



MICROCHIP

Regional Training Centers

**Designing a WiFi Solution on
WiFi G Demo Board**

COM-WiFi



Agenda

1. WiFi® G Demo Board Demonstration

- Install MPLAB IDE/X IDE and XC32 1.x compiler
- Pickit Serial Analysis and WiFi G Demo Board
- Demo WiFi SoftAP

2. Embedded Wi-Fi®

- What's that?
- Microchip 802.11 Solution

3. WiFi® G Demo Board—Hand on

- Modify Code to connect Wireless AP

4. Wireless Network Deployment & Wireless AP Feature

- Wireless AP Hand on

5. Overview of 802.11 Networks

- IEEE 802.11 Protocol
- 802.11 Media Access and Frame Control
- 802.11 Security

6. Microchip TCP/IP Stack

- TCPIP WiFi Console App
- WiFi Remote Controller



MICROCHIP

Regional Training Centers

**1. Wi-Fi® G Demo Board
Demonstration**



MICROCHIP

Regional Training Centers

Working with the MRF

Wi-Fi® Development Tools

- **Select a development platform**
 - PICDEM.net™ 2 Board, PIC18 Explorer
 - Explorer 16, Multimedia Development Board
 - PIC32 Starter Kit with I/O Expansion Board



DV164037



DM163024



DM320001



DM320005

- Download Microchip TCPIP Framework
- Plug in MRF Wi-Fi® PICtail™ Board
- **Start with an example**
- **Create your application**



AC164149

Supported Platforms

- **8-bit**

- PICDEM.net™ 2 Board+ AC164149
- PIC18 Explorer + PICtail™ Board



- **16-bit**

- Explorer 16 + AC164149



- **32-bit**

- PIC32 Starter Kit + I/O Expansion Board + AC164149
- Explorer 16 + PIC32 PIM + AC164149
- PIC32 Starter Kit + Multimedia Expansion Board





MICROCHIP

Regional Training
Centers

Quick Start Tools

	Explorer Based Development Board				Stand Alone Evaluation Kit			
Series	MRF		RN		MRF		RN	
Platform	PICtail™/PICtail Plus Board		PICtail™/PICtail Plus Board		Wi-Fi® Comm Demo		Eval Kit	
Module	MRF24WB	MRF24WG	RN131	RN171	MRF24WB	MRF24WG	RN131	RN171
Image								
Part #	AC164136-04	AC164149	RN-131-PICTAIL	RN-171-PICTAIL	DV102411	DV102412	RN-131-EK	RN-171-EK
Availability	NOW	NOW	NOW	NOW	NOW	NOW	NOW	NOW

TCP/IP stacks and Application Demos online at: www.microchip.com/mla



Pickit Serial Analysis and WiFi G Demo Board

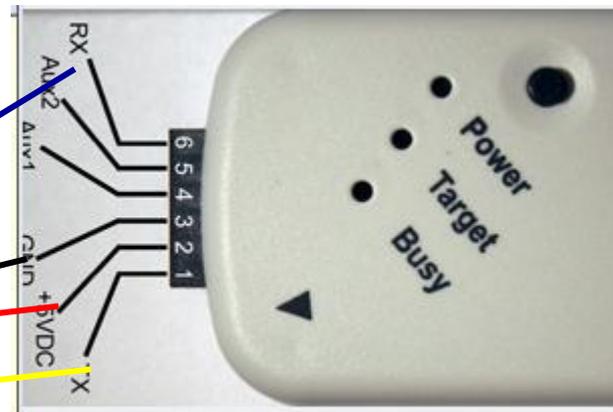


TABLE 2-1: J15 EXPANSION PORT PIN CONFIGURATIONS

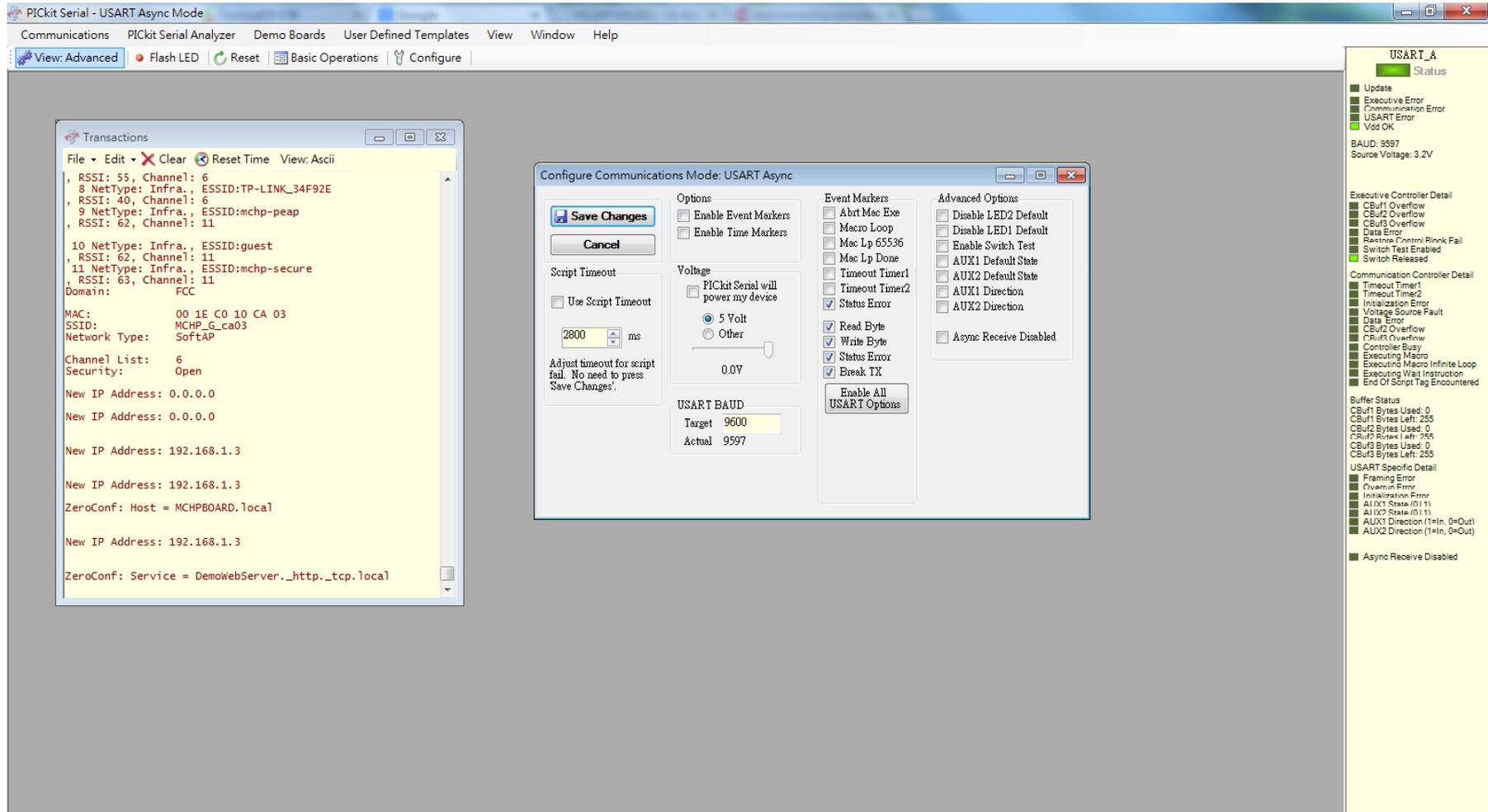
Pin 1	Pin 2	Pin 3	Pin 4	Pin 5	Pin 6	Pin 7	Pin 8	Interface
SCK	+3.3V	GND	SDI	SDO	/SS	.*	.*	SPI
-	+3.3V	GND	SDA	SCL	-	.*	.*	I ² C™
RTS	+3.3V	GND	RX	TX	CTS	.*	.*	USART
TX2	+3.3V	GND	RX1	TX1	RX2	.*	.*	DUAL USART



Pickit Serial Analysis and WiFi G Demo Board



Pickit Serial Analysis and WiFi G Demo Board



The screenshot displays the PICKIT Serial - USART Async Mode software interface. The main window shows a list of transactions with details such as RSSI, Channel, NetType, and ESSID. A 'Configure Communications Mode: USART Async' dialog box is open, showing options for Event Markers, Advanced Options, and USART BAUD settings. The USART BAUD section shows a Target of 9600 and an Actual of 9597. The right-hand side of the interface features a 'USART_A' status panel with various error and status indicators, and a 'Buffer Status' section showing bytes used and left for different buffers.

