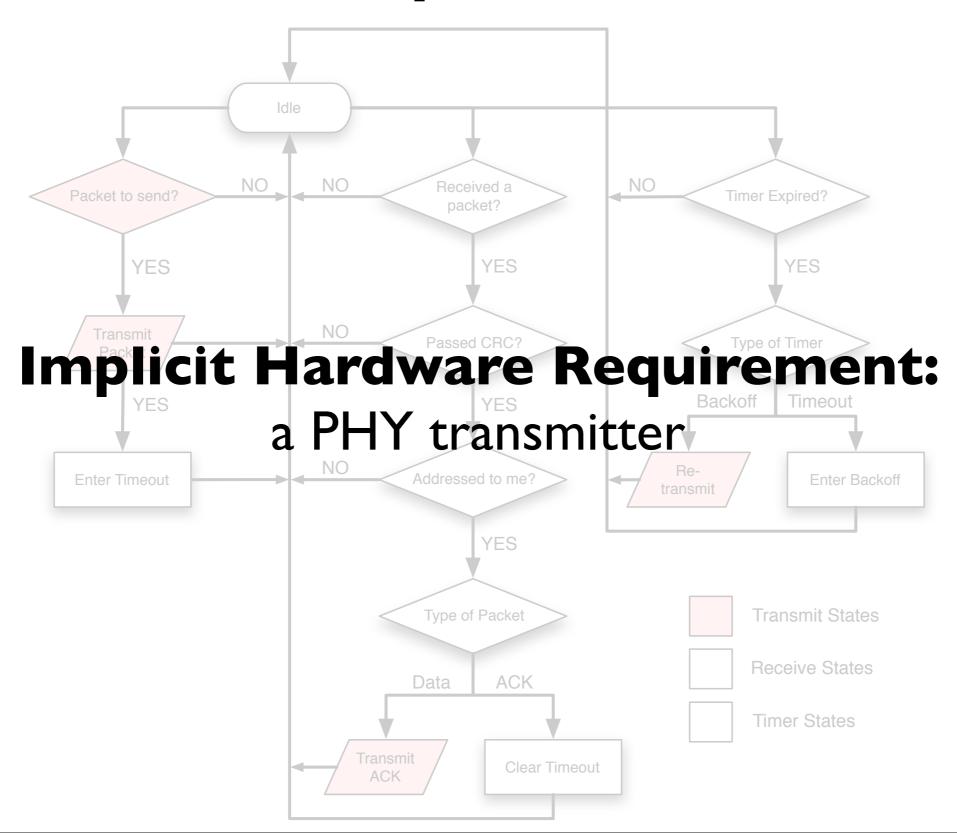
- Simple, usable MAC
- Serves as a foundation for a large class of other random access protocols
- The algorithm is simple:
 - Packet to send? Just send it

