fdwifiモジュール

静岡大学 情報学研究科
杉山 佑介

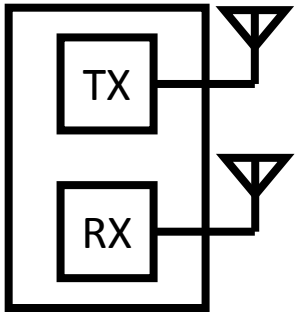
2014/07/22

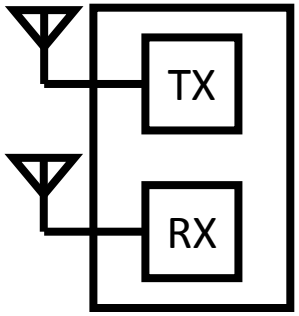
fdwifiモジュール

- 無線全二重通信のモジュール
  - RFD（Relay Full-Duplex）-MACを実装
- wifiモジュールを拡張
- 対応範囲
  - IEEE 802.11 a
  - AODV，DSRなどのルーティングプロトコル
- 対応範囲外
  - IEEE 802.11 a以外の標準
  - Qos制御
  - パケットのセグメント化

無線全二重通信 1/3

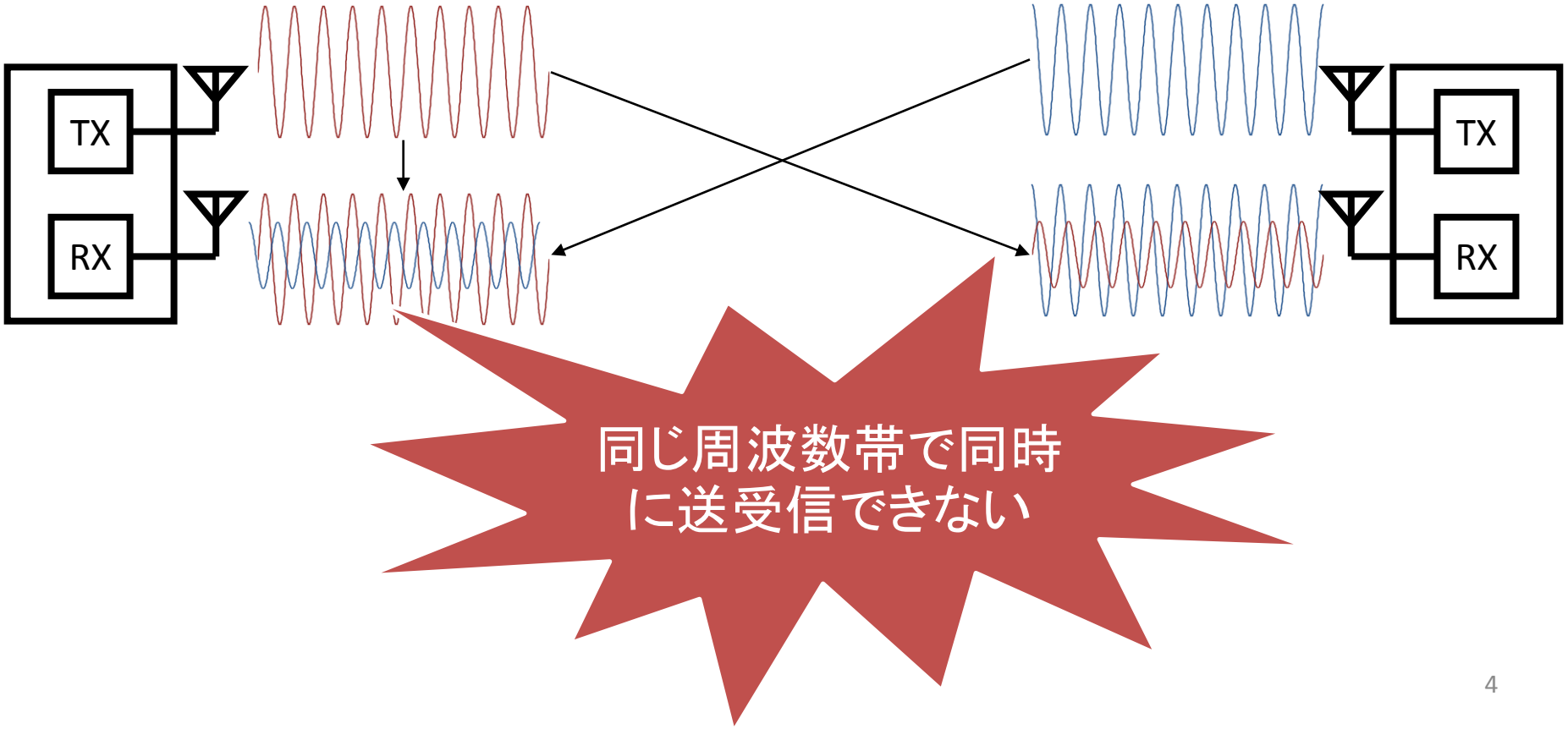
- 無線全二重通信は同じ周波数帯で同時に送受信可能な技術





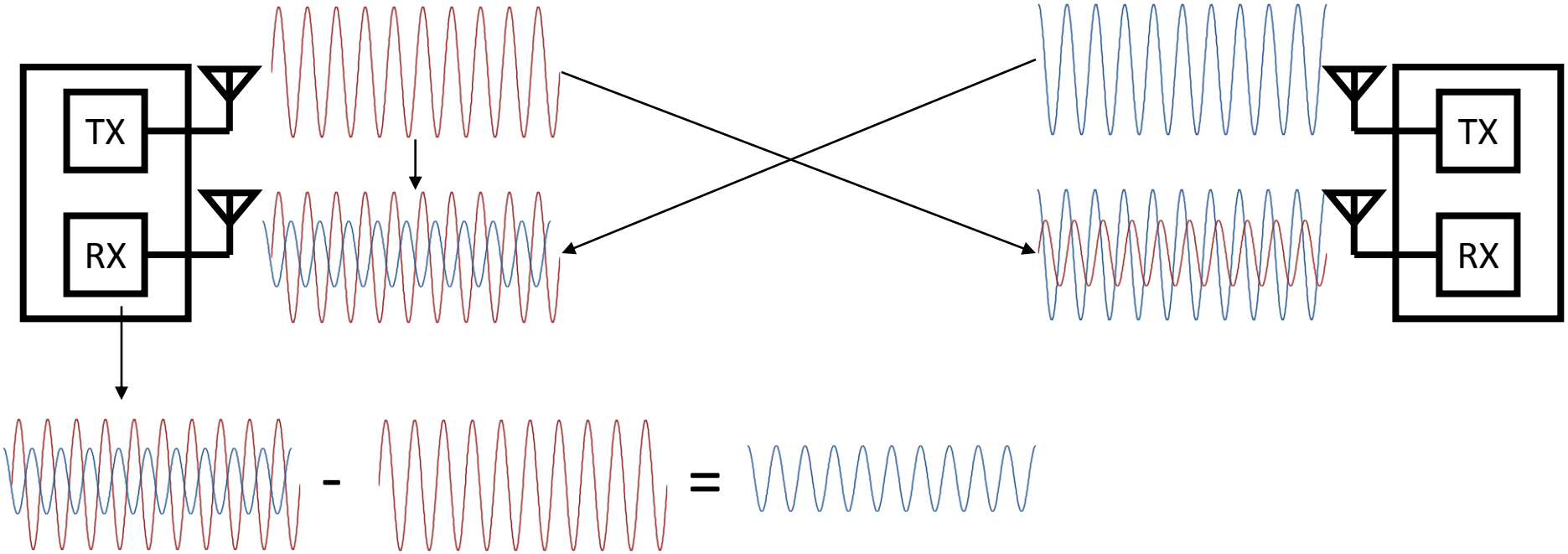
無線全二重通信 2/3

- 無線半二重通信で同時に送信と受信をした場合
  - 自身の送信する電波の方が他の端末が送信する電波より電力が大きい



無線全二重通信 3/3

- 無線全二重通信
  - 自身の送信は既知であるため受信信号から取り除く



Full Duplex MAC

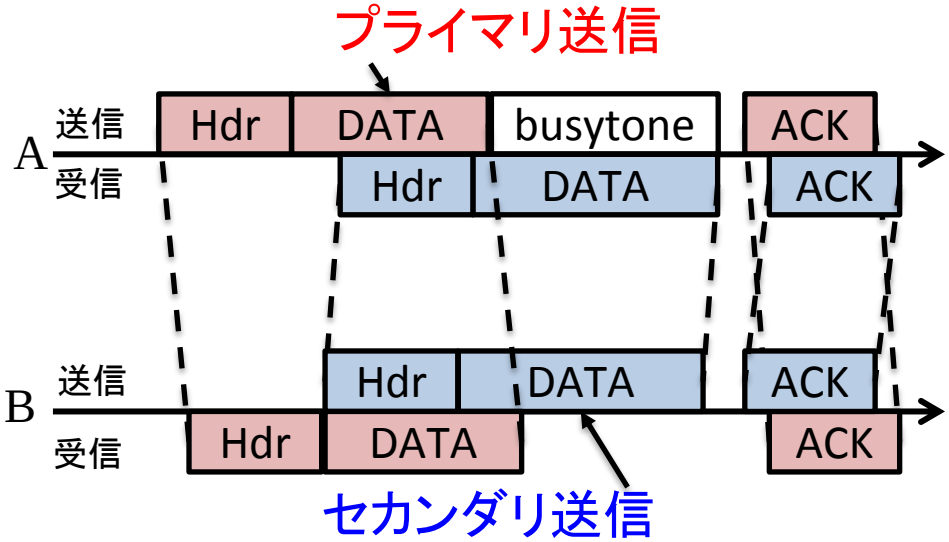
- M. Jain, et al: 非同期型FD-MAC [1]
- A. Sahai, et al: 同期型FD-MAC [2]

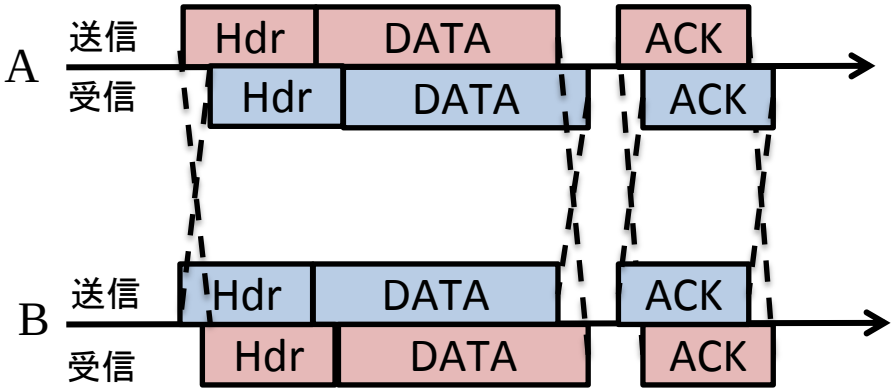
1) 非同期型

- プライマリ送信とセカンダリ送信
- クロック同期不必要

2) 同期型

- 同時に2つの送信
- クロック同期必要





非同期型のFD-MACを実装

[1] Mayank Jain, et al., **Practical, Real-time, Full Duplex Wireless**, ACM MobiCom 2011.