Transactions

```

File Edit Clear Reset Time View: Ascii
, RSSI: 55, Channel: 6
8 NetType: Infra, ESSID:TP-LINK_34F92E
, RSSI: 40, Channel: 6
9 NetType: Infra, ESSID:mchp-peap
, RSSI: 62, Channel: 11
10 NetType: Infra, ESSID:guest
, RSSI: 62, Channel: 11
11 NetType: Infra, ESSID:mchp-secure
, RSSI: 63, Channel: 11
Domain: FCC
MAC: 00 1E C0 10 CA 03
SSID: MCHP_G_ca03
Network Type: SoftAP
Channel List: 6
Security: Open
New IP Address: 0.0.0.0
New IP Address: 0.0.0.0
New IP Address: 192.168.1.3
New IP Address: 192.168.1.3
ZeroConf: Host = MCHPBOARD.local
New IP Address: 192.168.1.3
ZeroConf: Service = DemoWebServer._http._tcp.local
  
```

Configure Communications Mode: USART Async

Options

- Enable Event Markers
- Enable Time Markers

Script Timeout

- Use Script Timeout
- 2000 ms

Adjust timeout for script fail. No need to press 'Save Changes'.

Voltage

- PICKIT Serial will power my device
- 5 Volt
- Other
- 0.0V

USART BAUD

- Target 9600
- Actual 9597

Event Markers

- Abrt Mac Exe
- Macro Loop
- Mac Lp 65536
- Mac Lp Done
- Timeout Timer1
- Timeout Timer2
- Status Error
- Read Byte
- Write Byte
- Status Error
- Break TX

Advanced Options

- Disable LED2 Default
- Enable Switch Test
- AUX1 Default State
- AUX2 Default State
- AUX1 Direction
- AUX2 Direction
- Async Receive Disabled

Enable All USART Options

USART_A Status

- Update
- Executive Error
- Communication Error
- USART Error
- Vdd OK

BAUD: 9597
Source Voltage: 3.2V

Executive Controller Detail

- CBu1 Overflow
- CBu2 Overflow
- CBu3 Overflow
- Data Error
- Restime Control Pinok Fail
- Switch Test Enabled
- Switch Released

Communication Controller Detail

- Timeout Timer1
- Timeout Timer2
- Initialization Error
- Voltage Source Fault
- Data Error
- CBu2 Overflow
- CBu1 Overflow
- Controller Busy
- Executing Macro
- Executing Macro Infinite Loop
- Executing Wait Instruction
- End Of SScript Tag Encountered

Buffer Status

- CBu1 Bytes Used: 0
- CBu1 Bytes Left: 255
- CBu2 Bytes Used: 0
- CBu2 Bytes Left: 255
- CBu3 Bytes Used: 0
- CBu3 Bytes Left: 255

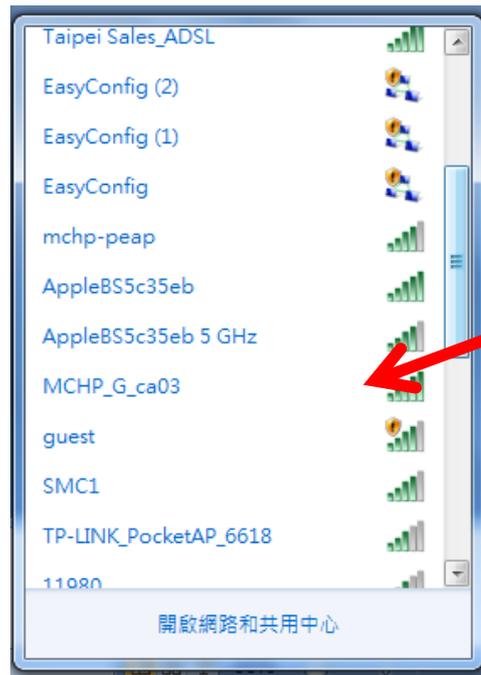
USART Specific Detail

- Framing Error
- Overrun Error
- Initialization Error
- AI(X) State (0/1)
- AI(X) State (0/1)
- AUX1 Direction (1=In, 0=Out)
- AUX2 Direction (1=In, 0=Out)
- Async Receive Disabled

Hand on

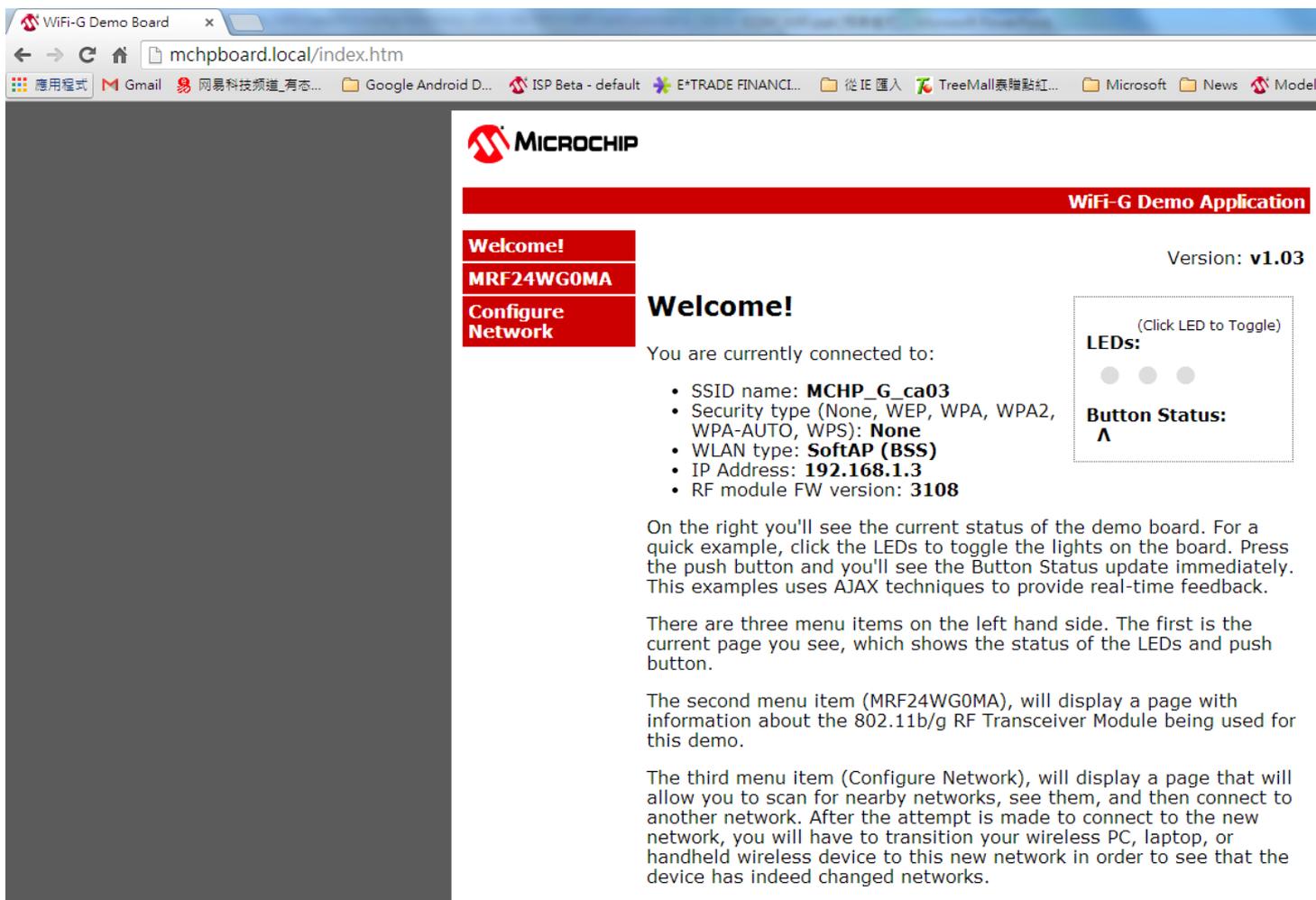
- **Lab 1: Compiler WiFiGDemo _SoftAP Project and Using IPHONE or Android PHONE to connect WiFi G Demo board**