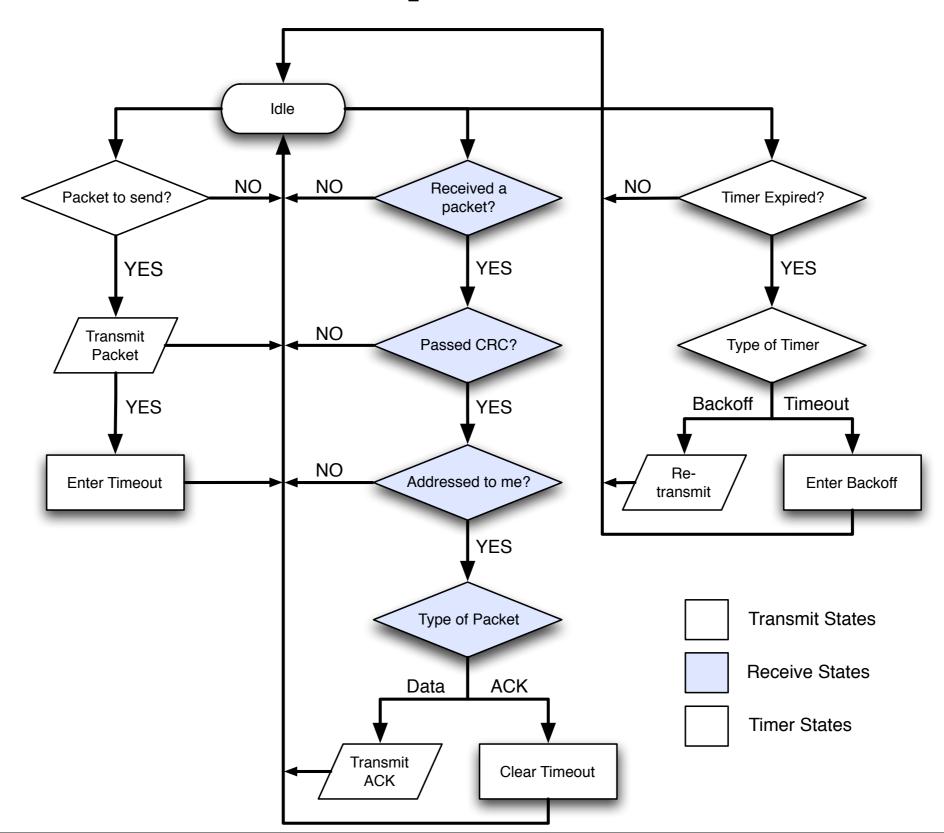
Received a packet? Send an ACK

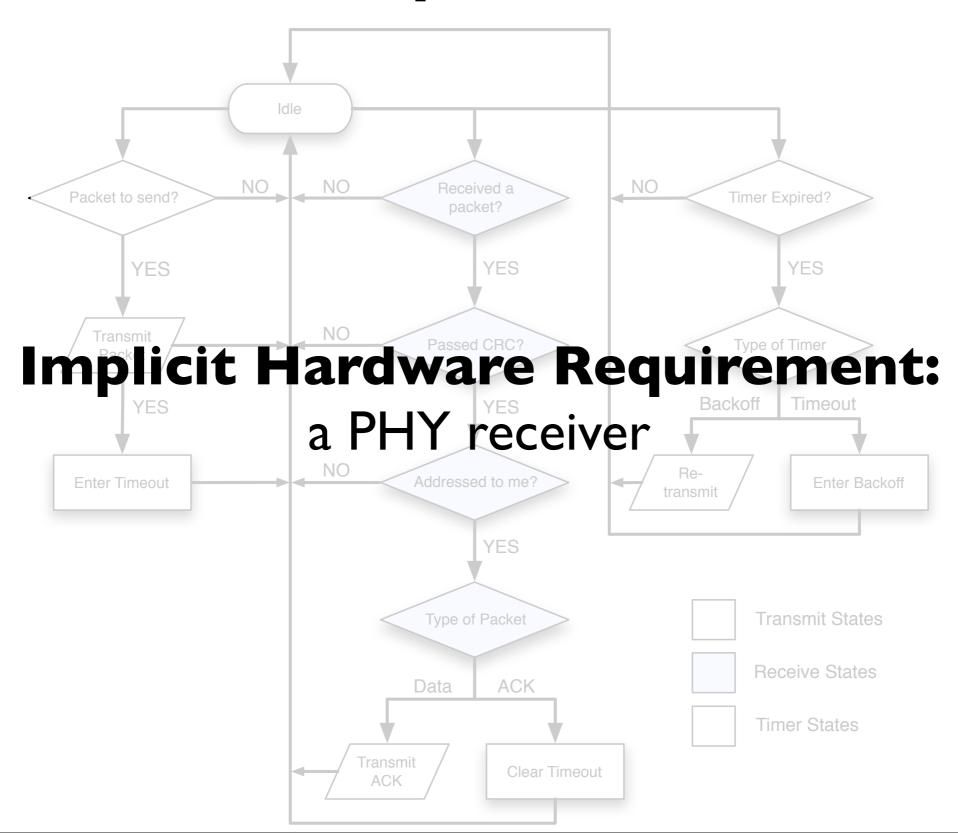
Received no ACK? Backoff and resend

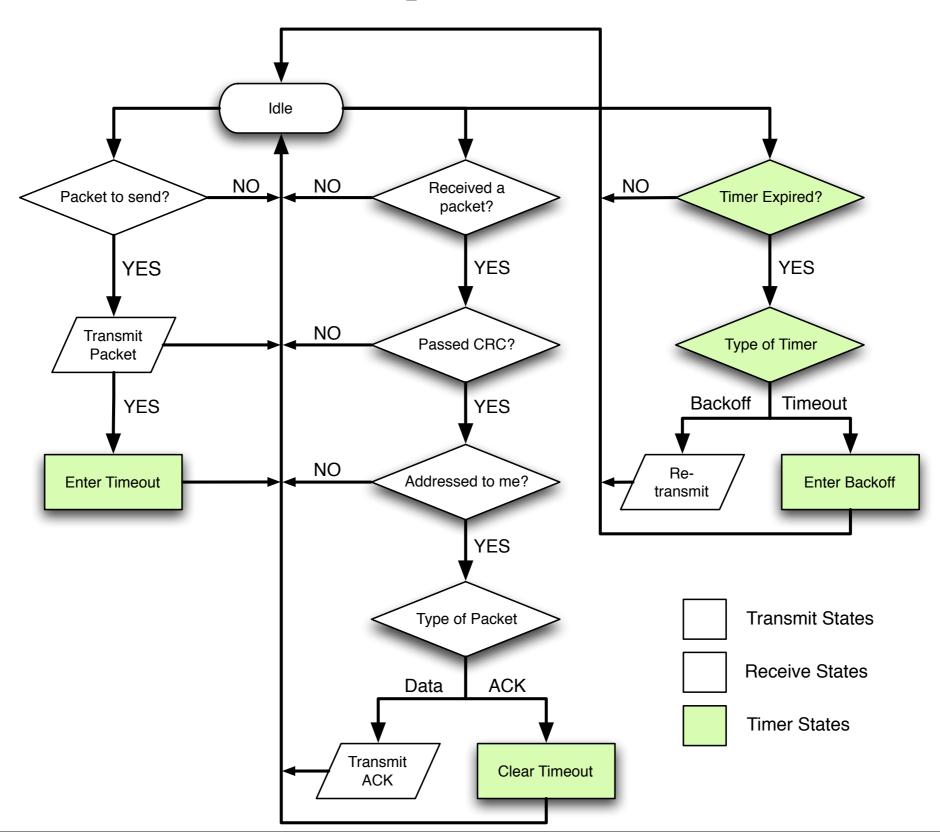


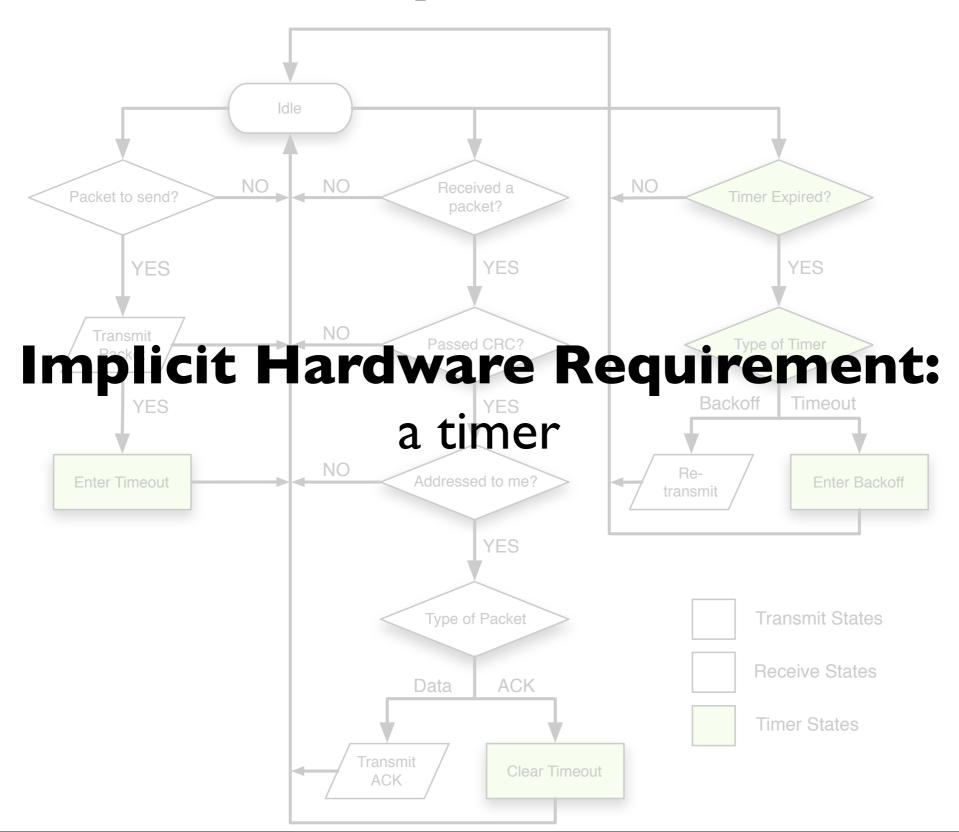


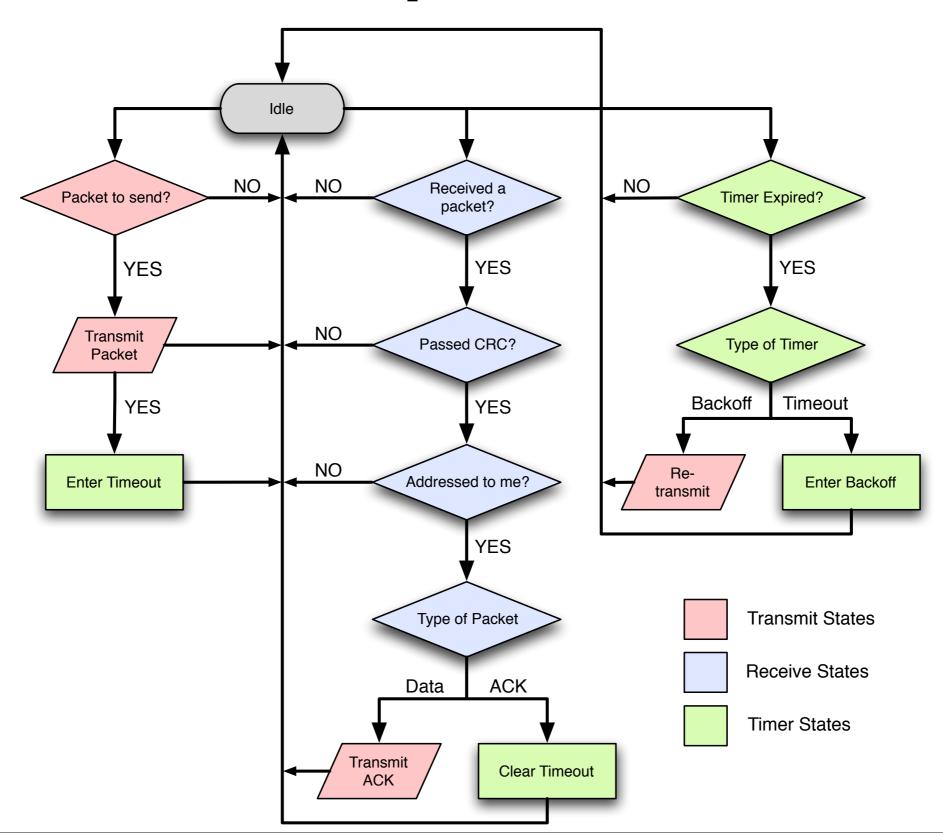


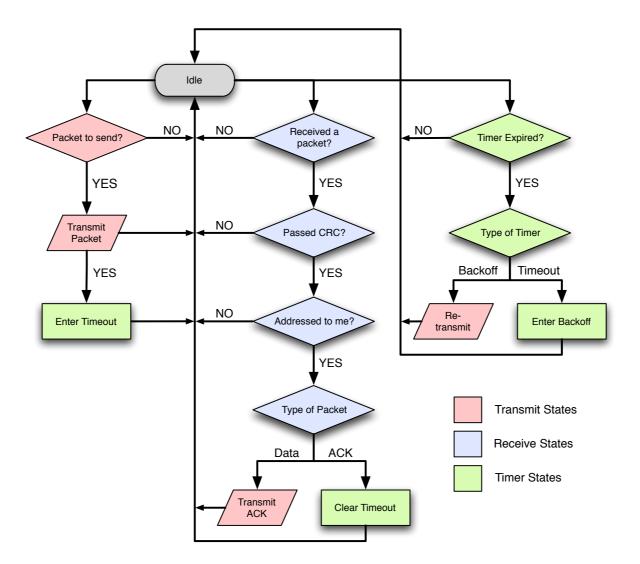


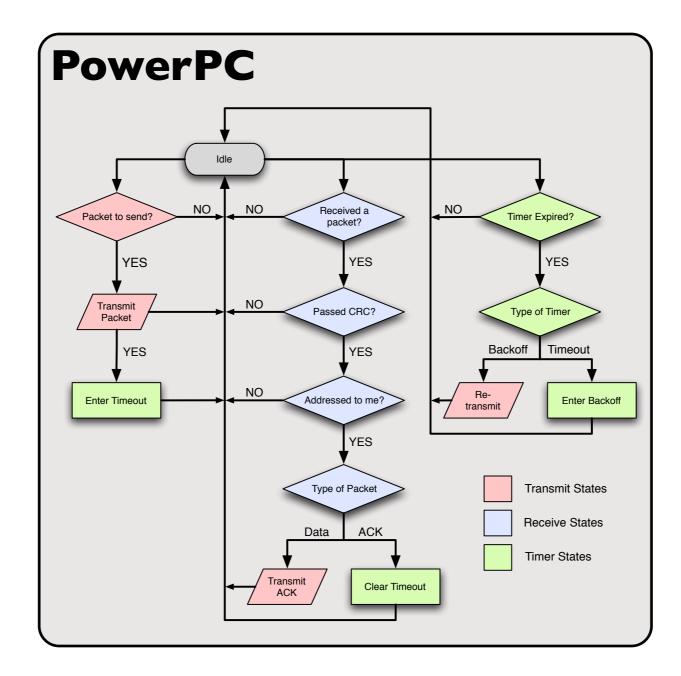


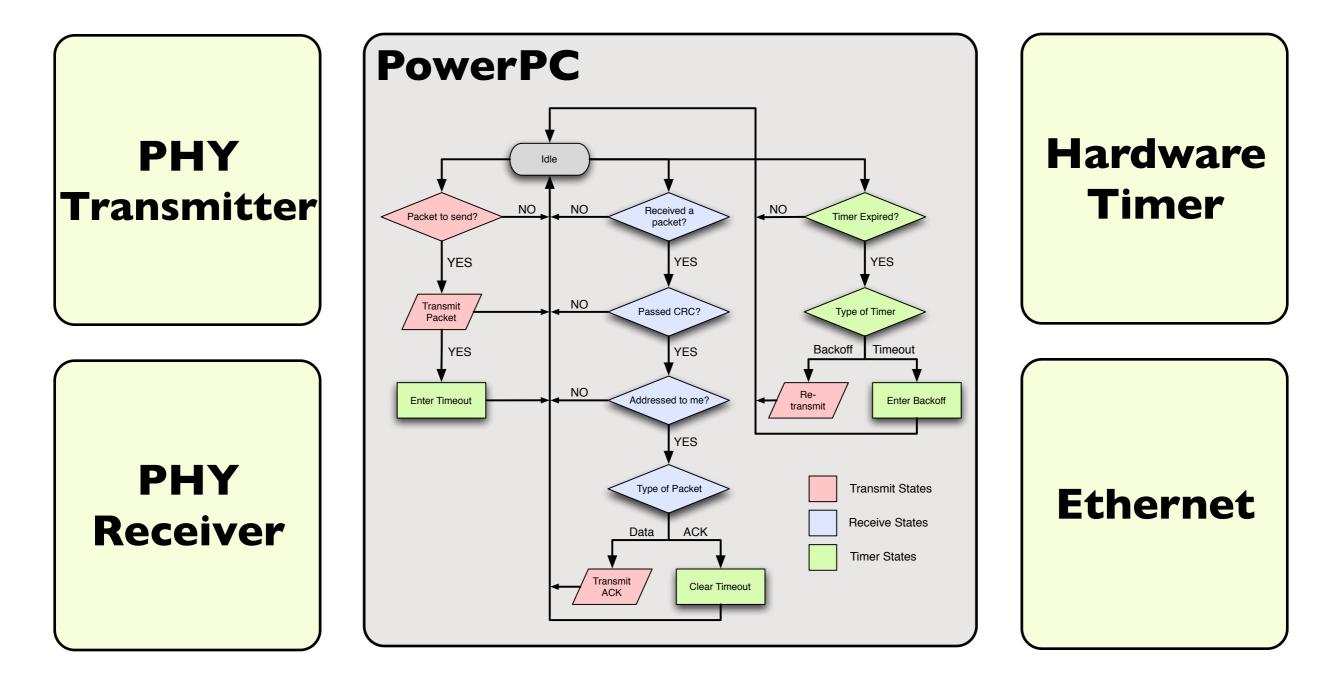


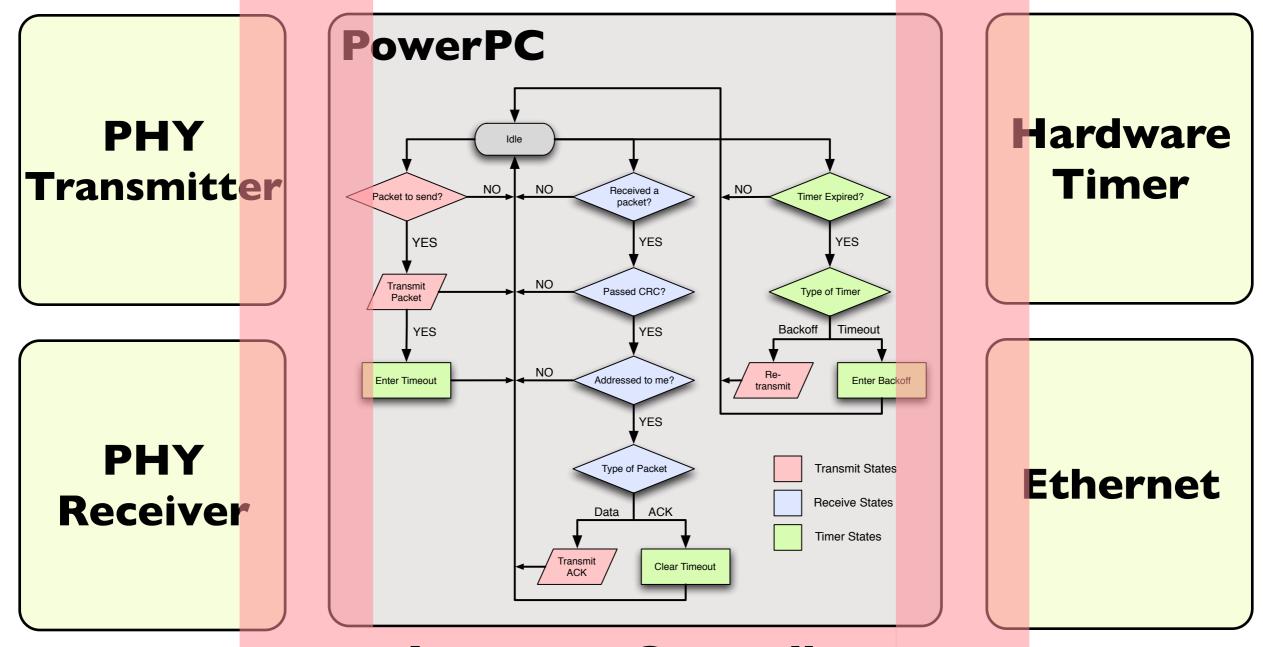




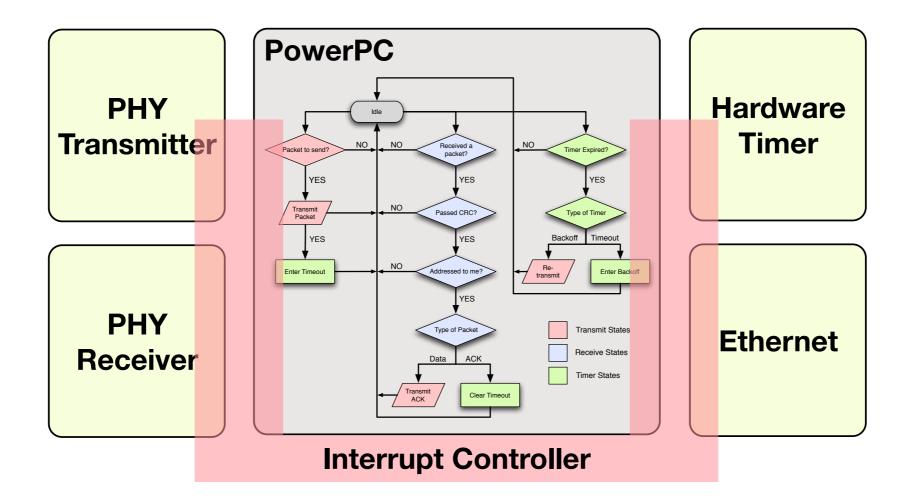


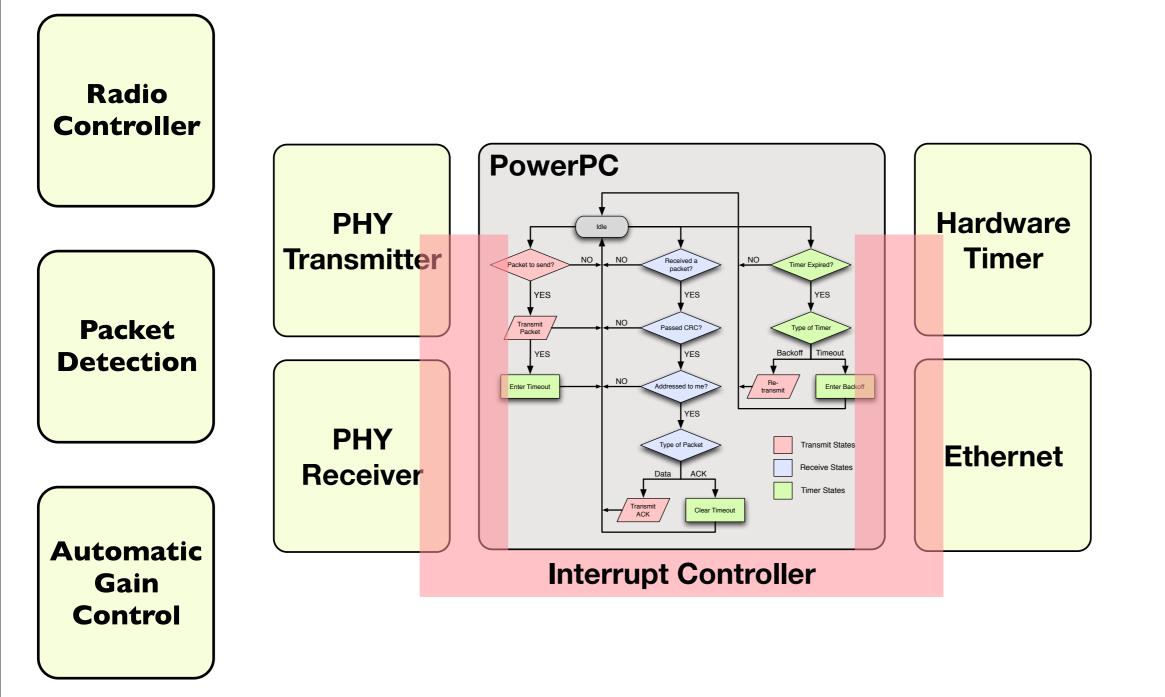


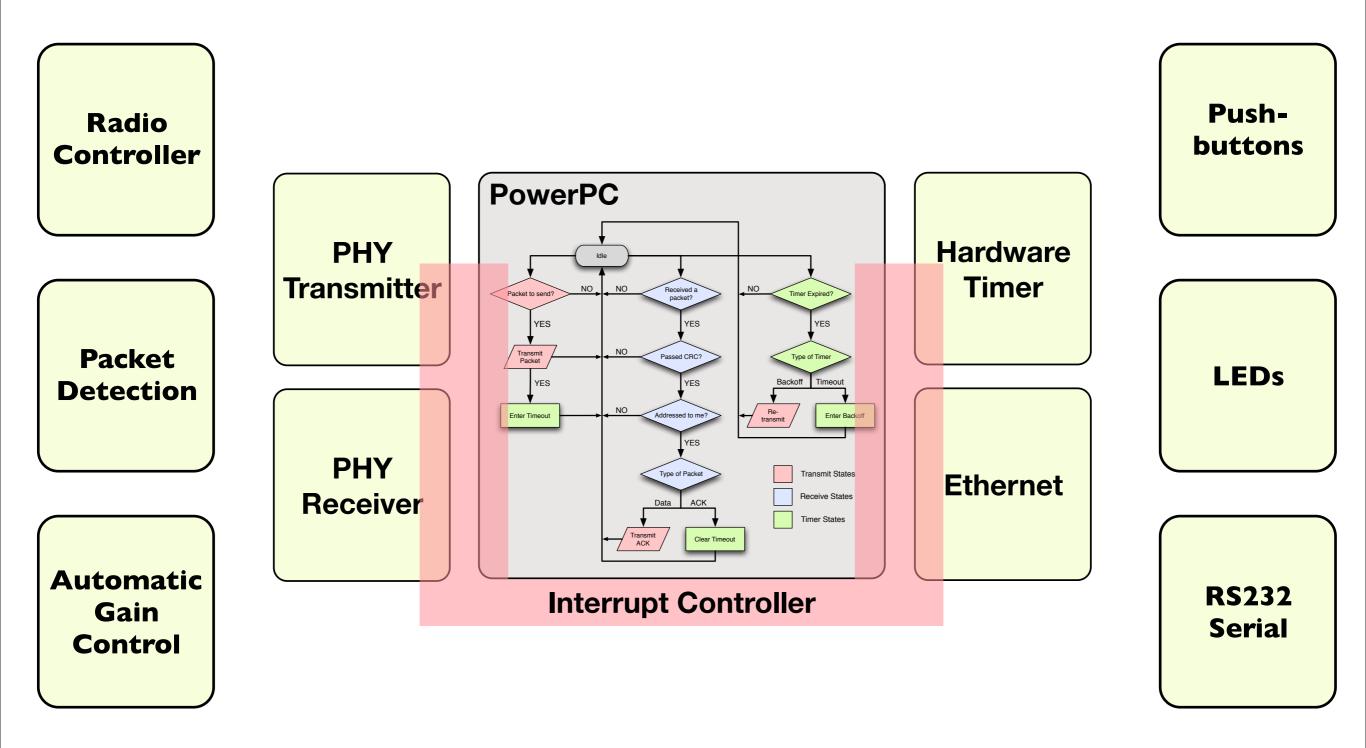


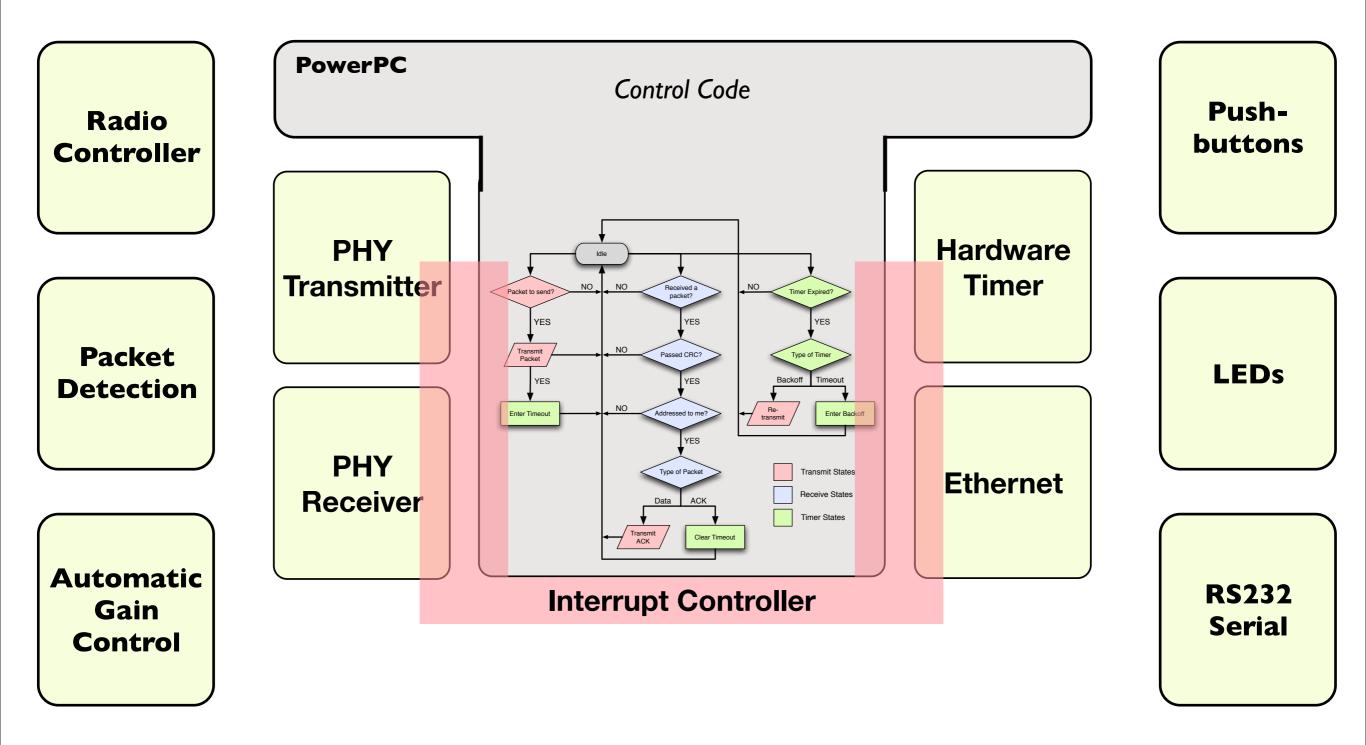


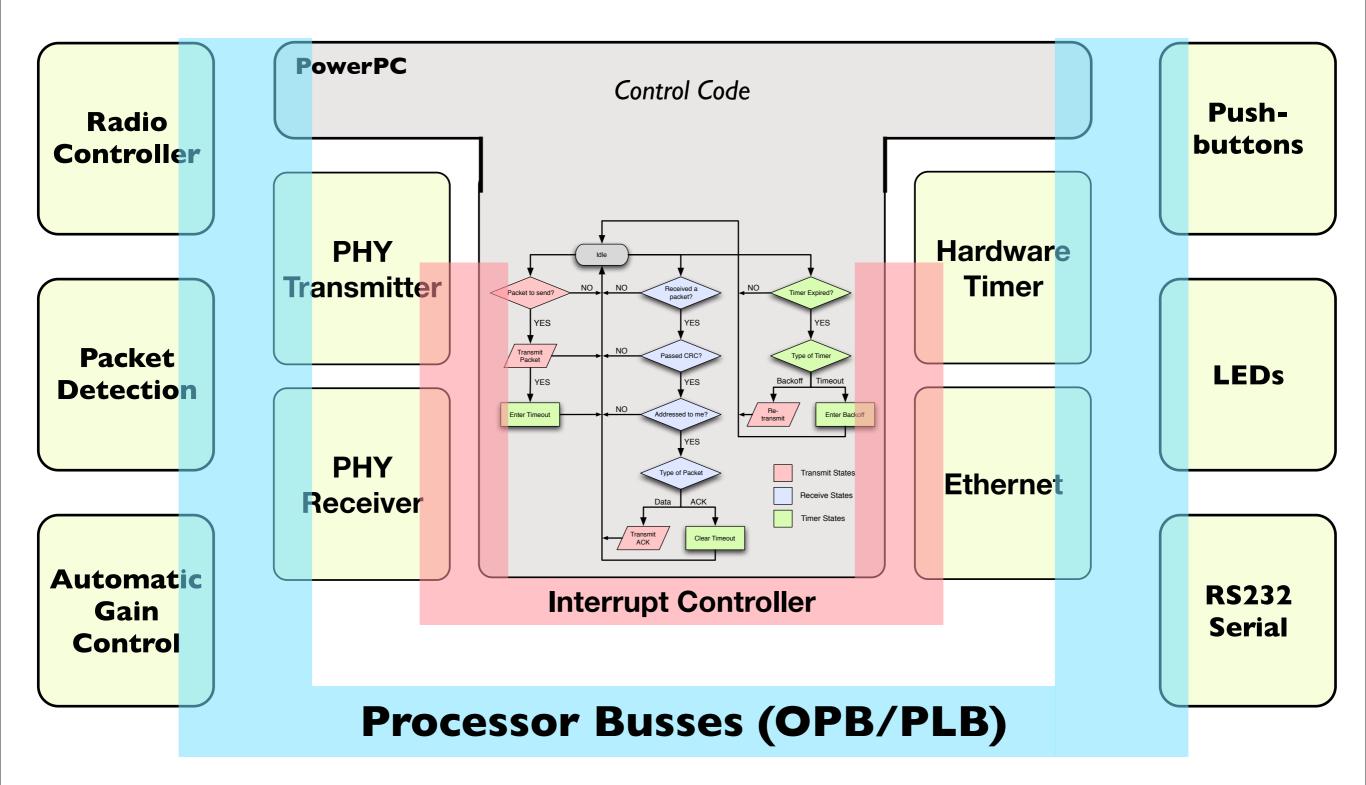
Interrupt Controller

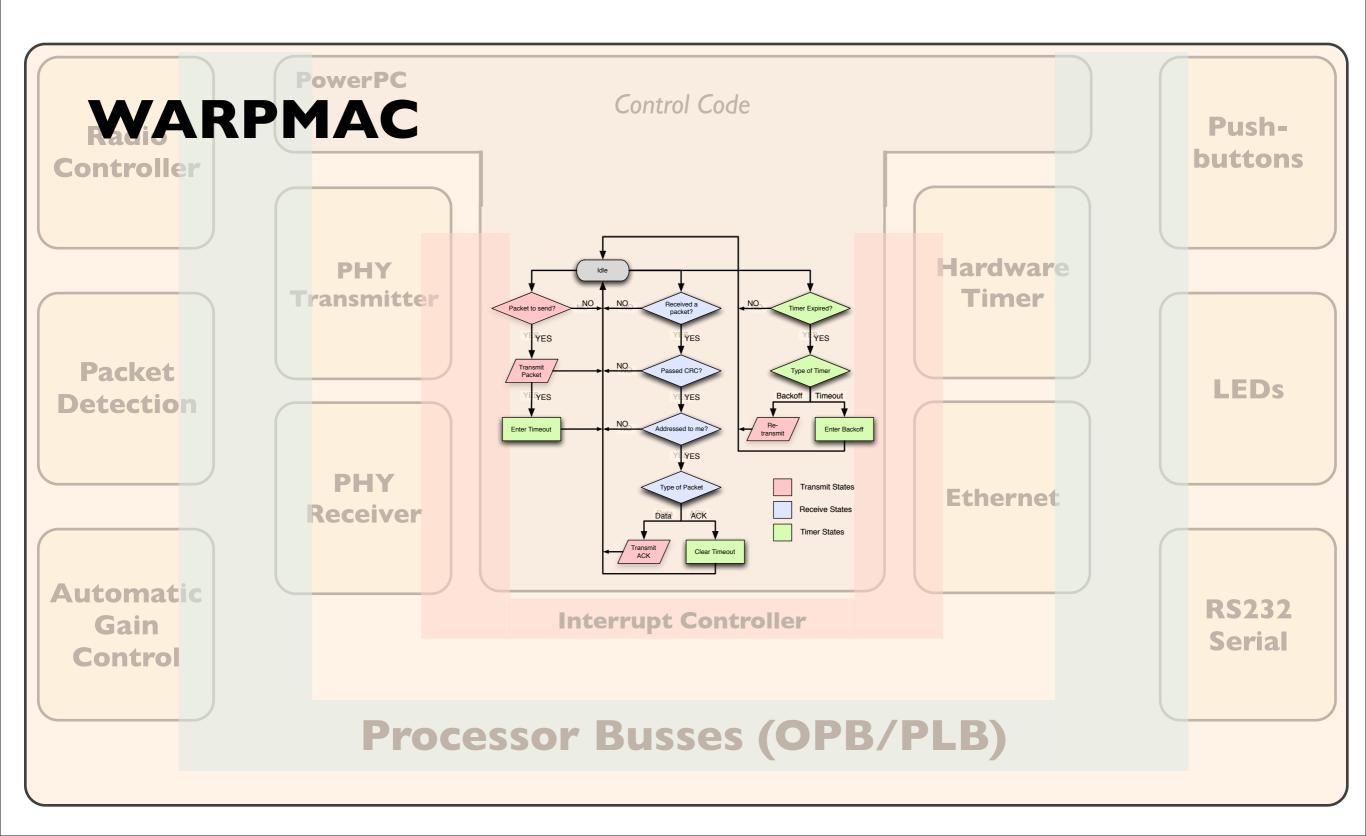




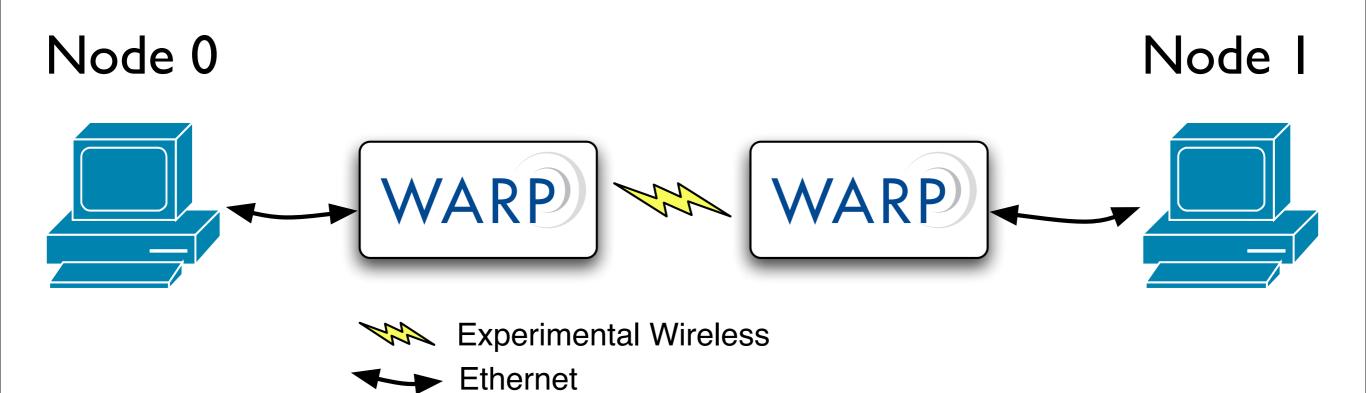








Detailed Example CSMA



WARPMAC Structs

Macframe:

- phyHeader header /* Another struct */
- unsigned char isNew /* Flag for new packets */

Fully documented in API (http://warp.rice.edu/WARP_API)

WARPMAC Structs

phyHeader:

unsigned char fullRate; /* Payload modulation rate */ unsigned char reserved4; /* Unused */ unsigned short int length; /* Payload length */ unsigned char pktType; /* Packet type */ unsigned char destAddr[6]; /* Destination address */ unsigned char srcAddr[6]; /* Source address */ unsigned char currReSend; /* Re-send count */ unsigned char reserved I; /* Unused */ unsigned char reserved2; /* Unused */ unsigned char reserved3; /* Unused */ unsigned short int checksum; /* CRC placeholder */

Fully documented in API (http://warp.rice.edu/WARP_API)

```
Let main(3)
   untigned than antist - Bi
   //Read Dig Settik colum from SPEA hoard