[2] A. Sahai, et al., **Pushing the Limits of Full-duplex: Design and Real-time Implementation**, Rice University Technical report 2011.

双方向・中継全二重通信

- 双方向全二重通信
  - アクセスポイントとノードAは互いのデータを受信
  - プライマリ送信の宛先に指定されているノードがセカンダリ送信の送信元ノード

- 中継全二重通信
  - ノードBはノードAの信号を受信しつつノードCに送信
  - プライマリ送信の送信元ノードはセカンダリ送信の送信元ノードを指定

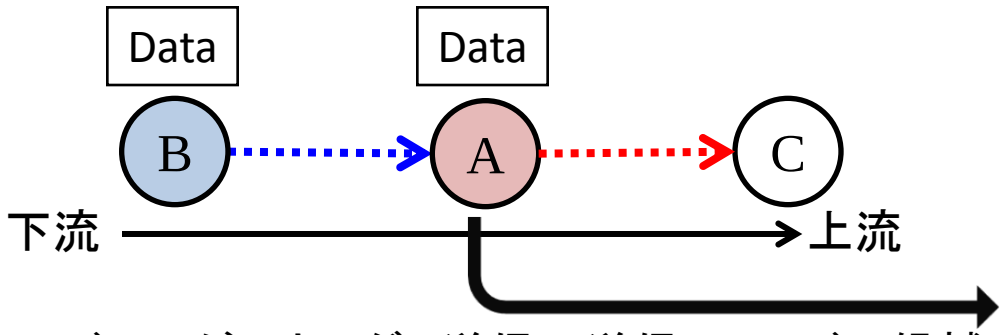
双方向・中継全二重通信に対応

RFD-MAC概要

- 中継全二重通信のMACプロトコル
  - プライマリ・セカンダリ送信を利用
  - 非同期の中継全二重通信
- セカンダリ送信の送信元ノードの選択
  - 隣接ノードテーブルを基にセカンダリ送信の送信元ノードを決定
  - MACヘッダのaddress4にセカンダリ送信の送信元アドレスを含ませることでセカンダリ送信の送信元ノードに通知
- 隣接ノードテーブル
  - 隣接ノードの通信を盗聴して隣接ノートテーブルを更新
  - 3つのフィールドを持つ
    - Address: 隣接ノードのMACアドレス
    - HasFrames: フレームを持っている場合1，そうでない場合0
    - NextHop: 上流ノードの場合1，そうでない場合0

隣接ノードテーブル

- セカンダリ送信の送信元ノードの選択に利用



ノードB，Cがセカンダリ送信の送信元ノードの候補
どちらかを選択しなければならない

ノードAの隣接ノードテーブル

Node Address	Has Frames	Next Hop
B	1	0
C	0	1

- 優先順位
  1. HasFrames=1 & NextHop=1
  2. HasFrames=1 & NextHop=0
  3. HasFrames=0 & NextHop=0
  4. HasFrames=0 & NextHop=1

HasFrames
・DATAとACK送信時にMACヘッダのMoreDataビットに値を代入
・MoreDataビットには送信可能なデータがある場合は1，ない場合は0を代入

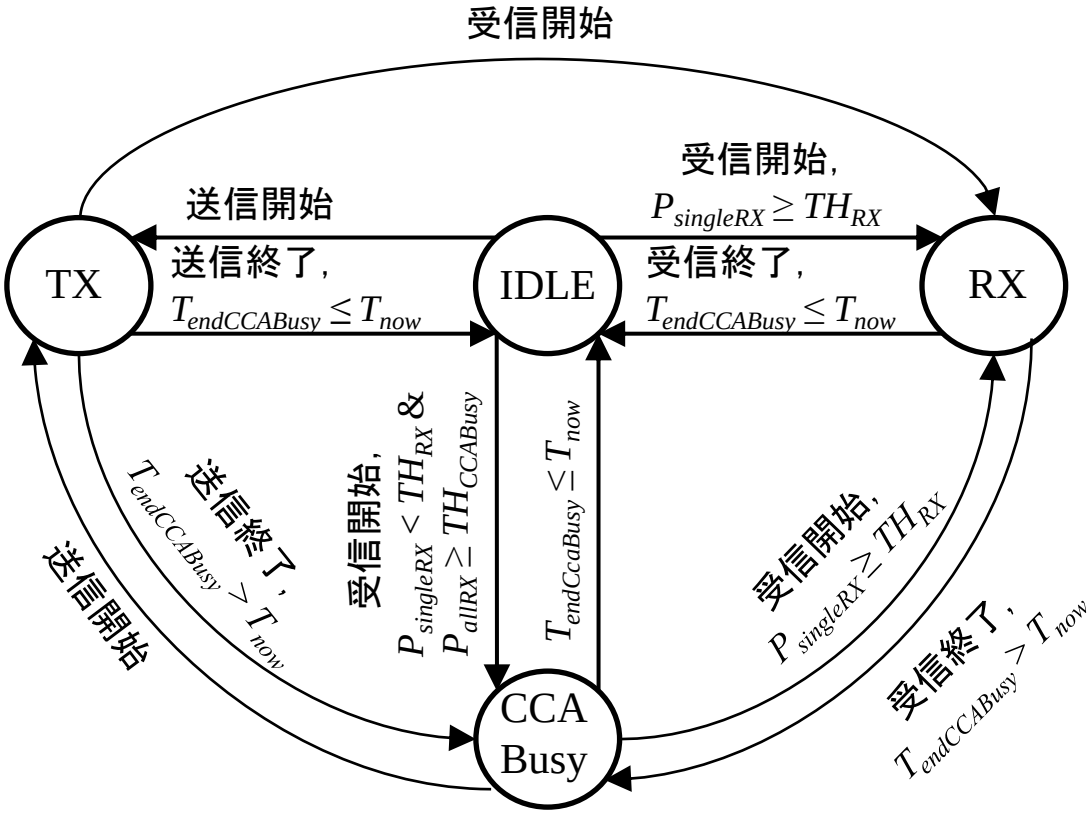
fdwifiモジュール ~実装した機能~

- 全二重通信
  - 新しいステータスとしてFD（Full-Duplex）状態を追加
  - フレームの受信開始時にヘッダ受信イベントのスケジューリング処理を追加
  - ヘッダとペイロードでPERを計算できるように電力イベントの変更
  - ヘッダ受信イベントを追加
  - セカンダリ送信機能の追加
  - Busytoneの作成と送信機能の追加

- 隣接ノードテーブル
  - DATA，ACK送信時にMACヘッダのMoreDataビットの設定処理を追加
  - DATA受信時に隣接ノードテーブル更新処理を追加
  - セカンダリ送信の送信元ノードの決定アルゴリズムの追加