Selecting a Network Name (SSID)



1. Connect to SSID: MCHP_G_XXXX
XXXX = Last 4 digits of MAC address
2. Browse to: <http://mchpboard.local> or
<http://192.168.1.3>

Hand on



WiFi-G Demo Board

mchpboard.local/index.htm

應用程式 Gmail 網易科技頻道_有态... Google Android D... ISP Beta - default E*TRADE FINANCI... 從 IE 匯入 TreeMail泰聯點紅... Microsoft News Model

MICROCHIP

WiFi-G Demo Application

Welcome! Version: **v1.03**

MRF24WG0MA

Configure Network

Welcome!

You are currently connected to:

- SSID name: **MCHP_G_ca03**
- Security type (None, WEP, WPA, WPA2, WPA-AUTO, WPS): **None**
- WLAN type: **SoftAP (BSS)**
- IP Address: **192.168.1.3**
- RF module FW version: **3108**

(Click LED to Toggle)

LEDs:

Button Status:
Λ

On the right you'll see the current status of the demo board. For a quick example, click the LEDs to toggle the lights on the board. Press the push button and you'll see the Button Status update immediately. This examples uses AJAX techniques to provide real-time feedback.

There are three menu items on the left hand side. The first is the current page you see, which shows the status of the LEDs and push button.

The second menu item (MRF24WG0MA), will display a page with information about the 802.11b/g RF Transceiver Module being used for this demo.

The third menu item (Configure Network), will display a page that will allow you to scan for nearby networks, see them, and then connect to another network. After the attempt is made to connect to the new network, you will have to transition your wireless PC, laptop, or handheld wireless device to this new network in order to see that the device has indeed changed networks.



MICROCHIP

Regional Training Centers

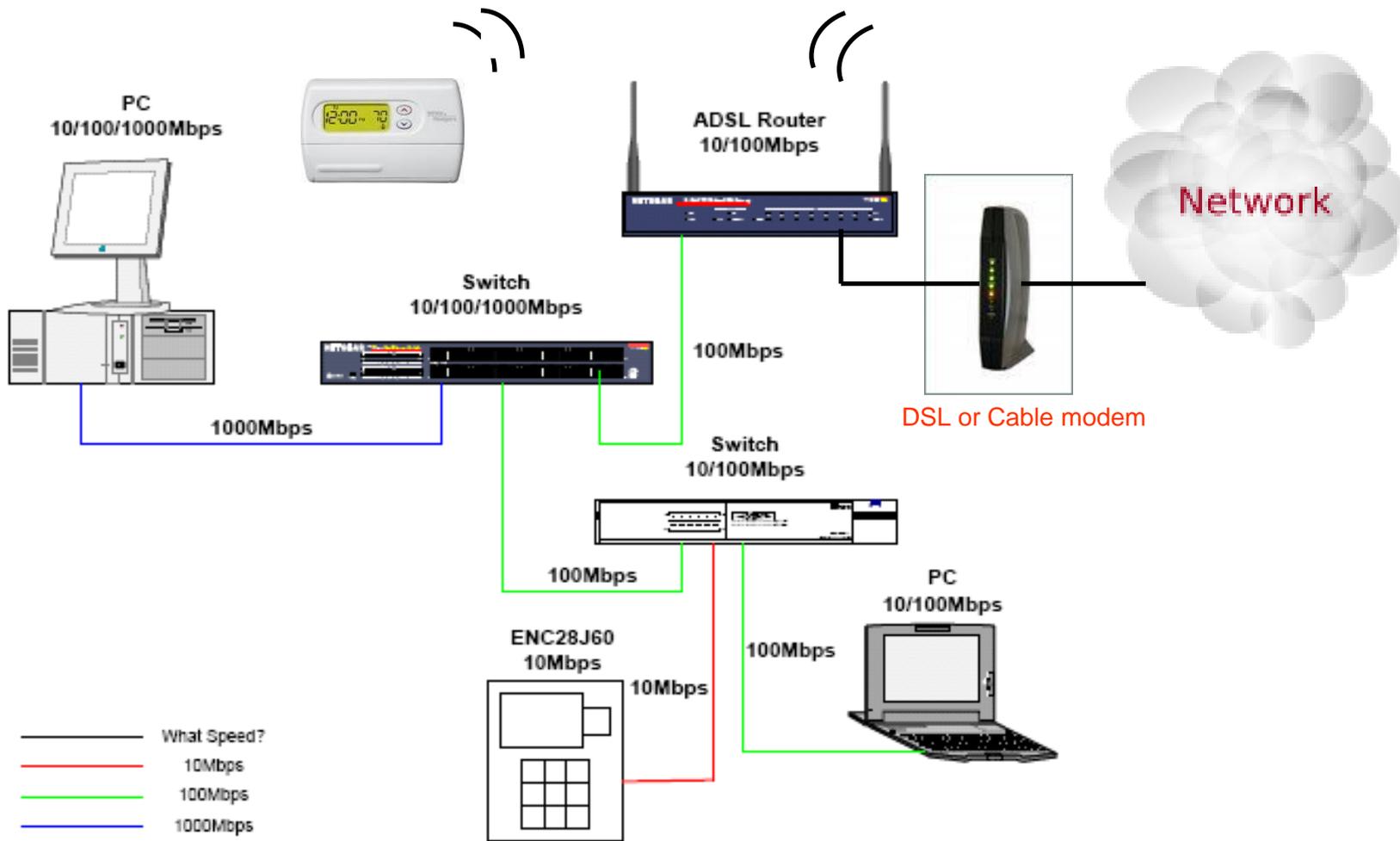
2. Embedded WiFi



MICROCHIP

Regional Training
Centers

What is Wi-Fi®?





What is Wi-Fi®?

- **Ethernet is the most widely deployed datacom network in the world**
- **Wi-Fi is wireless Ethernet**
 - **Adds mobile internet connectivity**
 - **Removes the wire, but retains the LAN, WAN, WWW connection**



What is the Internet?