   //This value will be used on me index lets the residing table far other ordes
   AyID = worpaus.prtMy105);
   //Crapts on orbitrary address for this make
   unsigned than inpiAddr[6] = {Bal6, Bal4, Ba63, Ba53, Bac2, Bac2omy10};
   managyCryAddr., UnpAddr., 63;
   279113 on arbitrary routing table as that soles know each others' addresses
   unvigned they to
    ForCist; 1=10:1++30
        routsTable[1].addr[0] = wyAddr[0];
       routeTable[i].mddr[1] - myAddr[1];
        routsTuble[1].addr[]] = wykidr[]];
        routeTuble[(].udd=[]] = #yAddr[]])
        restaTuble[i].addr[4] = myAddr[4])
       rmutefuble[i].oddr[] = #yAddr[]; + #y10;
   Winistalize the frammark
   sarpenc_init():
   marphair_setMacAddr(%nyAddr);
   warpenus_actMonResend(#)
   warpheac, hetMosCH(1);
   adrpinac_setT(emout(160))
   wirpmut_net51et7ime(5);
   margenuc_setEsSuffer(&ralluffer, #)
   strpmc_art[shuffar(1);
   mmspyCtaBuffer.header.arcAddr.myAddr.fil(
   marphuc_setCondPackstHeruflerCrecely#GoodPacket);
   warpout: setRadPacketHandler(receiveRadPacket):
    eurpeut_setTimerHandlerCtimerExpire];
   surproc_setEmscHuidler(sthernet_collback3;
   worpmic_setOwned(CH2_2_0);
```

```
Lot main()(
   //Read Dig Settik celus from SMCA hoard
   //This value will be used on me today tets the multing table far uther odds.
   AyID = worphic_prtMy1dC);
   artslanted ither impAddr[6] - [Bal6, 8a24, 8a63, 8a13, 8ac2, 8ac2omy10];
   mmaryCryAddr, UnpAddr, 53;
   271113 as arbitrary routing table as that nodes know each other
    forCl+D:1+10:1++3(
        rests[ldbld[1].dddr[0] = myAddr[0];
        routsTable[i].midr[1] - muAddr[1]:
        routsTuble[1].addr[d] = wykidr[T];
        routeTuble[(].udd=[]] = wykidr[[]]