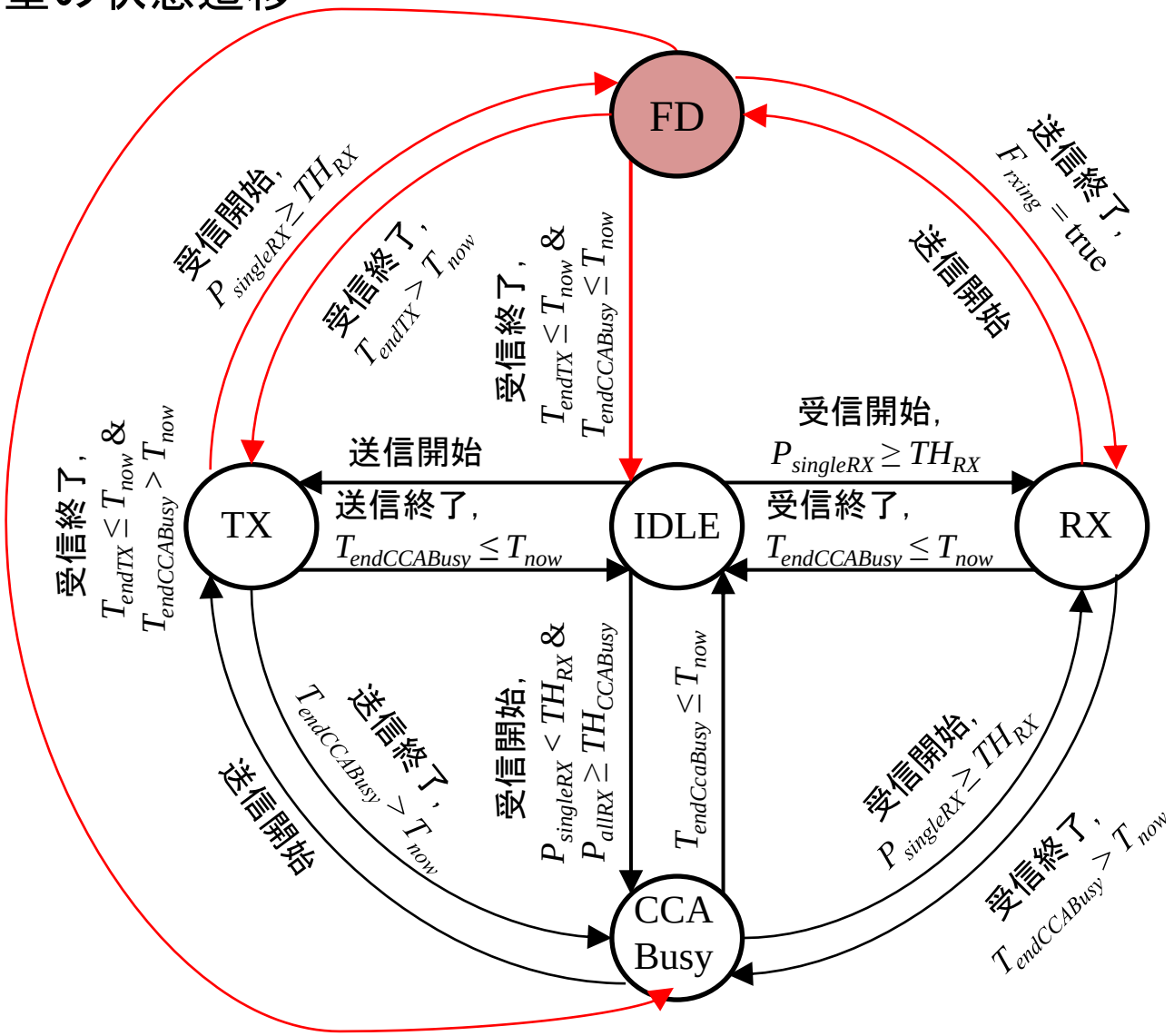
全二重通信
~FD状態の追加~

- 半二重通信の状態遷移

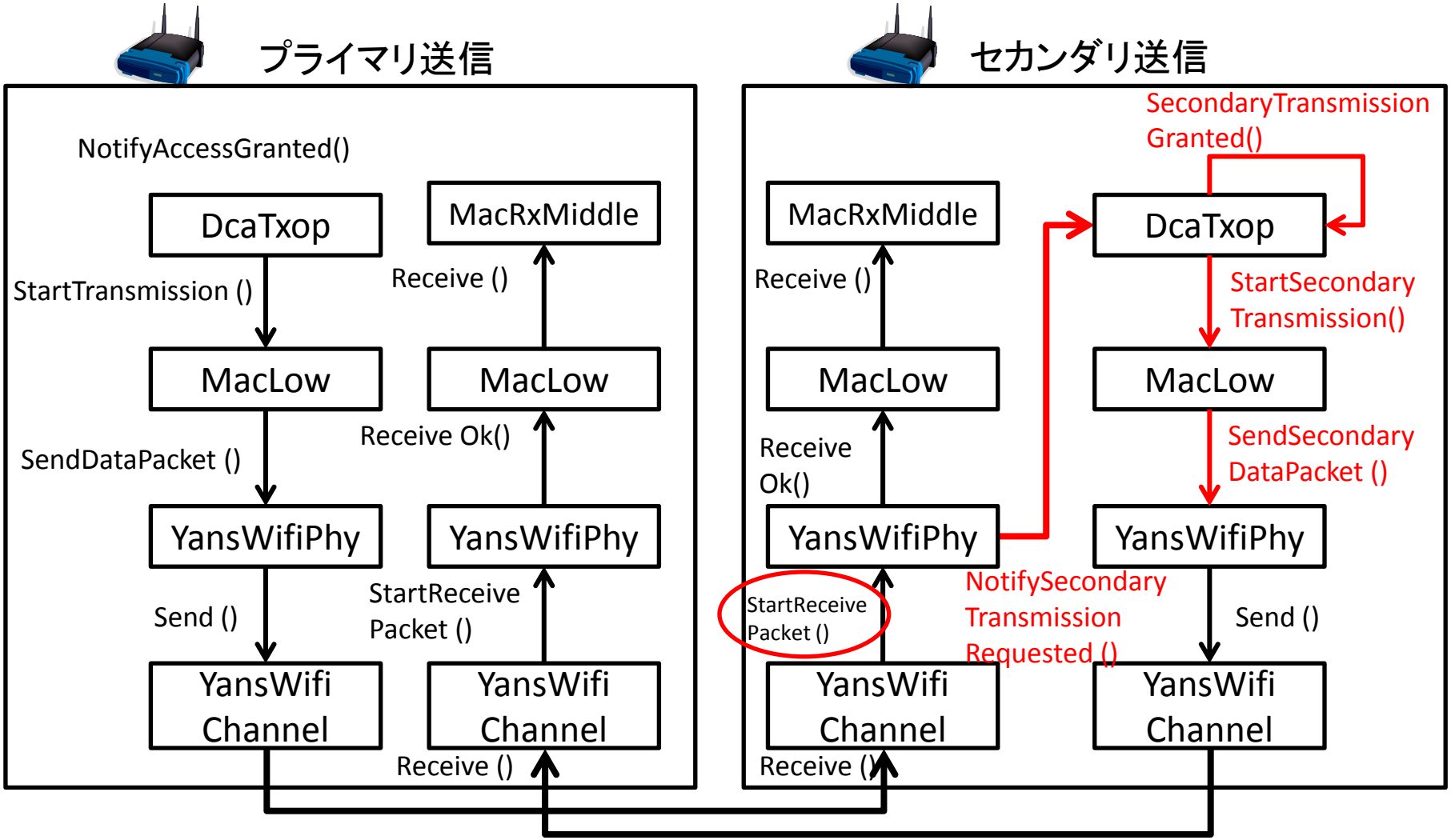


全二重通信
~FD状態の追加~

- 全二重の状態遷移



プライマリ・セカンダリ送信全体像



セカンダリ送信のStartReceivePacket ()からの流れを追う

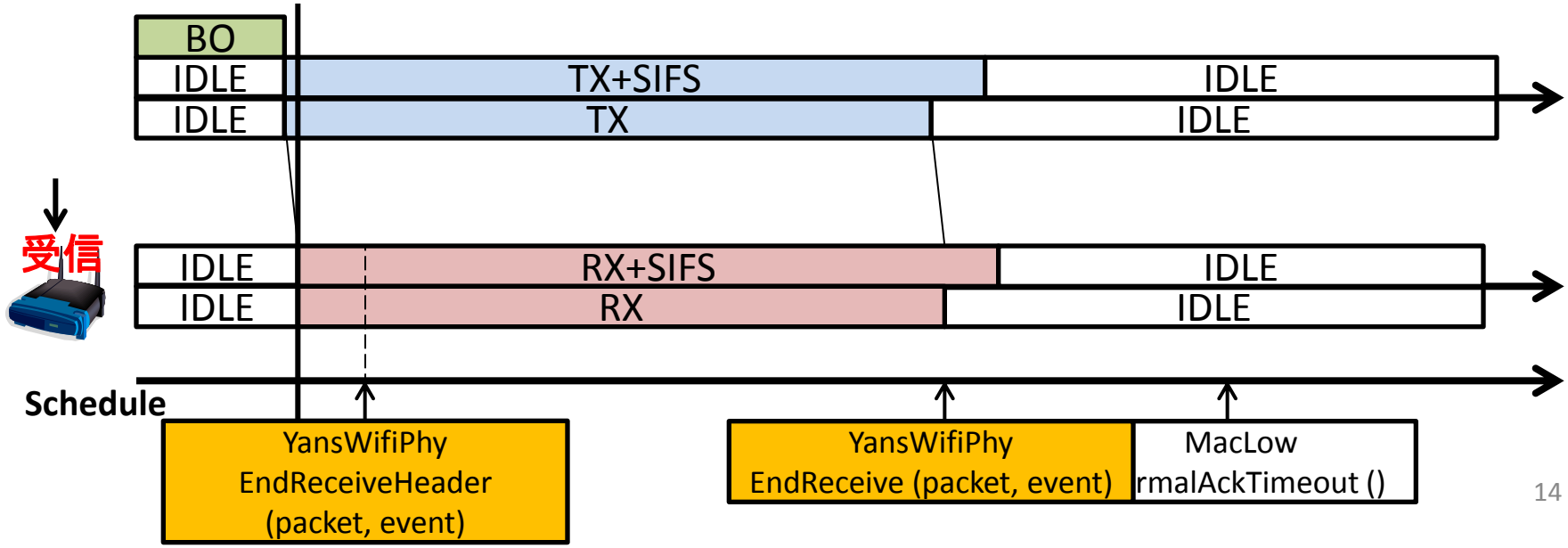
全二重通信

～フレームの受信開始時にヘッダ受信イベントのスケジューリング処理を追加～

YansWifiPhy

StartReceivePacket (packet, rxPowerDbm, txVector, preamble)

- 仕様：パケット受信開始時の処理にヘッダ受信開始時の処理を追加
  1. rxDuration計算: パケットサイズ，txVector，preambleから受信時間を計算
  2. 物理層の状態がCCA_BUSYまたはIDLE状態の場合かつ受信電力が受信スレッショルドを上回っていた場合
     1. WifiPhyStateHelperのSwitchRx()関数を呼び出し状態を変える
     2. EndReceiveHeader (packet, eventHdr)をスケジューリング
     3. EndReceive (packet, event)をスケジューリング

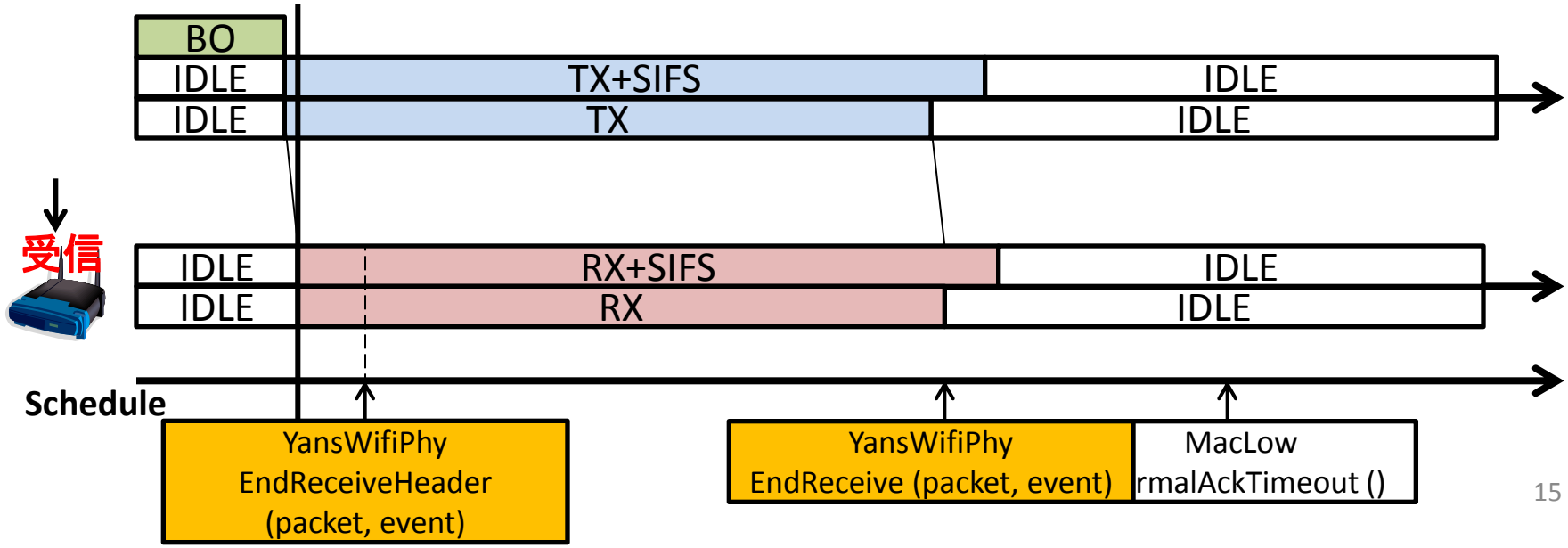


全二重通信
~ヘッダとペイロードでPERを計算できるように電力イベントの変更~

YansWifiPhy

StartReceivePacket (packet, rxPowerDbm, txVector, preamble)

- 仕様：パケット受信開始時の処理にヘッダ受信開始時の処理を追加
  1. rxDuration計算: パケットサイズ，txVector，preambleから受信時間を計算
  2. InterferenceHelperにHeader受信時のheaderEventを追加
  3. InterferenceHelperにPayload受信時のpayloadEventを追加
  4. 物理層の状態がCCA_BUSYまたはIDLE状態の場合かつ受信電力が受信スレッショルドを上回っていた場合
     1. WifiPhyStateHelperのSwitchRx()関数を呼び出し状態を変える
     2. EndReceiveHeader (packet, eventHdr)をスケジューリング
     3. EndReceive (packet, event)をスケジューリング