- A “nuts and bolts” view -



PC



server



wireless laptop



cellular handheld



access points

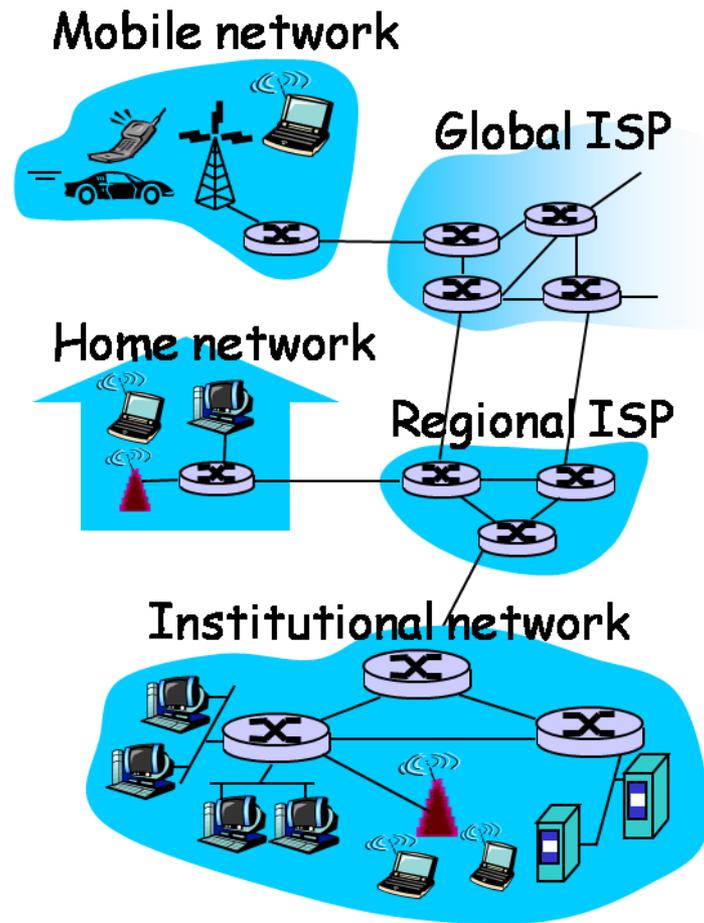


wired links



router

- **Millions of connected computing devices:**
 - **Hosts (or “stations”)**
 - Some run application services (“Servers”)
 - Some consume application data (“Clients”)
- **All running TCP/IP protocol suite**
- **Communication links:**
 - **Fiber, copper, radio, satellite**
 - **Data Transmission rate = link bandwidth**
- **Routers: Forward packets (chunks of data)**



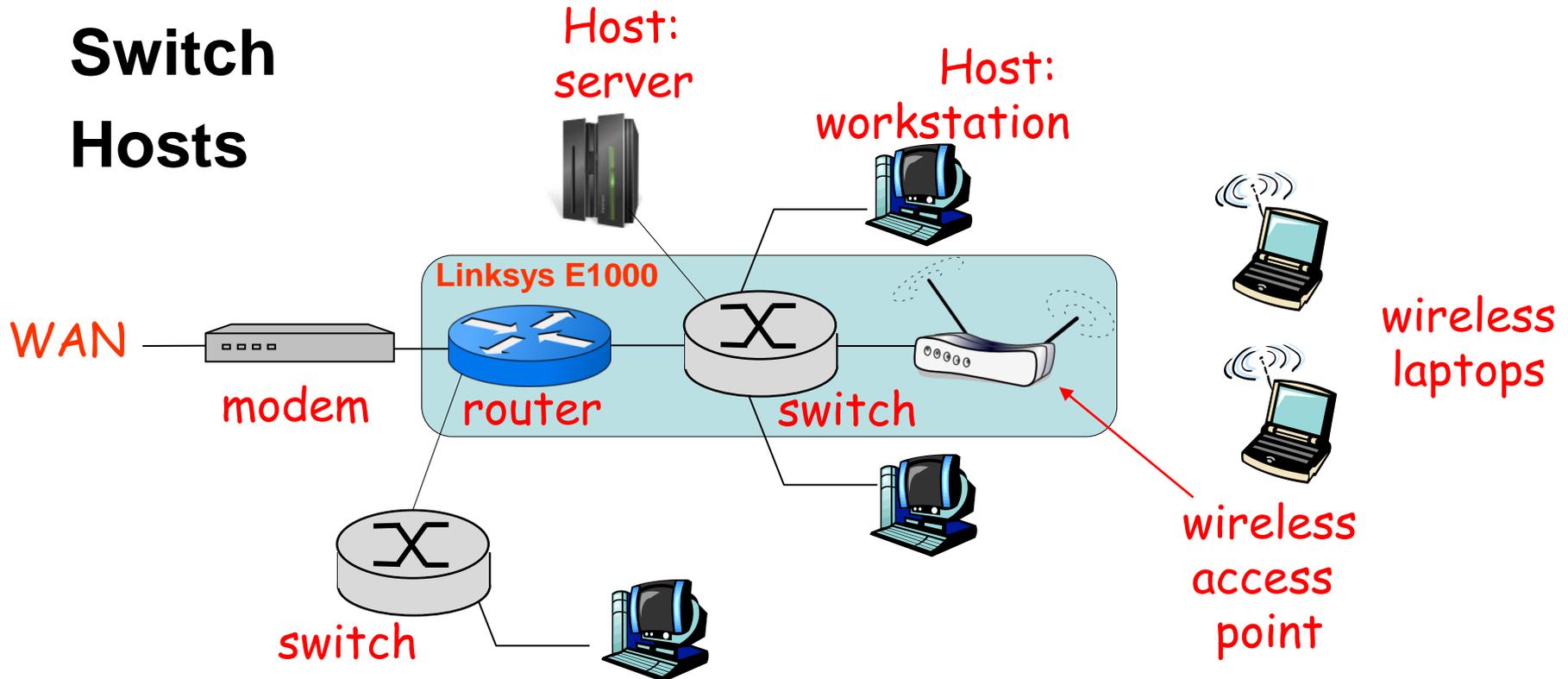
Typical Access Network

Modem to connect to the Wide Area Network (WAN) – other physical end to Ethernet

Router

Switch

Hosts



Embedded Wi-Fi®

- **Embedded products require**
 - **8, 16, 32-bit, processor support**
 - **Small memory footprints**
 - ◆ **Run from on-chip memory**
 - **Low power**
 - ◆ **Battery operation**
 - **Fast time to market**

Microchip 802.11 Solution

- **Infrastructure (BSS)**



- **Ad Hoc (IBSS)**



Embedded Wi-Fi®

Wi-Fi Protected Setup™

- Enables users to:
 - Automatically configure new wireless networks
 - Add new devices
 - Enable security