        restaTuble[i].addr[4] = myAddr[4])
        rauts[uble[i].addr[] = eyAddr[i]. = eyAddr[i].
   serpenc.init():
    surprise_satMacAddr(%eyAddr);
                                                                       displays
   surproc_setModlasetd(#);
   warphusc.httMusCF(1);
   adrinuc_setT(ensut(CEAR))
   wirpmid_netSletTime(5);
    margemic_setReduffer(&calluffer,R))
    strpmc_art[shuffar(1);
   mmspy(tallaffer.header.br(Addr,myAddr,h))
    warphuc_setCondPackstHandTarCrecktVeGoodPacket);
    warpnuc_setMadPacketHandler(receiveMadPacket);
```

eurpent_setTimerHundlerCtimerExpire); warpmac_setEmacMundlerCetHernet_collback3;

```
warpowit, antiOvarior1(CONE_3, IC);
```

- Reads the value from the dip switch on the FPGA board for use as identification
- This function also displays the value on the seven-segment

```
Lot main()(
   //This unlise will be used as no lodes lets the emiting table fur other codes
   AyID = serpeic.prtMyIdC);
   //Crapts on orbitrary address for this make
   unsigned that inpiAddr[6] = [Bel6, Bel4, Be63, Bu53, Bec2, Bec2emy10];
   managyCryAddr., UnpAddr., 63;
   27113 as arbitrary routing table as that nodes know each others' addresses
   unvigned they to
    ForClub:Lelb:1++3[
                                                                 restaTable(t).addr(0) = wyAddr(0);
        routeTable[i].uddr[1] - myAddr[1];
        routsTuble[1].addr[]] = wykidr[]];
        routeTuble[(].udd=[]] = #yAddr[]]
                                                                      node ID
        restaTuble[i].addr[4] = myAddr[4])
       rmutefuble[i].oddr[] = #yAddr[]; -#y10;
   serpenc.init():