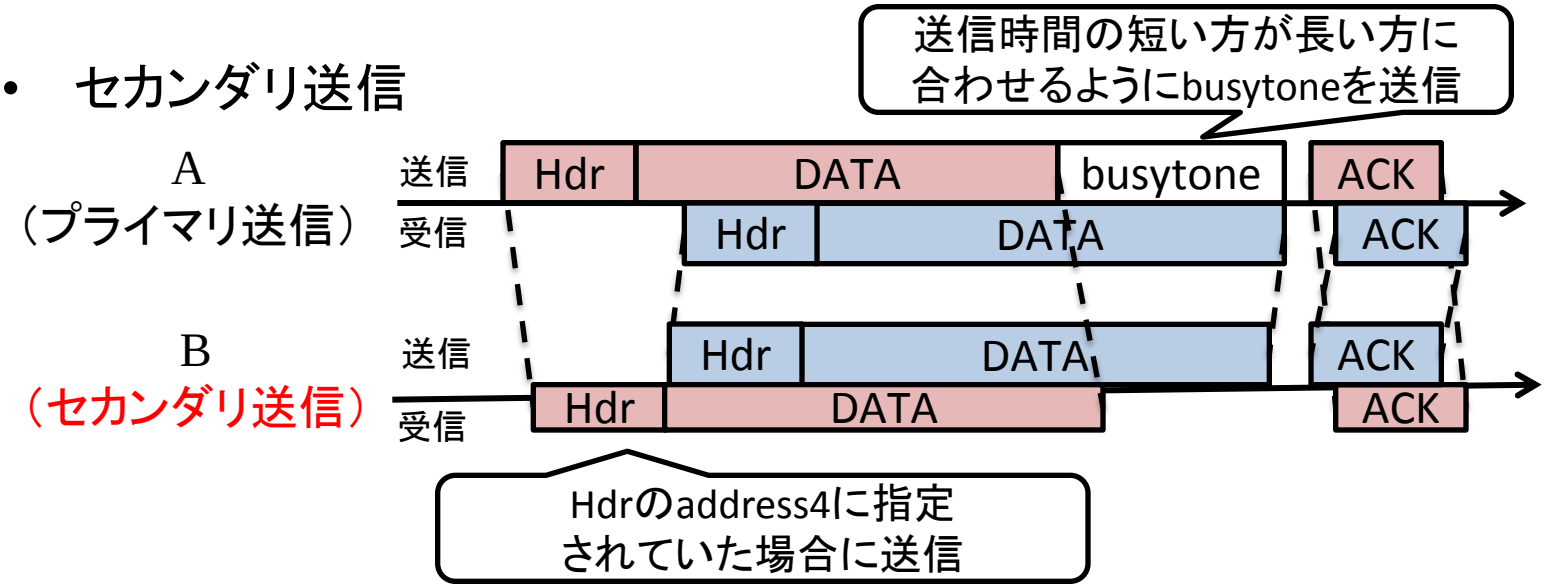


全二重通信
~ヘッダ受信イベントを追加（全体像）~

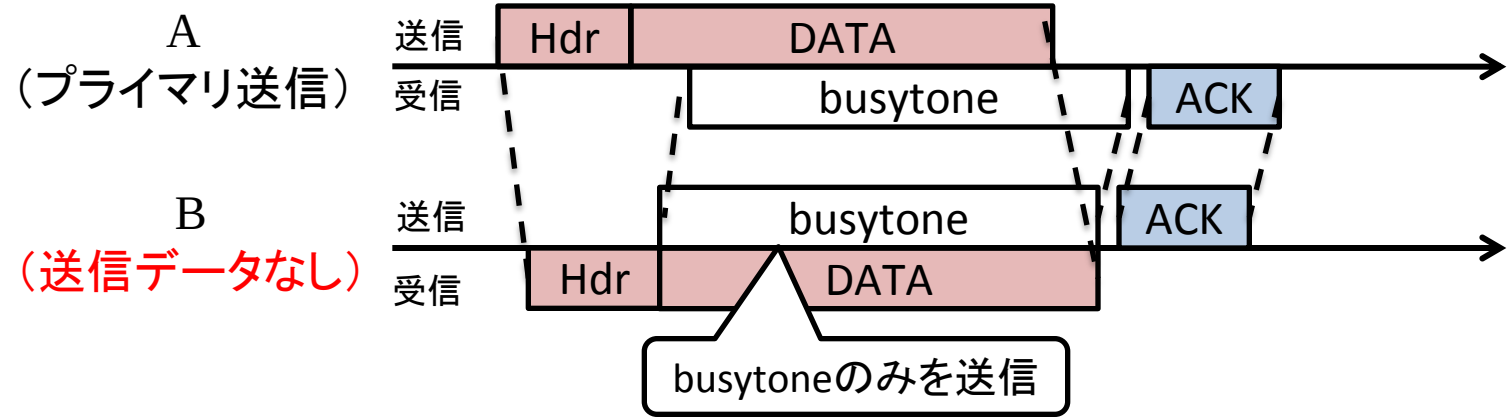
- 受信ノードによって処理が異なる
  - 全ノード共通
    - プリアンブル，ヘッダ長，受信電力，変調方式等からPERを算出してパケットを受信可能か判定
  - セカンダリ送信の送信元ノードが受信した場合
    - 送信可能な状態ならばBusytoneまたはセカンダリ送信を開始
  - プライマリ送信の送信元ノードが受信した場合
    - プライマリ送信とセカンダリ送信の送信終了時間を比較してプライマリ送信の送信時間が短い場合にプライマリ送信の送信時間を延長
  - その他のノードが受信した場合
    - 特になし

全二重通信
~ヘッダ受信イベントを追加（セカンダリ送信orBusytone）~

- セカンダリ送信



- Busytone送信



全二重通信
~ヘッダ受信イベントを追加（セカンダリ/Busytone送信の条件）~

- セカンダリ/Busytone送信の発生条件

送信の種類 / MAC層の状態	プライマリ送信	セカンダリ/Busytone送信
IDLE	○	○
送信中	×	×
受信中	×	○
キャリアセンス中	×	○
Backoff待機中	×	○
AckTimeout待ち	×	×
CTSの返信待ち	×	×

- セカンダリ/Busytone 選択

DcaTxopのキューが空	DcaTxopのm_currentPacketが0	セカンダリ/Busytone送信
×	×	セカンダリ
×	○	セカンダリ
○	×	セカンダリ
○	○	Busytone

全二重通信
~ヘッダ受信イベントを追加（セカンダリ送信の送信元ノードが受信）~

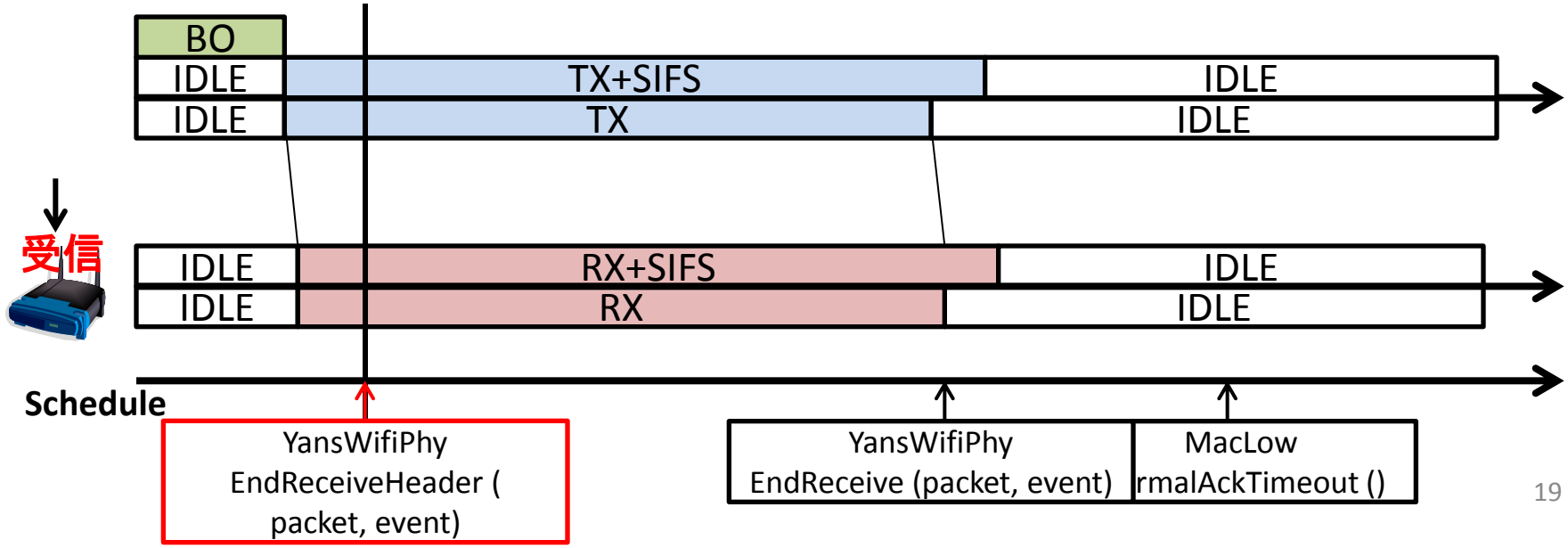
YansWifiPhy

EndReceiveHeader (packet, event)

- 仕様：ヘッダを受信時の処理
  1. eventを基にSNRとPSRを計算して受信判定
  2. SecondaryTagがpacketに付加されていない場合（プライマリ送信を受信）
     1. MACヘッダのaddress4に自身が指定されている場合（セカンダリ送信の送信元に指定）
        1. DcfTxopのNotifySecondaryTransmissionRequested () or YansWifiPhyのSendBusyTone ()を呼び出す

↓

DcfTxop::NotifySecondaryTransmissionRequested ()

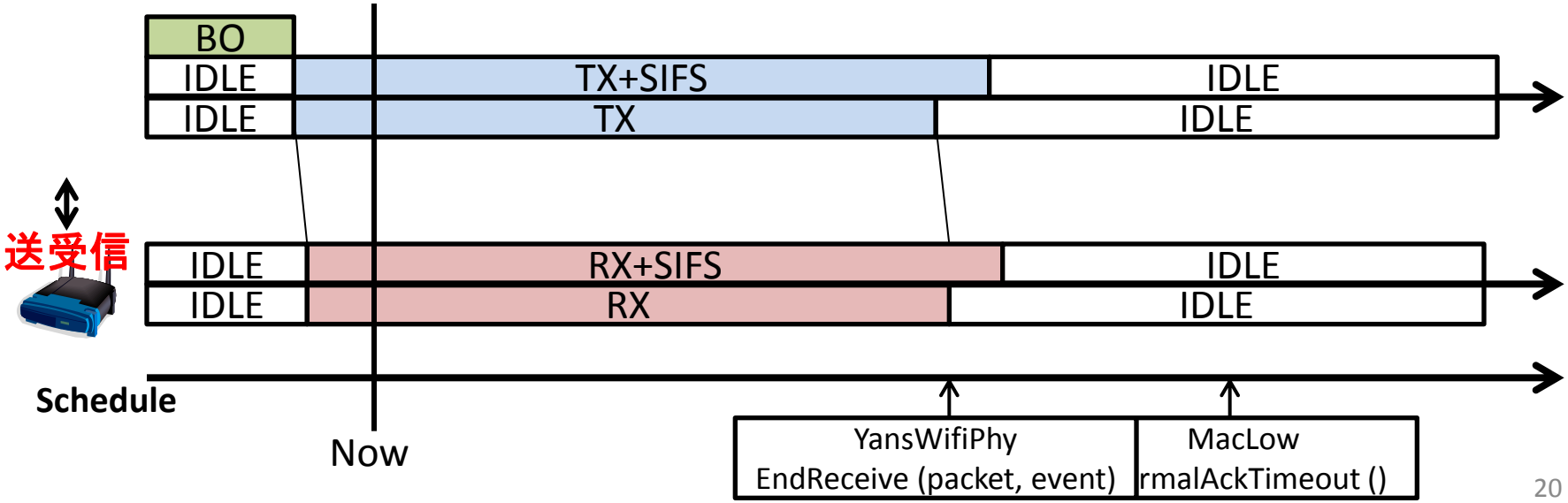


全二重通信
~ヘッダ受信イベントを追加（セカンダリ送信の送信元ノードが受信）~

DcaTxop

NotifySecondaryTransmissionRequested ()