<http://www.wi-fi.org/wifi-protected-setup>

Wi-Fi Direct™

- Enables users to:
 - Connect to one another without joining a traditional network



http://www.wi-fi.org/Wi-Fi_Direct.php



MICROCHIP

Regional Training
Centers

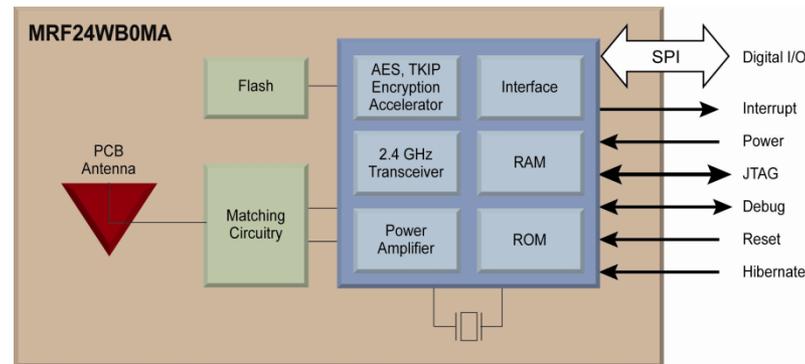
MRF24WB0MA/MB 802.11b Wi-Fi®

MRF24WB0MA and MRF24WB0MB Wi-Fi Transceiver Module Features

- Low data-rate Wi-Fi
- 1&2 Mbps
- Used when can control AP that is used
- Ultra low power consumption
 - 0.1uA hibernate
 - 250uA 802.11PS
 - 185mA Max. Tx
- FCC, and international regulatory certification
- Wi-Fi Certified
- Works with PIC® microcontrollers
- SPI interface to PIC microcontrollers
- Supports WEP, WPA and WPA2 security protocols

Free TCP/IP Stack

- Uses current Microchip TCP/IP stack
- Free download from
www.microchip.com/mla





MICROCHIP

Regional Training
Centers

MRF24WG0MA/MB

802.11b/g Wi-Fi®

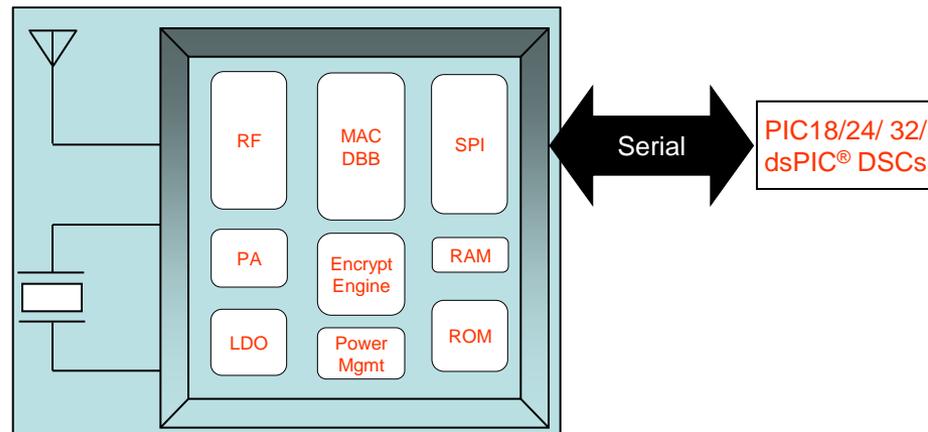
MRF24WB0MA and MRF24WB0MB Wi-Fi Transceiver Module Features

- 5mbps streaming Wi-Fi, 54mbps burst
- 1,2,5.5, 6,9,11,12,18,24,36,48,54 Mbps
- Connects to standard wireless access points
- FCC and international regulatory certification
- Works with PIC® microcontrollers
- SPI interface to PIC microcontrollers
- SoftAP, Wi-Fi Direct
- Infrastructure and Adhoc
- 802.11PS
- Wi-Fi Direct Client
- Supports WEP, WPA, WPA2, WPA-EAP security protocols

Free TCP/IP Stack

- Uses current Microchip TCP/IP stack
- Free download from

www.microchip.com/mla





Product Portfolio

MODULES	MRF24WB 0MB	MRF24W B0MA	RN-171	RN-131 C/G	MRF24W G0MB	MRF24W G0MA
802.11 Radio	b	b	b/g	b/g	b/g	b/g
Power	+10dBm	+10dBm	+12dBm	+18dBm	+18dBm	+18dBm
Antenna	uFL	Built-in	PIN	Chip/uFL	uFL	Built-in
Stack	PIC® MCU	PIC MCU	Onboard	Onboard	PIC MCU	PIC MCU
MCU Support	PIC18+	PIC18+	PIC12+	PIC12+	PIC18+	PIC18+
Throughput	1mbps	1mbps	0.7- 2mbps	0.7- 2mbps	5mbps	5mbps



RN Wi-Fi[®] Modules



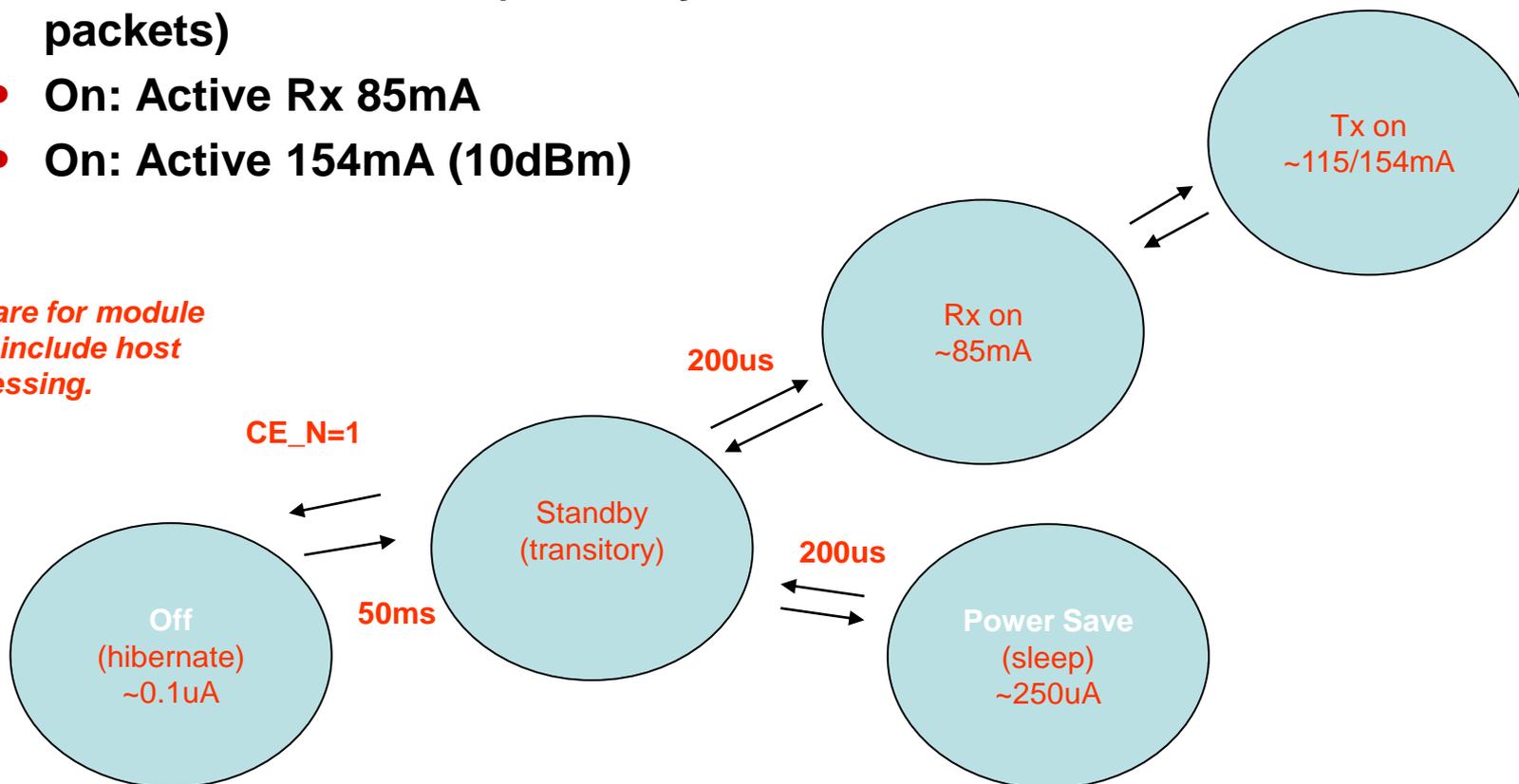
	RN-171	RN-131
Radio	True 802.11 b / g	True 802.11 b / g
Antenna	PCB trace, wire, chip, U.FL connector for external	On board chip and U.FL connector for external
Power	4uA sleep, 30mA RX, 130mA TX	4uA sleep, 40mA RX, 200mA TX
Range	10 meter to 180 meter (software configurable)	200 meter chip 300 meter external 4" dipole
HW Interface	TTL UART, SPI slave*	TTL UART, SPI slave*
Development Kit	RN-171-EK	RN-131-EK