   surprise_satMacAddr(%eyAddr);
   surproc_setModlasetd(#);
   warphusc.httMusCF(1);
   adrymuc_setT(mout(160))
   wirpmid_netSletTime(5);
   margemic_setReBuffer(&calluffer,R))
   strpmc_art[shuffar(1);
   mmspy(tallaffer.header.arcAddr.myAddr.fi)(
    warphuc_setCondPackstHandTarCrecktVeGoodPacket);
    warpnuc_setMadPacketHandler(receiveMadPacket);
   eurpent_setTimerHundlerCtimerExpire);
   surproc_setEnschundler(sthernet_collbuck3;
    warpowit, antiOvarior1(CONE_3, IC);
```

- Defines an arbitrary address, based on the node ID
- Specifies a crude "routing table" to allow nodes to communicate with one another using only the node IDs

```
Lot main()(
   Within walke will be used on me index lets the multing table far-
   AvID = serpenic_petMy1dC);
   antsigned that inpidde[6] - [Bal6, Bal4, Ba63, 0x53, 0xe2, Bac2ony10];
   mmaryCryAddr, UnpAddr, 53;
    for(1-0;1+10;1++)[
        routs[ld[t].dddr[0] = wyAddr[0])
        routsTable[i].midr[1] - muAddr[1]:
        routsTuble[1].addr[d] = wykidr[T];
        routeTuble[(].udd=[]] = wykidr[[]]
        restaTuble[i].addr[4] = myAddr[4])
        rmatefuble[i].oddr[1] = myAddr[1]. + myl0;
   Winitighter the frammerk
   surprise_init():
   marpman setMacAddr(KeyAddr);
   warpetus_actMonResend(#);
   warpheac, hetMosCH(1);
```

warphuc_setMosCB(1); warphuc_setT(meout(160); warphuc_setS1s(Time(5);

margemic_selfsduffer(&rsHuffer,#))
margemic_ast1sduffer(1);

memopyCtwBuffer.header.proAddr.myAddr.filt

warpmac_setCondPackstHandler(receiveGoodPacket); warpmac_setRadPacketHandler(receiveBadPacket); exrpmac_setTimerHandler(timerExp(re); warpmac_setEmacHandler(etHernot_collback);

```
warpower, ant Overseal CONT_2_10 c
```