- 仕様：セカンダリ送信可能か判定，可能ならば送信する
  1. m_currentPacketが存在するまたはキューが空でない場合
     1. m_accessRequestedをtrueにする
     2. セカンダリ送信の発生条件を満たしている
        1. SecondaryTransmissionGranted ()を呼び出す



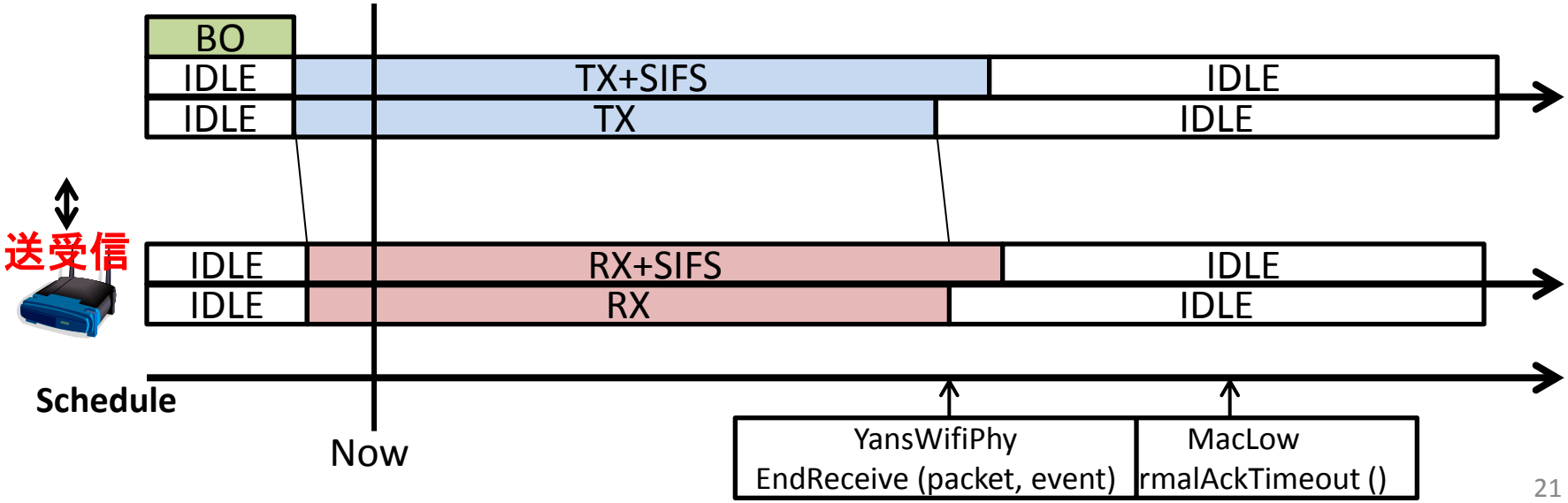
全二重通信
~ヘッダ受信イベントを追加（セカンダリ送信の送信元ノードが受信）~

DcfTxop

SecondaryTransmissionGranted () ***※NotiryAccessGranted ()とほとんど一緒***

- 仕様：Mac層のパラメータを生成して，MacLowにパケットを渡す
  1. m_currentPacketにパケットを代入
  2. MacLowTransmissionParameter paramを生成（RTS，ACKがいるかどうかなどの情報が入る）
  3. MacLowのStartSecondaryTransmission (m_currentPacket, &m_currentHdr, params, m_transmissionListener)を呼び出す

MacLow::StartSecondaryTransmission (packet, hdr, params, listener)

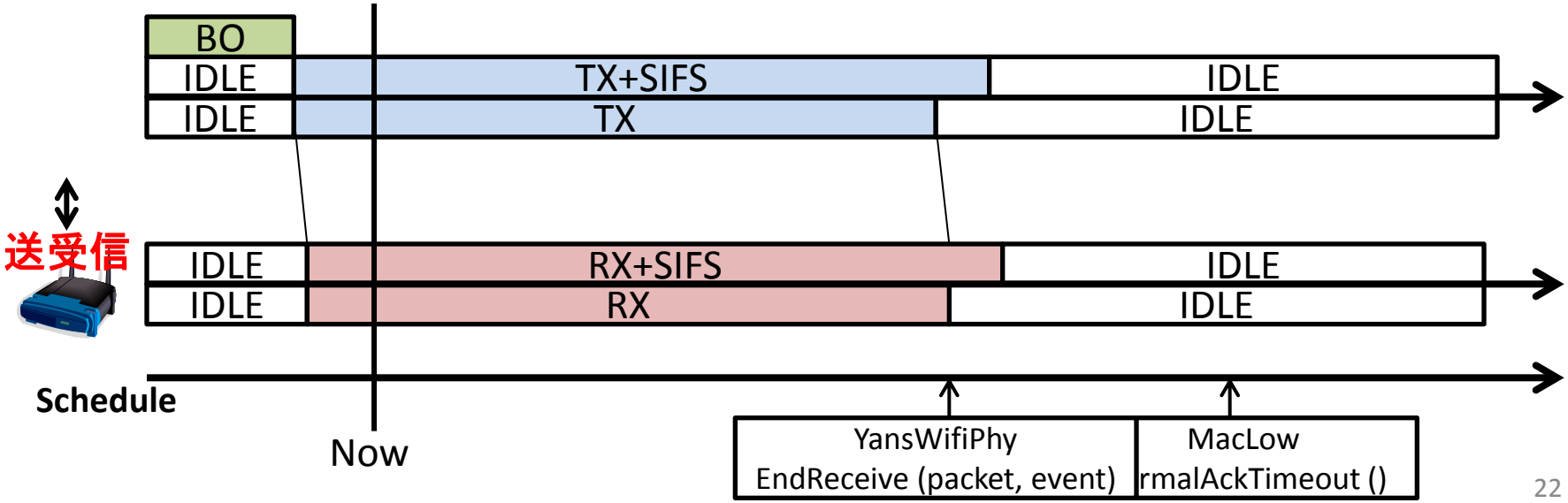


全二重通信
~ヘッダ受信イベントを追加（セカンダリ送信の送信元ノードが受信）~

MacLow

StartSecondaryTransmission (packet, hddr,params, listener) ※ ***StartTransmission (packet, hdr, params, listener) 関数とほとんど一緒***

- 引数
  - packet: 送信パケット
  - hdr: 送信ヘッダ
  - params: MACのパラメータ（DATA，ACK，RTS，CTSを判別するのに利用）
  - listener: DcaTxopのリスナー（ACKやCTSのタイムアウトを通知するのに利用）
- 仕様：メンバ変数に引数を登録
  1. 引数で与えられた変数をメンバ変数に登録
  2. SendSecondaryDataPacket ()を呼び出す



全二重通信
~ヘッダ受信イベントを追加（セカンダリ送信の送信元ノードが受信）~

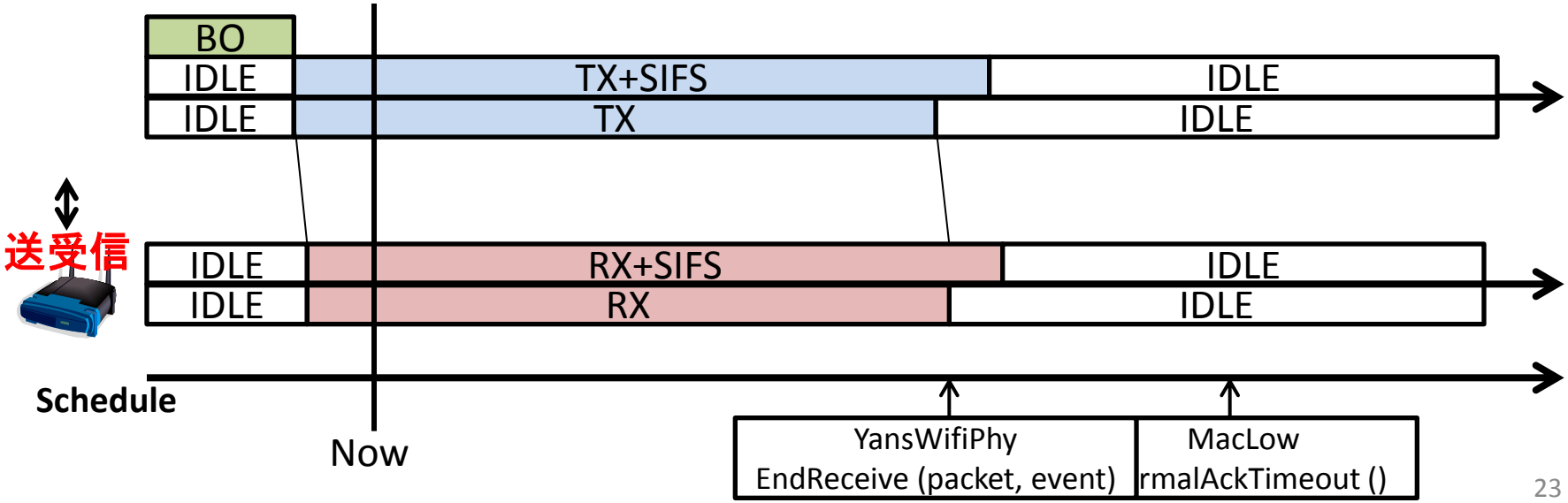
MacLow

SendSecondaryDataPacket () ***※SendDataPacket ()関数にbusytone分のデータを付加SecondaryTagを付加***

- 仕様：セカンダリ送信のパケットを作って物理層に送信
  1. SendDataPacket ()関数と同様の処理（割愛）
  2. SecondaryTagをm_currentPacketに付加（ヘッダ受信時にセカンダリ送信の判断に利用）
  3. BusytoneTagをm_currentPacketに付加（データサイズを変更するために利用）
  4. ForwardDown (m_currentPacket, m_currentHdr, dataTxVector, preamble)を呼び出す