<http://www.microchip.com/wi-fi>

Understanding Power Modes on MRF24WB0Mx

- Several power modes for low-power operation
 - Off: 0.1uA (including SPI flash on module)
 - Power Save: 250uA (Stand-by & between packets)
 - On: Active Rx 85mA
 - On: Active 154mA (10dBm)

Note: Times are for module and do not include host processing.





Product Fit

- **Types of products that are ideal**
 - ◆ **Small data quantity to transfer**
 - ◆ **Non-continuous transfer**
 - ◆ **Data is not high bandwidth streaming**



End-User Station Configuration

- **May need to:**
 - **select or enter SSID**
 - **enter Security Key**
- **Types of products:**
 - **Product has display and data entry – easy**
 - **Product does not have display or keyboard**



Configuring Light Clients

- **USB**

- ◆ Can use USB cable from PC or USB stick

- **WPS™**

- ◆ “one-button”, Wi-Fi Protected Setup™
- ◆ Routers after Jan. 2008 are compatible
- ◆ Router and client handshakes then shares info

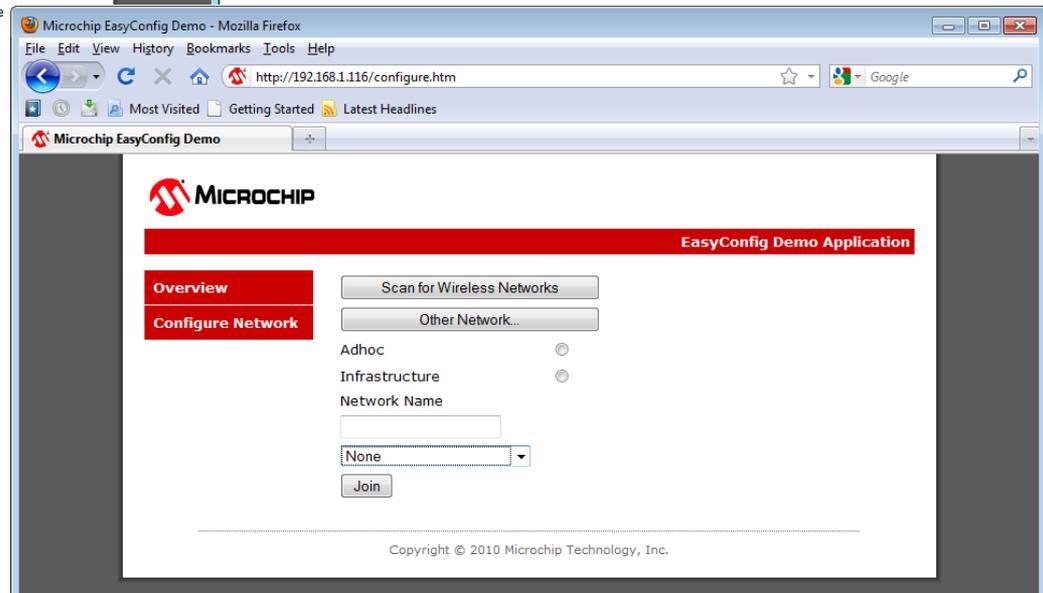
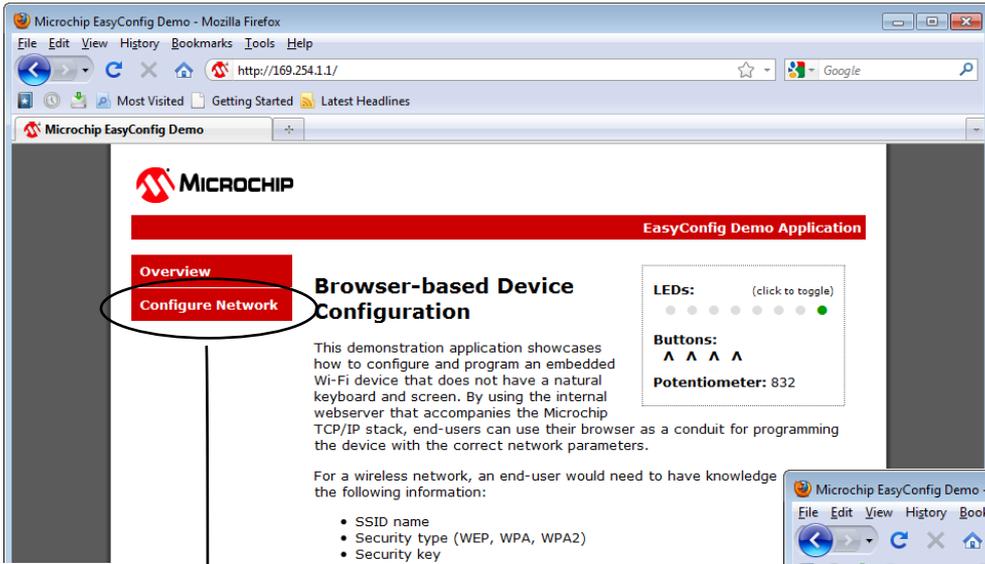
- **Browser Ad hoc configuration**

- ◆ “EasyConfiguration”
- ◆ Start in Ad Hoc and serve page to allow data entry
 - Can connect via any computer supporting Ad Hoc and has a browser

- **SoftAP Browser configuration**

- ◆ Client starts AP-like network allowing connection

EasyConfig Function





MICROCHIP

Regional Training Centers

3. WiFi Demo App—Hand on

Crash Course in Networking

- **Lab 1: Change SSID**
- **Lab 2: Change Security: WEP**
- **Lab 3: Change Security: WPA/WPA2**
- **Lab 4: Change Channel**



MICROCHIP

Regional Training
Centers

TCPIP MRF24W.h

Stack Configurations

WiFiG_DemoBoard - MPLAB IDE v8.91 - [C:\2013_12_23_CDFiles_WiFiClass\microchip_solutions_v2013-06-15\TCPIP\WiFiGDemo_AP\Configs\TCPIP MRF24W.h]