- Initializes the framework
 - Initializes PHY, radio, AGC, packet detection, interrupts, etc.
- Sets specific parameters
 - 8 resends
 - Maximum contention window of 5 * (Slot-time)
 - I 60 usec timeout
 - 9 usec Slot-time

```
Let. main(3)(
   //This unlise will be used as no lodes lets the maiting table fur other codes
   AvID = serpesic_petMy1dC);
   antigread other implable[6] = [Bal6, Bal4, Ba63, Ba53, Bac2, Bac2eny10];
   mmapyCryAddr., traAddr., 53;
    ForCloD:Lab:14+36
        restsTable(t).sidr(0) = wyAddr(0);
        rootsToble[i].mddr[1] - myAddr[1]:
        routsTuble[1].oddr[d] = wyAddr[T])
        routeTuble[(].udd=[]] = wykidr[]];
        restaTuble[i].addr[4] = myAddr[4])
        rauts[uble[i].addr[1] = #yAddr[5]+1-#y10;
   //Initialize the frammark
   sarpenc_init():
   marpman_setMacAddr(%eyAddr);
   warpetor_actMonResend(#)
   warphase_betMoxCH(1);
   adrymuc_setT(emout(CE60))
   wirpmut_art51st7ime(5);
    margemit_petReduffer(&rallaffer,R))
   strptuc_act1shuffar(1);
   mmspy((allaffer.header.arcAdd).myAdd/.h)(-
    warphurc_setCondPacks(HandTarCreck[veGoodPacket);
```

serpenc.artOvener10002_2.01

eurpeut_setTimérHundlérCtimerExpire);

warpnuc_setRadPacketHandler(receiveRadPacket);

surproc_setEnschundlerCethernet_collback3;

```
marpsuc_setReduffer(&reduffer,R))
marpsuc_astTeBuffer(1);
```

```
memopyCtaBuffer.header.orcAddr.myAddr.631
```

```
marpmuc_setCondPackstHandler(receiveGoodPacket);
warpmuc_setBadPacketHandler(receiveBadPacket);
marpmuc_setTimerHandler(timerExptre);
marpmuc_setEmacHandler(ethernet_collback);
```

```
wargenic_artOvener1(GH2_2_IO);
```

```
marphus_anableCSMA();
marphus_anableEthernetInterrupt())
```

```
//Set the modulation ochone use her here rate (hender) system)+
surpror_petEnseRate(QPSK);
```

```
with (1) (
```

```
mmmrry(myAddr,tmpAddr,6);
//fill us arbitrary counting table as that mades know asth athers' add
unvigned that t;
for(1=0;(=10;(=10;(=1))
routsTuble(1).midr(1) = myAddr(0);
routsTuble(1).midr(1) = myAddr(0);
routsTuble(1).midr(2) = myAddr(0);
routsTuble(1).midr(2) = myAddr(0);
routsTuble(1).midr(2) = myAddr(0);
//Initialize.khe frummenth
murpmut_setMosResend(2);
murpmut_setMosResend(2);
murpmut_setTuble(1);
murpmut_setTuble(1);
murpmut_setTuble(2);
```

```
marpsuc_setReBuffer(&reBuffer,B))
marpsuc_astTeBuffer(1);
```

```
memopyCtxBuffer_header_arcAddr_myAddr_hit
```

```
warpmuc_setCondPackstHandler(receiveGoodPacket);
warpmuc_setBadPacketHandler(receiveBadPacket);
warpmuc_setFlmerHandler(timerExptre);
warpmuc_setEmacHandler(etHernet_collback);
```

```
wargenut_art(Namer)(GHZ_2_IO);
```

```
warpmac_anableCSMAC3;
warpmac_anableEthernetInterrap(C);
```

```
//Set the modulation ochone use for been rate (hunder) systels 
surpror_setEnseRate(QPSK);
```

- Tells WARPMAC to receive wireless packets into a particular buffer
- Tells WARPMAC to send wireless packets from a particular buffer
- Registers user interrupt handlers with the frameworks

-111-03

```
mumpyCryAddr., UnpAddr., 53;
ForCloP:1-10:14+3[
    restsTable(t).sidr(0) = wyAddr(0);
    routeTuble[i].mddr[1] - myAddr[1]:
    routsTuble[1].oddr[1] = wykddr[1])
    routeTuble[(].udd=[]] = wykidr[]];
    routsTuble[i].addr[4] = myAddr[4])
    rmstafuble[i].addr[1] = eyhidr[1]+1-eylü;
sarpenc_init():
marganic_ha-MacAdd+('AnyAdd+');
marphus_actMonResend(#))
serpress, antMosCH(1);
adrphus_setT(eesutCE60)
mirphut_netSletTime(5);
```

```
marprox_setReduffer(&reduffer,B))
marprox_antTeBuffer(1);
```

```
memopyCtaBuffer.header.orcAddr.myAddr.631
```

```
marpmuc_setCondPackstHandler(receiveGoodPacket);
warpmuc_setBadPacketHandler(receiveBadPacket);
surpmuc_setTimerHandler(timerEspire);
surpmuc_setEmacHandler(ethernet_collback);
```

```
warpmic_artDunnel(GHZ_2,IO)
```

```
marphus_anableCSMAC3;
marphus_anableEthernetInterraptC);
```

```
//Set the modulation ochone use for been rate (Annder) system)& margenc_setEnseRate(QPSK);
```

```
while(1)(
```

```
//initialize the framework
morpmot_init();
//initialize the framework
morpmot_init();
morpmot_setMucAddr(teryAddr);
morpmot_setMusResend(T);
morpmot_setMusResend(T);
morpmot_setT(mout(160);
morpmot_setT(mout(160);
morpmot_setT(mout(160);
morpmot_setTslatfier(&rsHuffer,B))
morpmot_setTslatfier(%rsHuffer,B);
```

```
monopyCtallaffer.header.orcAddr.myAddr.Klt
```

```
warpmac_setCondPackstHardlerCreativeGoodPacket);
warpmac_setRadPacketHardlerCreativeBadPacket);
warpmac_setTimerHardlerCtimerExp(re];
warpmac_setEmacHardlerCtimerExp(re];
```

```
worpmic_setOvener1(GH2_2_ID);
```

```
warpmus_snableCSMA();
warpmus_snableEthernetInterrop(C);
```

```
//Set the modulation ochone use for been rate (hander) symbols
warpmon_petEnseRote(QPIK);
```

uni)(03)(

enturn;

```
//Initialize the framework
marphic.init();
```

```
marphic_sasMocAld+(SnyAddr2)
```

```
warpmuc_actMosResend(#);
warpmuc_actMosCH(1);
adrpmuc_actT(meout(EE00);
warpmuc_actSletTime(5);
```

```
margemic_setReduiffer(&rsHuffer,B))
margemic_art1eBuffer(1);
```

```
monopyCtaBuffer.header.orcAddr.myAddr.631
```

```
warpmac_setGondPackstHardlerCreceiveGoodPacket);
warpmac_setBadPacketHardlerCreceiveBadPacket);
warpmac_setTimerHardlerCtimerExp(re];
warpmac_setEmacHardlerCetHermot_collback3;
```

warzmaic.art/bannel(GHZ.2.IO)

```
marphus_anableCSMA();
marphus_anableEthernetInternapt();
```

//Set the modulation ochone use for here rote (funder) symbols
warpmac_retEnseRote(OPSK);

whished 234

```
ortum;
```

- Sets the frequency band to 802.11 channel 8 of the 2.4GHz band
- Enable carrier-sensing mode of WARPMAC
- Enable the Ethernet interrupt
- Set the base modulation rate to QPSK (must be agreed upon by all nodes in the network)
- Spins forever in a while loop, waiting for an interrupt

Case I: Packet received from Ethernet

```
int ethernet_callback(Xuint32 length, char* payload){
   warpmac_disableEthernetInterrupt();
   txBuffer.header.currReSend = 0;
   txBuffer.isNew = 1;
   txBuffer.header.length = length;
   txBuffer.header.pktType = DATAPACKET;
```

```
//Set the modulation scheme for the packet's full-rate symbols
txBuffer.header.fullRate = QPSK;
```

```
//Copy in the packet's destination MAC address
//Hard-coded as this node's partner node
memcpy(txBuffer.header.destAddr,routeTable[(myID+1)%2].addr,6);
```

```
if(warpmac_carrierSense()){
    warpmac_sendOfdm(&txBuffer);
    warpmac_setTimer(TIMEOUT);
}
else{
    warpmac_setTimer(BACKOFF);
}
return 0;
```

7

- Disables the Ethernet interrupt line until this frame is dealt with
- Metadata and header information is filled in
 - isNew = I, since it is a new packet
 - Length, packet type, full rate modulation order and the destination MAC address are filled into the header
- If the medium is free, the packet is sent and a timeout begins

Case 2: "Bad" packet received from OFDM

int receiveBadPacket(Macframe* packet) { warpmac_incrementLEDLow(); }

- If we receive a packet that fails checksum
 - Blink the bottom LEDs
- This way we can have a visualization of channel quality

Case 3: "Good" data packet received from OFDM

```
int receiveGoodPacket(Macframe* packet) {
```

```
worpmoc_incrementLEDHigh();
```

```
if(warpmac_addressedToMe(packet)){
   Macframe ackPacket;
   switch(packet->header.pktType){
        case ACKPACKET;
   }
}
```

```
if(warpmac_inTimeout()){
    warpmac_clearTimer(TIMEOUT);
    txBuffer.header.currReSend = 0;
    txBuffer.isNew = 0;
    warpmac_enableEthernetInterrupt();
}
```

break;

```
case DATAPACKET:
    warpmac_leftHex(packet->header.currReSend);
    ackPacket.header.length = 0;
    ackPacket.header.pktType = ACKPACKET;
    ackPacket.header.fullRate = QPSK;
    memcpy(ackPacket.header.srcAddr,myAddr,6);
    memcpy(ackPacket.header.destAddr,packet->header.srcAddr,6);
    warpmac_setTxBuffer(2);
    warpmac_setTxBuffer(2);
    warpmac_setTxBuffer(1);
```

```
packet->header.currReSend = 0;
packet->isNew = 1;
warpmac_phyInterruptClear();
warpmac_sendEthernet(packet);
packet->header.currReSend = 0;
packet->isNew = 0;
```

```
breck;
```

- Blink the top LEDs
- If destination address is equal to my source address and the type is a data packet
 - Create an acknowledgment and send it
 - Send the packet over Ethernet

```
}
return 0;
}
```