⬇

ForwardDown()関数以降はwifiモデルのデータ送信と同様に処理される



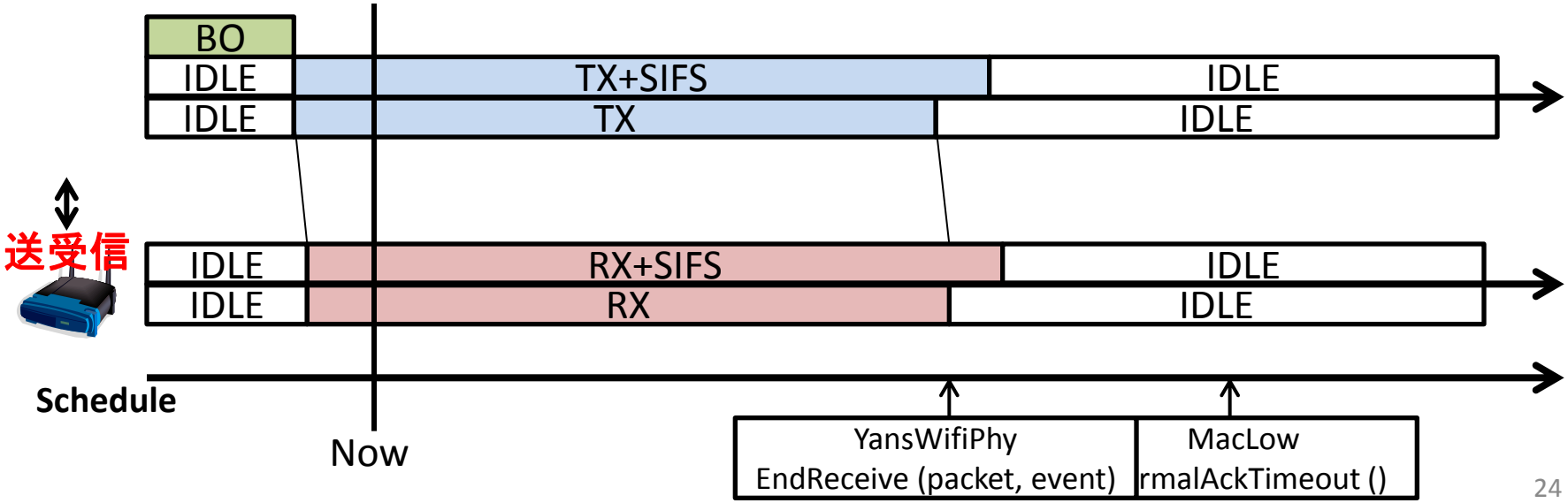
MacLow：送信

MacLow

ForwardDown (packe, hdr, txVector, preamble)

- 仕様：物理層のSendPacket ()関数を呼び出す
  1. 物理層のSendPacket (packet, txVector.GetMode(), preamble, txVector)を呼び出す

YansWifiPhy::SendPacket (packet, hdr, txMode, preamble, txVector)



YansWifiPhy : 送信

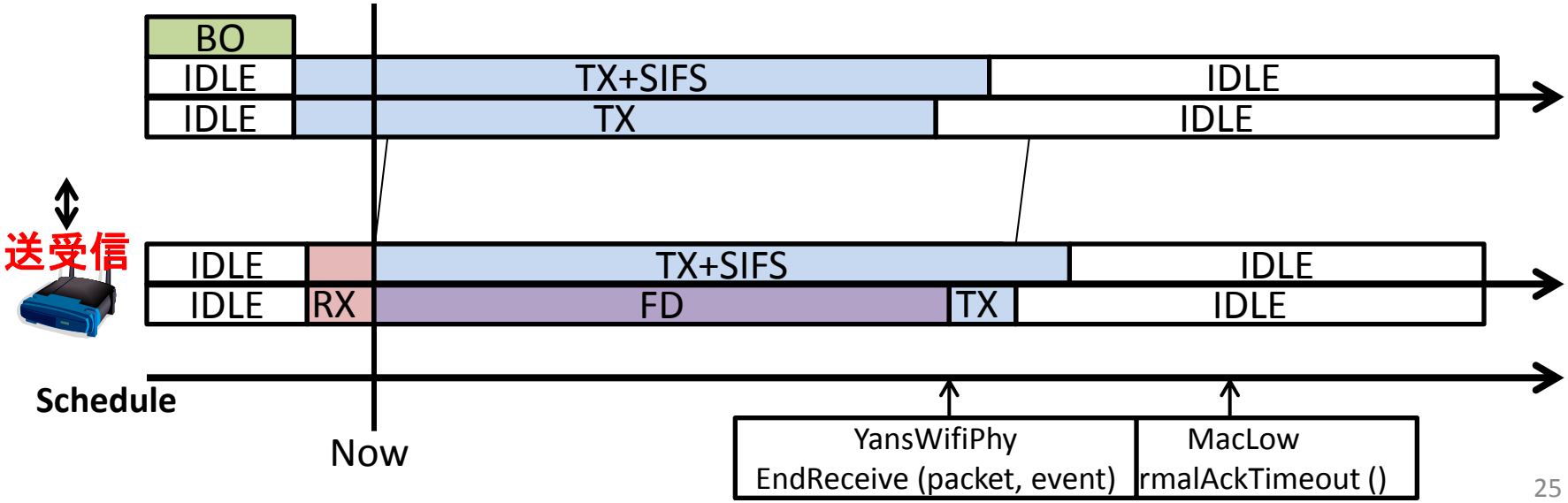
YansWifiPhy

SendPacket (packe, hdr, txMode, preamble, txVector)

- 引数
  - txMode: バンド幅，コードレート，変調方式などの情報を持つ
- 仕様 : WifiPhyStateHelperのステータスを変化，チャネルクラスにのSend ()関数を呼び出す
  1. WifiPhyStateHelperのSwitchTx ()関数を呼び出してステータスを変化
  2. チャネルクラスのSend (this, packet, GetPowerDbm (txVector.GetTxPowerLevel()) + m_txGainDb, txVector, preamble)を呼び出す

     送信ゲイン

YansWifiChannel::Send (sender, packet, txPowerDbm, txVector, preamble)

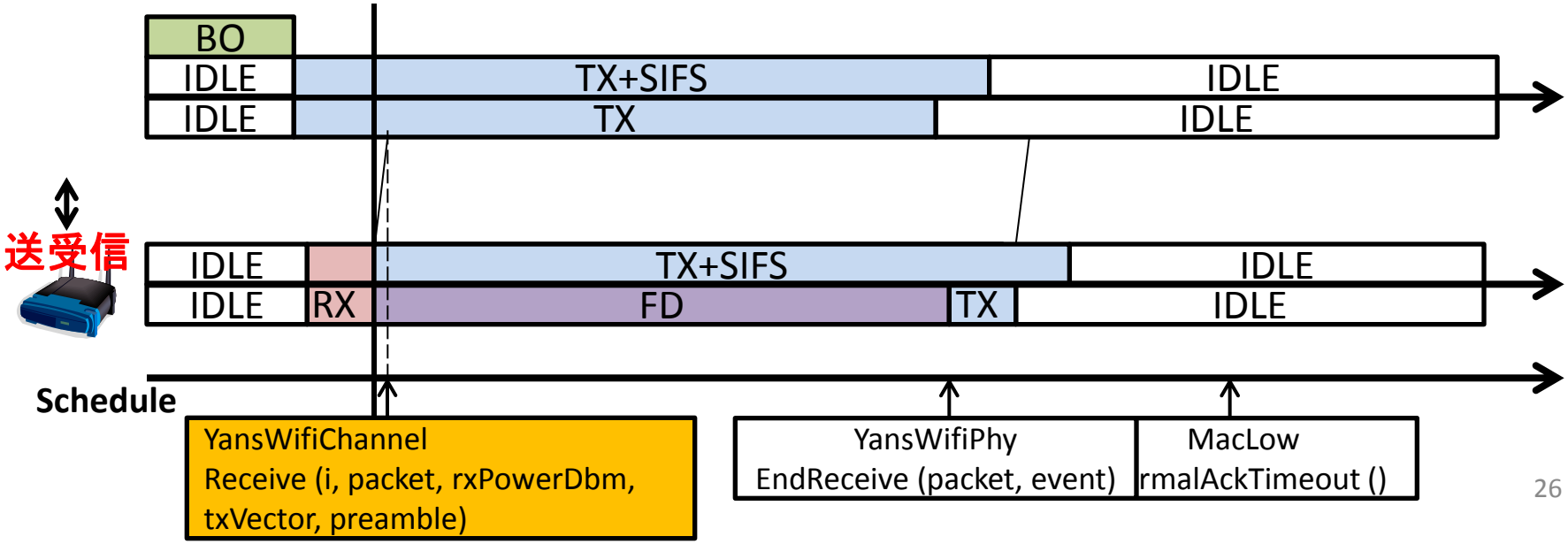


YansWifiChannel：送信

YansWifiChannel

Send (sender, packet, txPowerDbm, txVector, preamble)

- 引数
  - txPowerDbm: 送信電力
- 仕様：遅延，伝搬損失の算出，算出した結果に基づきReceive()関数をスケジューリング
  1. senderMobilityを取得：senderからモビリティ情報（位置情報等）を取得
  2. receiverMobilityを取得：受信ノードのモビリティ情報を取得
  3. 取得したモビリティ情報から遅延（delay），受信電力（rxPowerDbm）を計算
  4. Receive（受信と送信で共通のインデックス，パケット，rxPowerDbm，txVector，preamble）関数を伝搬遅延後にスケジューリング



YansWifiChannel：受信

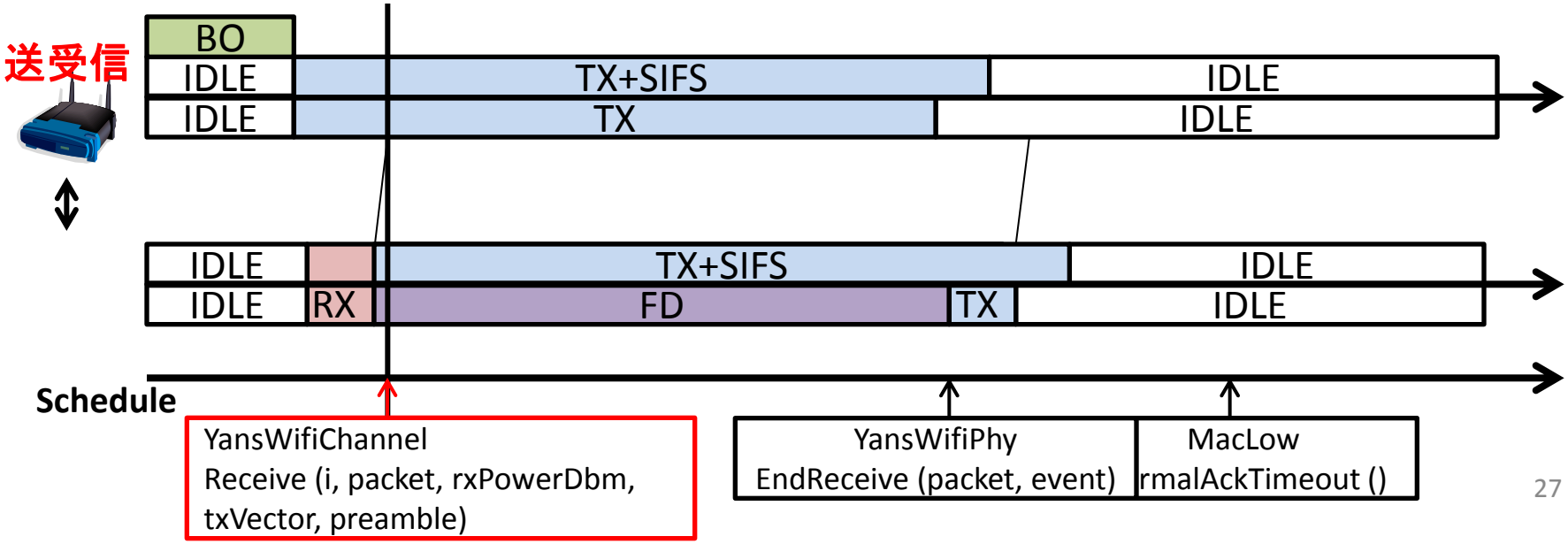
YansWifiChannel

Receive (i, packet, rxPowerDbm, txVector, preamble)