File Edit View Project Debugger Programmer Tools Configure Window Help

Release

WiFiG_DemoBoard.mcw

- WiFiG_DemoBoard.mcp
 - Source Files
 - CustomHTTPApp.c
 - MainDemo.c
 - TCPIP Stack
 - WF_Config.c
 - Wi-Fi
 - WiFiGDemoMPFSImg.c
 - Header Files
 - Common
 - HardwareProfile.h
 - HW Config
 - MainDemo.h
 - TCP Config
 - TCPIP MRF24W.h
 - TCPIP Stack
 - TCPIPConfig.h
 - WF_Config.h
 - Object Files
 - Library Files
 - Linker Script
 - Other Files

```

* disabled the following high-level application modules.
*/
// Comments:
//   iOS6.1   has wifi fixes that resolves connection issues with iPhone in softAP mode.
//
#define STACK_USE_UART           // Application demo using UART for IP address display and stack con
//#define STACK_USE_UART2TCP_BRIDGE       // UART to TCP Bridge application example
#define STACK_USE_IP_GLEANING
#define STACK_USE_ICMP_SERVER     // Ping query and response capability
#define STACK_USE_ICMP_CLIENT    // Ping transmission capability
#define STACK_USE_HTTP2_SERVER   // New HTTP server with POST, Cookies, Authentication, etc.
//#define STACK_USE_SSL_SERVER     // SSL server socket support (Requires SW300052)
//#define STACK_USE_SSL_CLIENT    // SSL client socket support (Requires SW300052)
//#define STACK_USE_AUTO_IP       // Dynamic link-layer IP address automatic configur
#define STACK_USE_DHCP_CLIENT    // Dynamic Host Configuration Protocol client for obtaining
//#define STACK_USE_DHCP_SERVER   // Single host DHCP server
//#define STACK_USE_FTP_SERVER    // File Transfer Protocol (old)
//#define STACK_USE_SMTP_CLIENT   // Simple Mail Transfer Protocol for sending email
//#define STACK_USE_SNMP_SERVER   // Simple Network Management Protocol v2C Community
//#define STACK_USE_SNMPV3_SERVER // Simple Network Management Protocol v3 Agent
//#define STACK_USE_TFTP_CLIENT   // Trivial File Transfer Protocol client
//#define STACK_USE_TELNET_SERVER // Telnet server
#define STACK_USE_ANNOUNCE       // Microchip Embedded Ethernet Device Discoverer server/client
#define STACK_USE_DNS            // Domain Name Service Client for resolving hostname strings to
//#define STACK_USE_DNS_SERVER    // Domain Name Service Server for redirection to the local
#define STACK_USE_NBNS           // NetBIOS Name Service Server for responding to NBNS hostname
#define STACK_USE_REBOOT_SERVER  // Module for resetting this PIC remotely. Primarily usefu
//#define STACK_USE_SNTP_CLIENT  // Simple Network Time Protocol for obtaining current date/
//#define STACK_USE_UDP_PERFORMANCE_TEST // Module for testing UDP TX performance characteristic
//#define STACK_USE_TCP_PERFORMANCE_TEST // Module for testing TCP TX performance characteristic
//#define STACK_USE_DYNAMICDNS_CLIENT // Dynamic DNS client updater module
//#define STACK_USE_BERKELEY_API // Berkeley Sockets APIs are available
//#define STACK_USE_ZEROCONF_LINK_LOCAL // Zeroconf IPv4 Link-Local Addressing

```

Output



WiFi® Configurations

WF_Config.h

Change SSID

The screenshot shows the MPLAB IDE interface for the WiFi_DemoBoard project. The left pane displays the project tree with the following structure:

- WiFi_DemoBoard.mcp
 - Source Files
 - CustomHTTPApp.c
 - MainDemo.c
 - TCPIP Stack
 - WF_Config.c
 - Wi-Fi
 - WifiGDemoMPFSImg.c
 - Header Files
 - Common
 - HardwareProfile.h
 - HW Config
 - MainDemo.h
 - TCP Config
 - TCPIP MRF24W.h
 - TCPIP Stack
 - TCPIPConfig.h
 - WF_Config.h
 - Object Files
 - Library Files
 - Linker Script
 - Other Files

The main editor window shows the content of `WF_Config.h`. A red arrow points from the text "Change SSID" to the line `#define MY_DEFAULT_SSID_NAME "MicrochipDemoAP" //spencer|`. The code includes various configuration macros for WiFi, such as `MY_DEFAULT_SCAN_TYPE`, `MY_DEFAULT_LIST_RETRY_COUNT`, `MY_DEFAULT_CHANNEL_LIST`, `MY_DEFAULT_BEACON_TIMEOUT`, and `MY_DEFAULT_PS_POLL`. It also contains a warning section about the connection manager and the module FW version.

```
#define EAP_USE_CA_CERT /* use CA pem cert */
#else
#error "MRF24WB does not support WPA-Enterprise. And this is verified with only PIC32"
#endif /* defined (MRF24WG) */

#endif /* #if ENABLE_WPA_ENTERPRISE == 1 */

#define MY_DEFAULT_SCAN_TYPE          WF_ACTIVE_SCAN
#define MY_DEFAULT_SSID_NAME          "MicrochipDemoAP" //spencer|
#define MY_DEFAULT_LIST_RETRY_COUNT  INFRASTRUCTURE_RETRY_COUNT /* Num
{1,2,3,4,5,6,7,8,9,10,11} /* Def:
(10) /* Number of beacon periods
#define MY_DEFAULT_BEACON_TIMEOUT    WF_DISABLED
#define MY_DEFAULT_PS_POLL           WF_DISABLED
#if !defined(MRF24WG)
/* #define WF_AGGRESSIVE_PS */ /* WARNING !!! : This only can work with 1209 module FW
* If you use the earlier version such as 1207 or 1205, then yo
* Defining this will lead ASSERT problem with old module FW.
*/
#endif

/* Warning !!! Please note that :
* RF Module FW has a built-in connection manager, and it is enabled by default.
* So if you want to run your own connection manager in host stack application side,
* then you should disable the module connection manager to avoid some possible conflict
* between the two. Especially these two APIs can be affected if you do not disable it.
* A) UINT16 WF_CMDISconnect(void)
* B) UINT16 WF_Scan(UINT8 CpId)
* If some conflict occurs then these APIs will return failure.
* Furthermore if you use old MRF24WB FW version, older than 120C, then
* it can cause fatal issue in module FW side such as FW crash.
* So for simplicity, if you want to run your own connection manager actively,
* we strongly recommend to disable the module connection manager, and this
* #define is make that thing possible. Just un-comment it to do so !
*/
#endif
#endif CONFIG_WPA_ENTERPRISE
```

WiFi® Configurations

WF_Config.h

```
C:\MAL\May2010\TCPIP WiFi Demo App\WF_Config.h

/*****
/*****
/*
          WIFI SECURITY COMPILE-TIME DEFAULTS
          */
/*****
/*****

// Security modes available on WiFi network:
//  WF_SECURITY_OPEN           : No security
//  WF_SECURITY_WEP_40        : WEP Encryption using 40 bit keys
//  WF_SECURITY_WEP_104       : WEP Encryption using 104 bit keys
//  WF_SECURITY_WPA_WITH_KEY   : WPA-PSK Personal where binary key is given to MRF24WBOM
//  WF_SECURITY_WPA_WITH_PASS_PHRASE : WPA-PSK Personal where passphrase is given to MRF24WBOM and it c
//  WF_SECURITY_WPA2_WITH_KEY  : WPA2-PSK Personal where binary key is given to MRF24WBOM
//  WF_SECURITY_WPA2_WITH_PASS_PHRASE : WPA2-PSK Personal where passphrase is given to MRF24WBOM and it
//  WF_SECURITY_WPA_AUTO_WITH_KEY : WPA-PSK Personal or WPA2-PSK Personal where binary key is given
//                                     connect at highest level AP supports (WPA or WPA2)
//  WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE : WPA-PSK Personal or WPA2-PSK Personal where passphrase is given
//                                     calculates the binary key and connects at highest level AP sup
```