}

Case 4: "Good" acknowledgment packet received from OFDM

```
int receiveGoodPacket(Macframe* packet) {
```

warpmac_incrementLEDHigh();

```
if(warpmac_addressedToMe(packet)){
   MacFrame ackPacket;
   switch(packet->header.pktType){
      case ACKPACKET:
```

```
if(warpmac_inTimeout()){
    warpmac_clearTimer(TIMEOUT);
    txBuffer.header.currReSend = 0;
    txBuffer.isNew = 0;
    warpmac_enableEthernetInterrupt();
}
```

break;

```
case DATAPACKET:
    warpmac_leftHex(packet->header.currReSend);
    ackPacket.header.length = 0;
    ackPacket.header.pktType = ACKPACKET;
    ackPacket.header.fullRate = QPSK;
    memcpy(ackPacket.header.srcAddr,myAddr,6);
    memcpy(ackPacket.header.destAddr,packet->header.srcAddr,6);
    warpmac_setTxBuffer(2);
    warpmac_setTxBuffer(2);
    warpmac_setTxBuffer(1);
```

```
packet->header.currReSend = 0;
packet->isNew = 1;
warpmac_phyInterruptClear();
warpmac_sendEthernet(packet);
packet->header.currReSend = 0;
packet->isNew = 0;
```

```
break;
```

return 8;

- Blink the top LEDs
- If destination address is equal to my source address and the type is an acknowledgment
 - If a timeout is currently running (i.e., the node is waiting on an ACK)
 - Stop the timer
 - Turn Ethernet interrupts back on (they were disabled in the ethernet handler)

Case 5: Timeout timer expires

```
int timerExpire(unsigned char timerType){
    int status;
    switch(timerType){
        case TIMEOUT:
            if(txBuffer.isNew){
                status = warpmac_incrementResend(&txBuffer);
                if(status == 0){
                    warpmac_enableEthernetInterrupt();
                    return 0;
                ł
                warpmac_setTimer(BACKOFF);
                return 0;
            break:
        case BACKOFF:
            if(warpmac_carrierSense()){
                warpmac_sendOfdm(&txBuffer);
                warpmac_setTimer(TIMEOUT);
            else{
                warpmac_setTimer(BACKOFF);
            break;
            return 0;
   }
}
```

- Increment the resend field of the packet
- Enter a backoff
- Re-enable Ethernet interrupts if maximum retransmissions were met

Case 6: Backoff timer expires

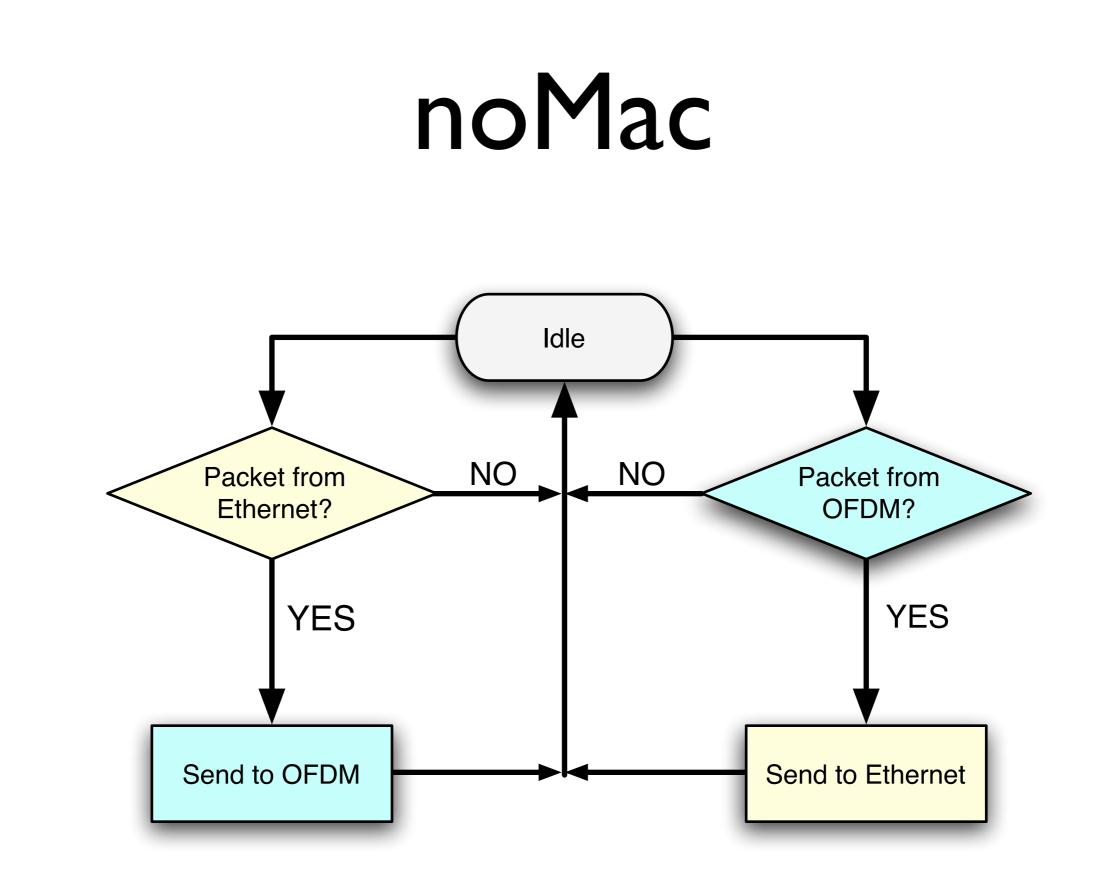
```
int timerExpire(unsigned char timerType){
    int status;
    switch(timerType){
        case TIMEOUT:
            if(txBuffer.isNew){
                status = warpmac_incrementResend(&txBuffer);
                if(status == 0){
                    warpmac_enableEthernetInterrupt();
                    return 0;
                warpmac_setTimer(BACKOFF);
                return 0;
            break;
        case BACKOFF:
            if(warpmac_carrierSense()){
                warpmac_sendOfdm(&txBuffer);
                warpmac_setTimer(TIMEOUT);
            }
            else[
                warpmac_setTimer(BACKOFF);
            break:
            return 0;
   }
}
```

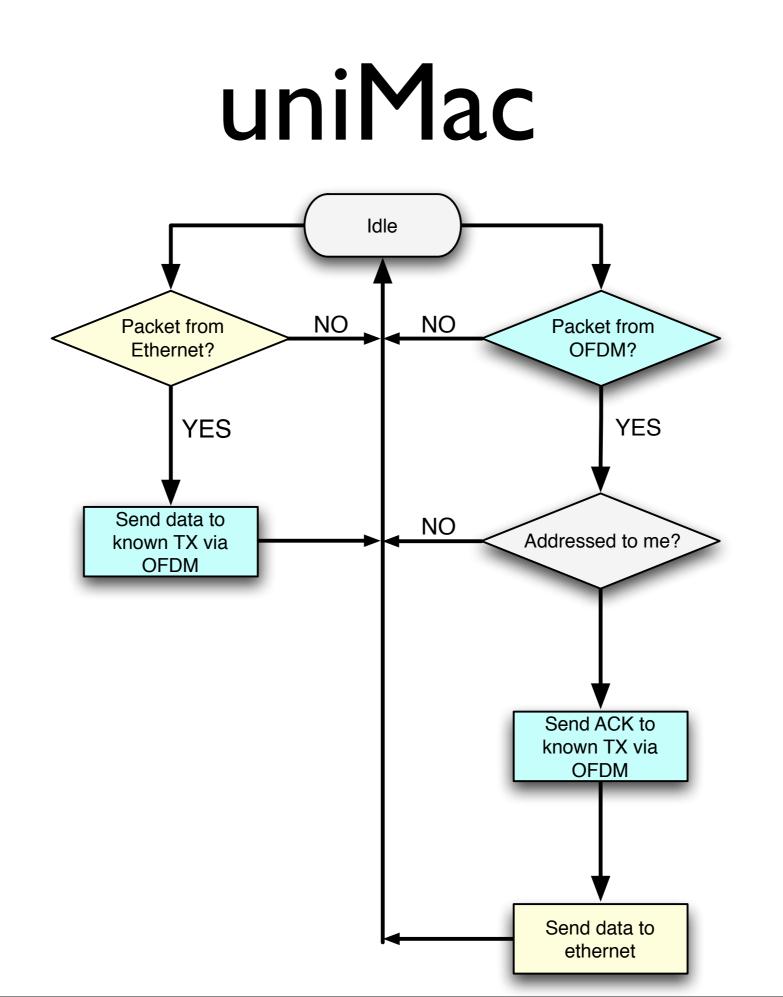
- If the medium is free
 - Send it over OFDM
 - Enter a timeout
- Otherwise, start another timeout