- 仕様：物理層の受信処理
  1. 共通の番号iを用いてStartReceivePacket (packet, rxPowerDbm, txVector, preamble)を呼び出す

⬇

YansWifiPhy::StartReceivePacket (packet, rxPowerDbm, txVector, preamble)

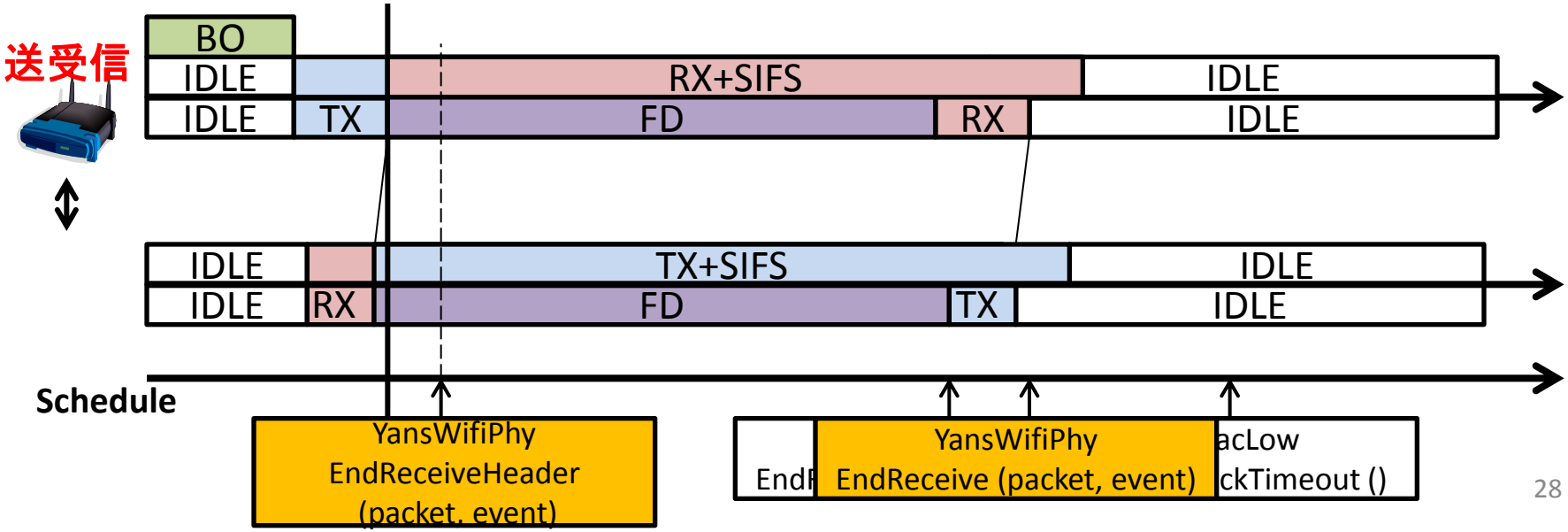


全二重通信
~ヘッダ受信イベントを追加(プライマリ送信の送信元ノードが受信)~

YansWifiPhy

StartReceivePacket (packet, rxPowerDbm, txVector, preamble)

- 仕様:パケット受信開始時の処理にヘッダ受信開始時の処理を追加
  1. rxDuration計算: パケットサイズ, txVector, preambleから受信時間を計算
  2. InterferenceHelperにHeader受信時のeventHdrを追加
  3. InterferenceHelperにPayload受信時のeventを追加
  4. 物理層の状態がCCA_BUSYまたはIDLE状態の場合かつ受信電力が受信スレッショルドを上回っていた場合
     1. WifiPhyStateHelperのSwitchRx()関数を呼び出し状態を変える
     2. EndReceiveHeader (packet, eventHdr)をスケジューリング
     3. EndReceive (packet, event)をスケジューリング

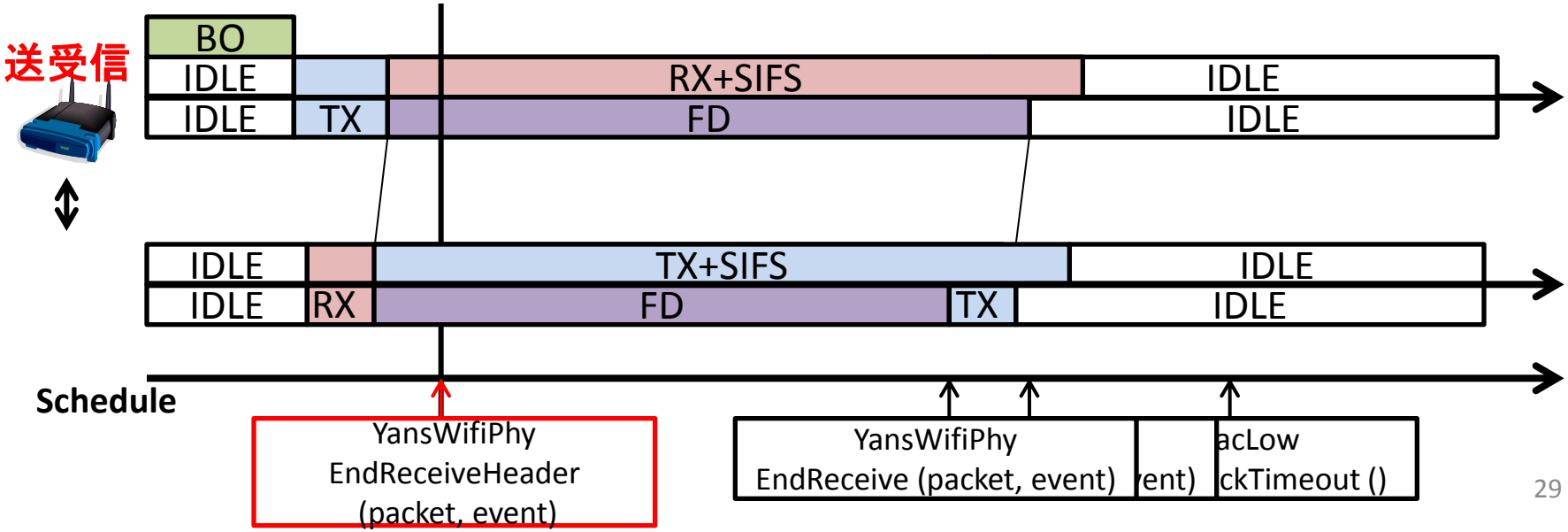


全二重通信
~ヘッダ受信イベントを追加(プライマリ送信の送信元ノードが受信)~

YansWifiPhy

EndReceiveHeader (packet, event)

- 仕様:ヘッダを受信時の処理
  1. eventを基にSNRとPSRを計算して受信判定
  2. SecondaryTagがpacketに付加されている場合(セカンダリ送信を受信)
     1. MACヘッダのaddress4に自身が指定されている場合
        1. プライマリ送信の終了時間 < セカンダリ送信の終了時間の場合
           1. プライマリ送信の終了をセカンダリ送信の終了時間に合わせて終了するように送信時間を変更
           2. プライマリ送信を受信しているノードの受信時間を変更するためYansWifiChannelのNotifyPostponeSend (sender, packet, powerDbm, txVector, preamble, rxEndTime)を呼び出す

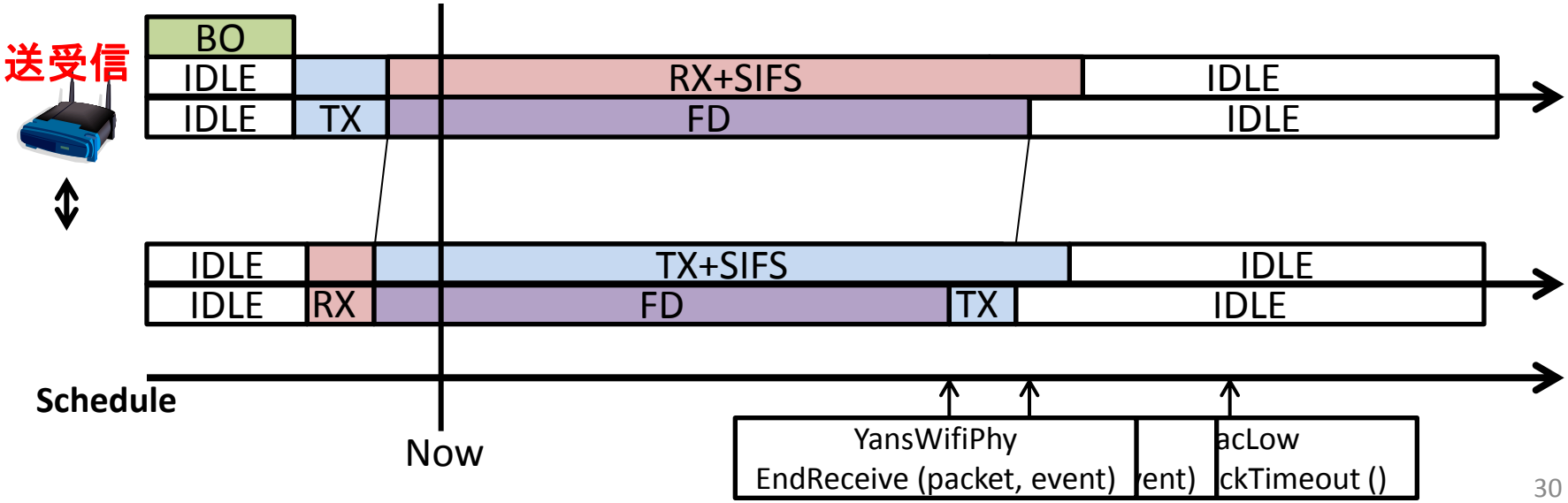


全二重通信
~プライマリ送信の延期を隣接ノードに通知~

YansWifiChannel

NotifyPostponeSend (sender, packet, txVector, preamble, rxEndTime)

- 仕様：EndReceive (packet, event)を再スケジューリングを受信ノードに通知
  1. YansWifiPhyのNotifyChangeEndRecieve (受信と送信で共通のインデックス, パケット, txVector, preamble, rxEndTime + delay) 関数を現在時間にスケジューリング



全二重通信
~プライマリ送信の延期を隣接ノードに通知~

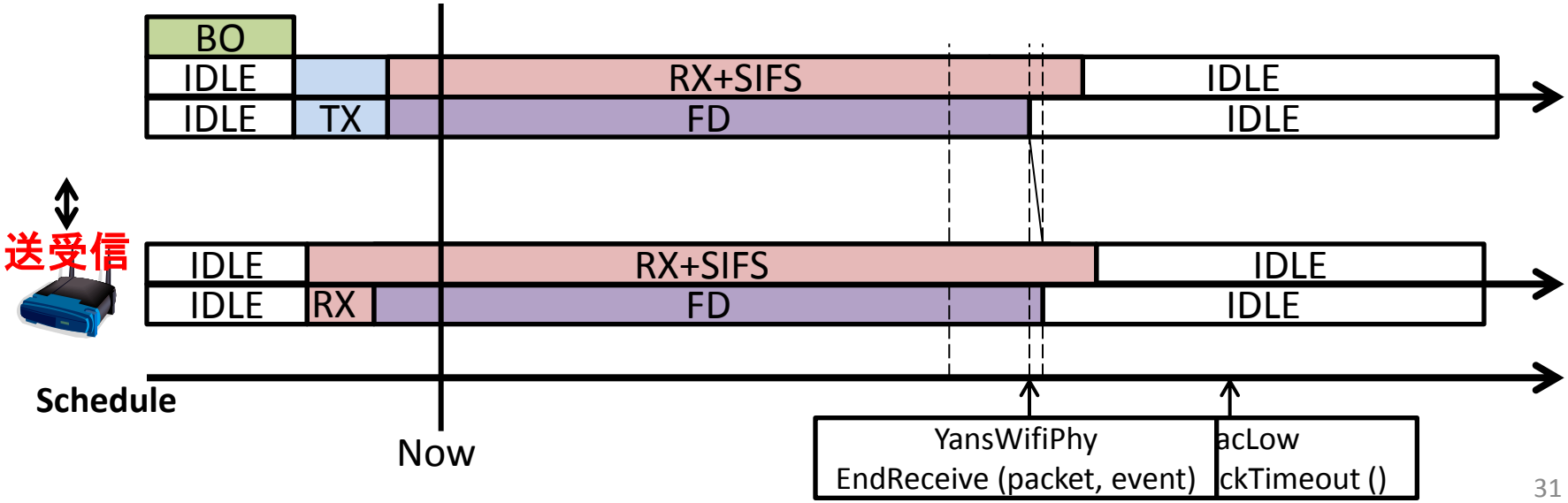
YansWifiChannel

NotifyChangeEndRecieve (**i, packet, txVector, preamble, rxEndTime**)

- 仕様:
  1. 共通の番号iを用いてYansWifiPhyのNotifyChangeEndReceive (I, packet, txVector, preamble, rxEndTime)をスケジューリング

⬇

YansWifiPhy::NotifyChangeEndReceive (i, packet, txVector, preamble, rxEndTime)



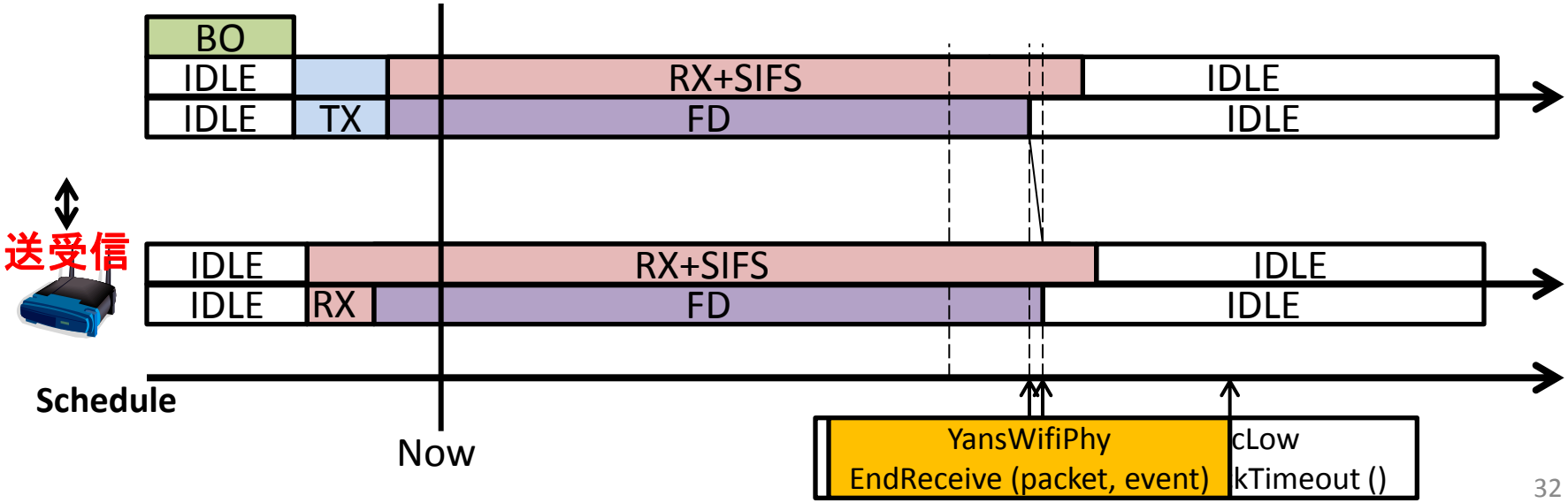
全二重通信
~プライマリ送信の延期を隣接ノードに通知~

YansWifiPhy

NotifyChangeEndRecieve (**packet, txVector, preamble, rxEndTime**)

- 仕様：EndReceive ()関数の延期処理
  1. EndReceive (packet, event)のスケジューリングをキャンセル
  2. rxEndTimeにEndReceive(packet, event)が発生するようにスケジューリング

EndReceive ()関数の処理とACKの返信処理はwifiモジュールと同様に処理

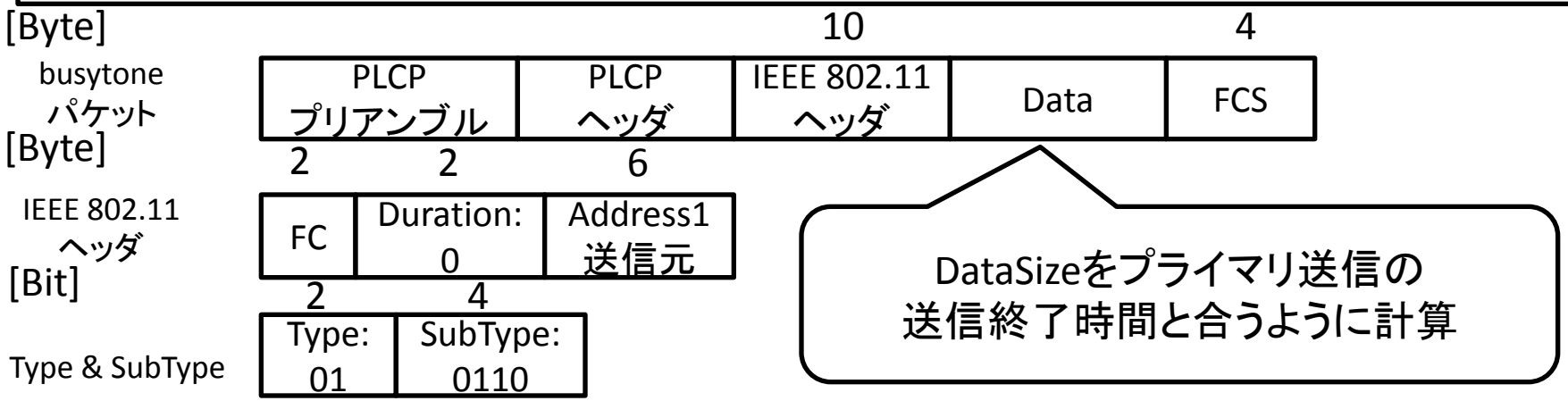


物理層
~ Busytoneの作成と送信~

YansWifiPhy

SendBusyTone (packet, txVector)

- 引数
  - packet: 受信パケット
  - txVector: 受信txVector
- 仕様：Busytoneのみのパケットを生成して送信
  1. busytoneHdr作成: ヘッダのタイプをWIFI_MAC_CTL_BUSYを指定
  2. ownTxVectorの作成: dammyPacket（一時的に作成したパケット）とbusytoneHdrから生成
  3. pktSizeを求める: packetのサイズから送信するBusytoneのサイズを計算
  4. busytoneパケット生成:pktSizeからbusytoneパケットを生成して，busytoneHdrとFCSを付加
  5. preambleを生成: WIFI_PREAMBLE_LONGを設定
  6. SendPacket (busytone, ownTxVecotr.GetMode(), preamble, ownTxVector)を呼び出して送信



隣接ノードテーブルの実装

~DATA，ACK送信時にMACヘッダのMoreDataビットを設定~

- SendDataPacket (), SendSecondaryDataPacket (), SendAckAfterData (…)に以下の機能を実装

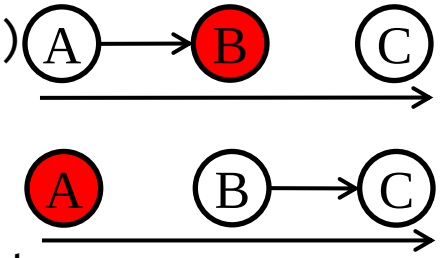
```
if (DcaTxopのQueueが空でない)
  macヘッダのMoreDataビット=1
else
  macヘッダのMoreDataビット=0
```

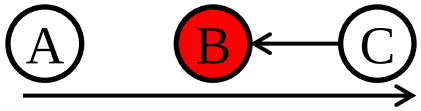
隣接ノードテーブルの実装

~ DATA受信時に隣接ノードテーブル更新処理~

- ReceiveOk (…)に以下の機能を追加

```
if (DATAフレームを受信)
  if (宛先アドレスが自身のアドレス)
    NextHop = 0
  else
    NextHop = 1
  HasFrames = macヘッダのMoreData
  隣接ノードテーブルを，受信アドレス，NextHop，HasFramesの情報を基に更新
if (ACKフレームを受信)
  if (宛先アドレスが自身のアドレス)
    NextHop = 1
  HasFrames = macヘッダのMoreData
  隣接ノードテーブルを，受信アドレス，NextHop，HasFramesの情報を基に更新
```





隣接ノードテーブルの実装

～セカンダリ送信の送信アルゴリズムの追加～

- SendDataPacket ()に以下の機能を実装

```
MACヘッダのaddress4 = SelectSecondaryTransmissionNode ()
SelectSecondaryTransmissionNode ()
  隣接ノードテーブルのMoreData, HasFramesを基に優先度別に配列Priority[4]を作成
  Priority [0]が最も優先度が高く Priority [3]が最も優先度が低い
  if (Priority[0]のサイズが0より大きい)
    Priority[0]の中からランダムにaddressを返す
  else if (Priority[1]のサイズが0より大きい)
    Priority[1]の中からランダムにaddressを返す
  else if (Priority[2]のサイズが0より大きい)
    Priority[2]の中からランダムにaddressを返す
  else if (Priority[3]のサイズが0より大きい)
    Priority[3]の中からランダムにaddressを返す
  else
    グローバルアドレスを返す (ff:ff:ff:ff:ff:ff)
```

- SendSecondaryDataPacket ()に以下の機能を実装

```
MACヘッダのaddress4 = 送信元ノードのアドレス
```