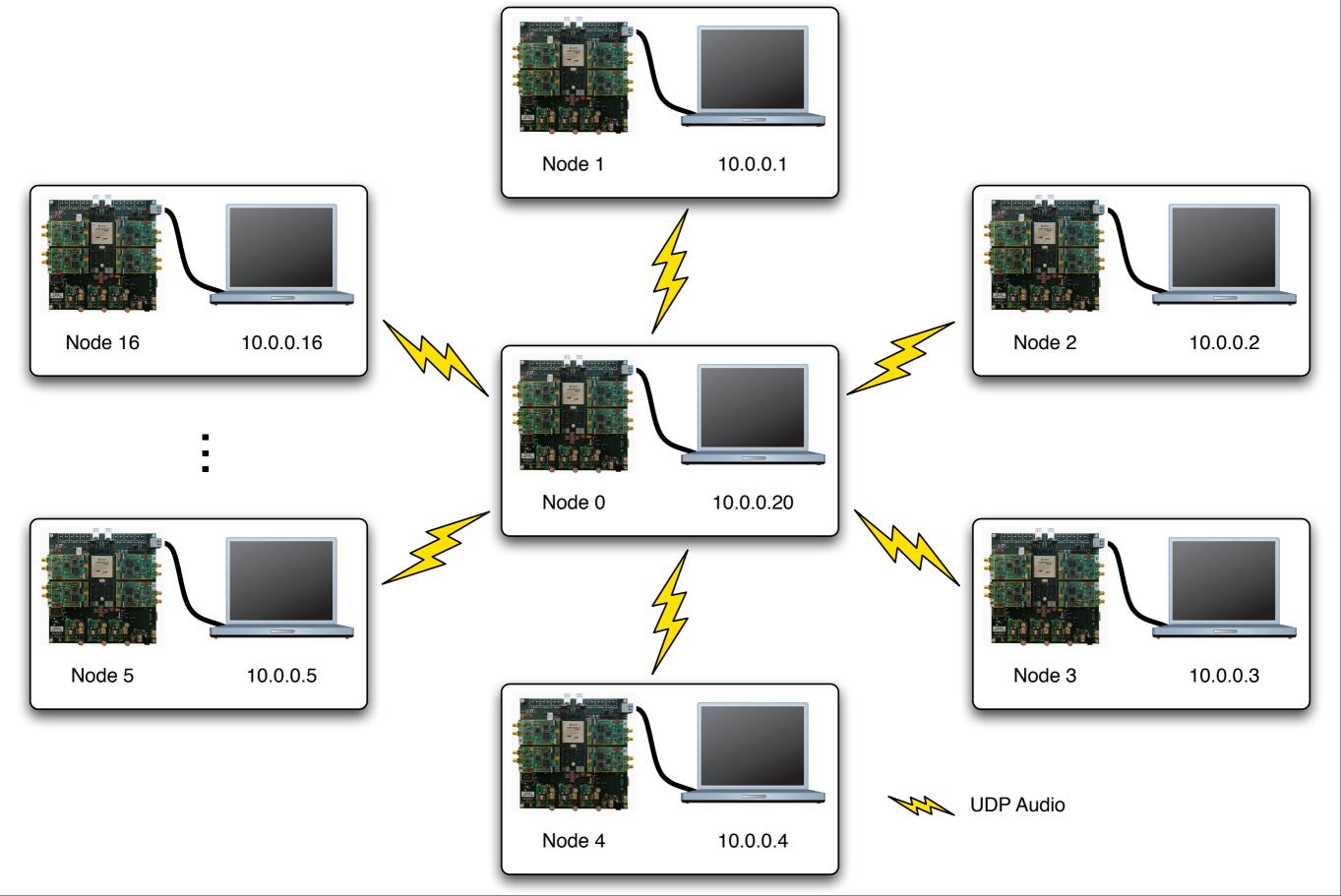
Lab Exercises

- Lab 4 noMAC: Simple "MAC" layer
- Lab 5 uniMAC: Unidirectional MAC
- Lab 6 hopMAC: Channel-hopping MAC

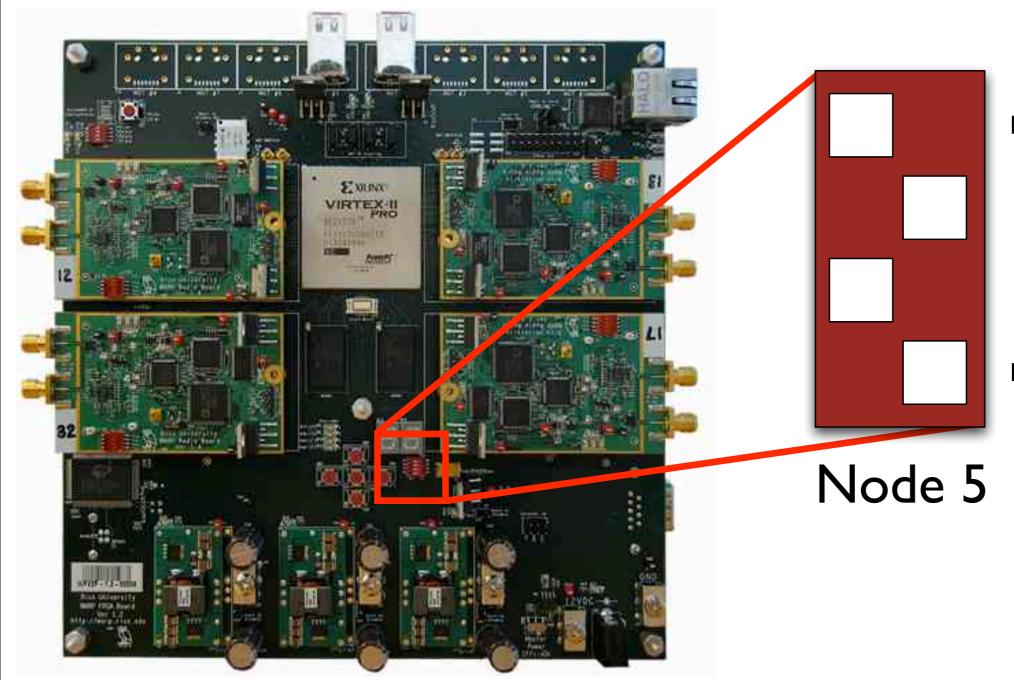




uniMac

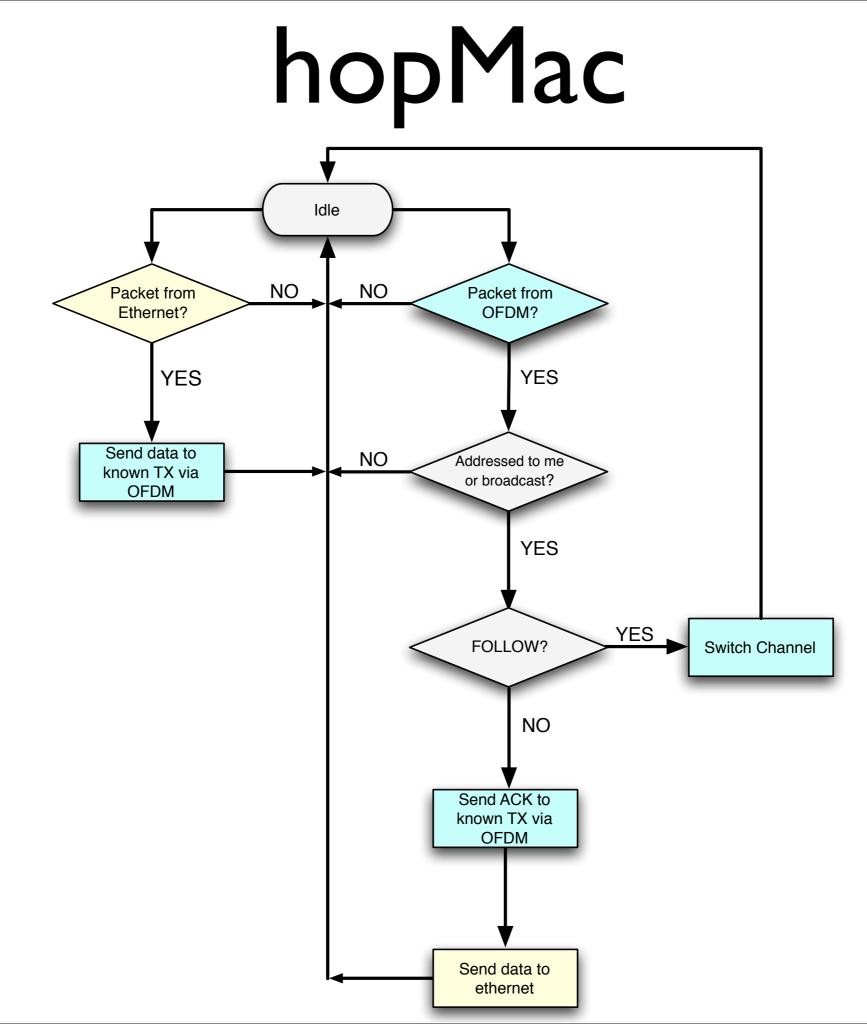


uniMac Lab

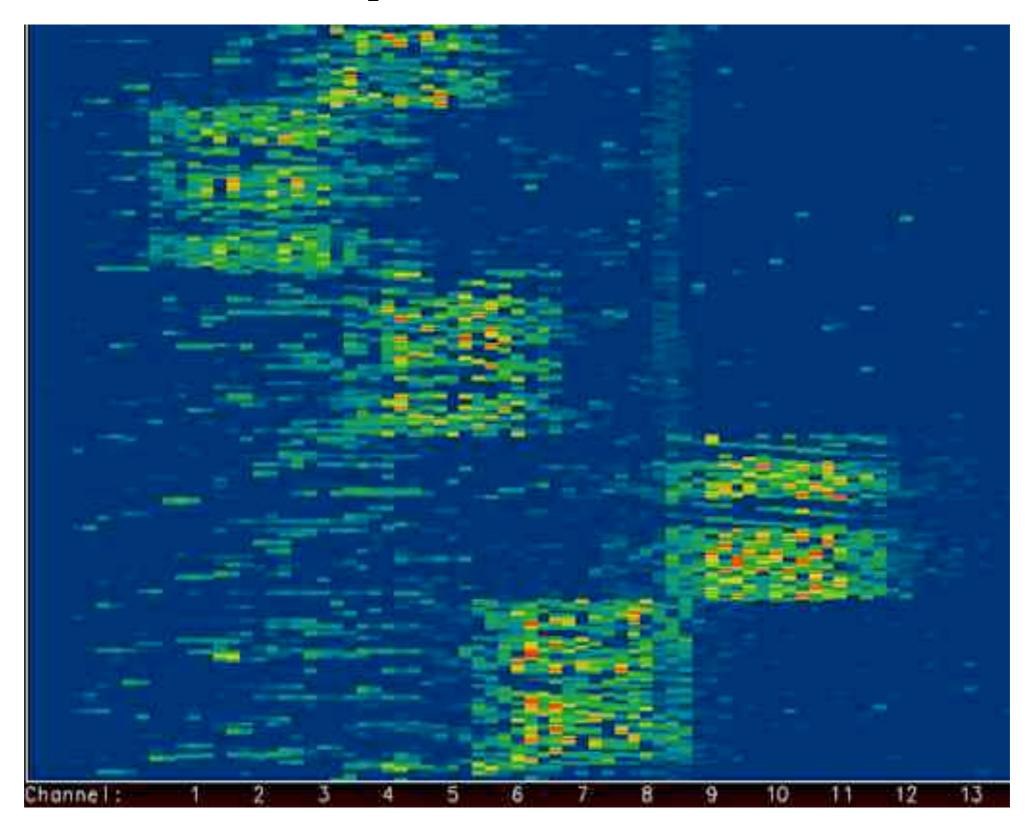


Most Significant Bit (MSB)

Least Significant Bit (LSB)



hopMac Lab



Questions?

Remember to use the API: http://warp.rice.edu/WARP_API

Logistics

- Review forms
- Contacting us
 - Support & technical questions
 - <u>http://warp.rice.edu/forums/</u>
 - Hardware sales
 - <u>http://warp.rice.edu</u>/
 - <u>warp-project@rice.edu</u>