Security Type



WiFi® Configurations

Security Details

WF_Config.h

```
es_WiFiClass\microchip_solutions_v2013-06-15\TCPIP\WiFiGDemo_AP\WF_Config.h*
Is Configure Window Help

// WF_SECURITY_WPS_PIN           : WPS PIN method
// WF_SECURITY_WPA2_ENTERPRISE   : WPA2 Enterprise

// Comments: CFG_WF_SOFT_AP is only avail in EZ Config
#define CFG_WF_INFRASTRUCTURE 1
#define CFG_WF_ADHOC          2
#define CFG_WF_P2P            3
#define DEMO_VERSION          "v1.03"
#define MY_DEFAULT_DOMAIN     WF_DOMAIN_FCC
#define MY_DEFAULT_NETWORK_TYPE CFG_WF_INFRASTRUCTURE /* CFG_WF_INFRASTRUCTURE, CFG_WF_ADHOC, CFG_WF_P2P */

#define INFRASTRUCTURE_RETRY_COUNT (WF_RETRY_FOREVER)
#define ADHOC_RETRY_COUNT         (3)

/*-----*/
/* if starting this demo in infrastructure mode */
/*-----*/
#if MY_DEFAULT_NETWORK_TYPE == CFG_WF_INFRASTRUCTURE
| #define MY_DEFAULT_WIFI_SECURITY_MODE     WF_SECURITY_OPEN
  // #define MY_DEFAULT_WIFI_SECURITY_MODE   WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE //spencer

#if ENABLE_WPA_ENTERPRISE == 1
/* This function requires module FW version 3108 or the later */
#if defined (MRF24WG) && defined (__C32__)
/* Security mode must be WF_SECURITY_EAP */
#undef MY_DEFAULT_WIFI_SECURITY_MODE
#define MY_DEFAULT_WIFI_SECURITY_MODE     WF_SECURITY_WPA2_ENTERPRISE
#define CONFIG_WPA_ENTERPRISE

/*
 * EAP method is validated with FreeRadius server.
 * Currently 2 major methods are validated :
 * A) EAP-TTLS/MSCHAPv2
 * B) EAP-TTLS/MSCHAPv2
 * Enable only 1 method among these by uncommenting the macro
 */

```



Configure Infrastructure

- **Demo Applications are pre-configured**
- **To change defaults, (erase EEPROM)**
- **To change: security, network**

Crash Course in Networking

- **Lab 1: Change SSID**
- **Lab 2: Change Security: WEP**
- **Lab 3: Change Security: WPA/WPA2**
- **Lab 4: Change Channel**



MICROCHIP

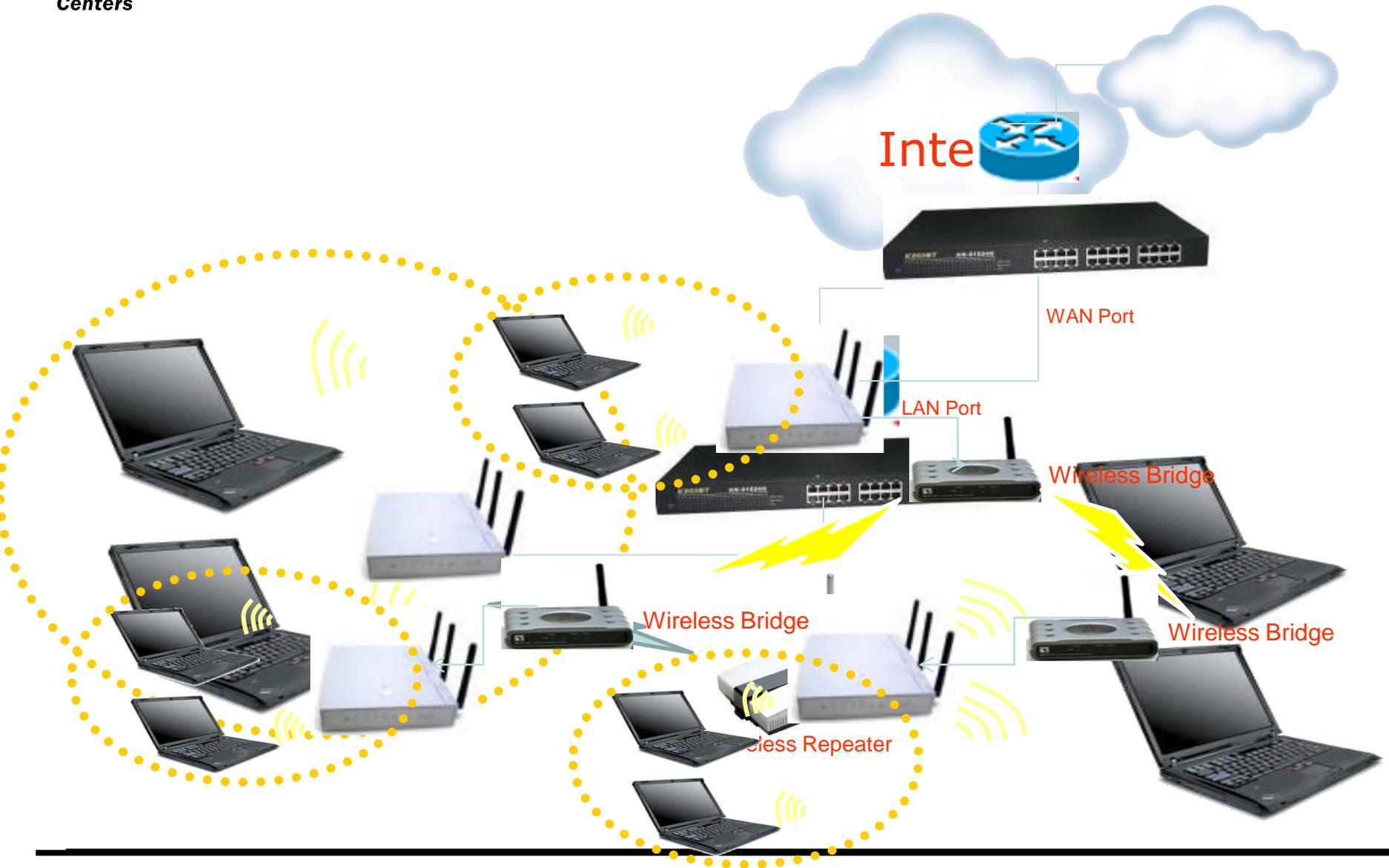
Regional Training Centers

**4. Wireless Network Deployment &
Wireless AP Feature**

Wireless Network Deployment

- Root Mode
- Repeater Mode
- Bridge Mode

Wireless Network Deployment



Wireless Network Deployment

一般設定 WPS **無線橋接** 無線存取控制 RADIUS設定 專業設定

無線網路 - 無線橋接

無線橋接（亦稱作「無線分散系統」或WDS）功能讓您可透過無線網路連接多個基地台。啓用此功能時，提醒您注意以下事項：

- 選擇「WDS Only」或「Hybrid」模式，並新增欲橋接的遠端基地台MAC位址。
- 為確保連接無誤，基地台請設定相同的頻道與**安全性加密**。
- 若欲橋接不同型號的基地台，本產品僅支援Open system WEP加密方式。

AP模式：	WDS Only
頻道：	1
連接清單中的基地台？	<input checked="" type="radio"/> Yes <input type="radio"/> No
遠端基地台清單	<input type="text"/> <input type="button" value="新增"/> <small>*請輸入完整包含12個16進位制字母的MAC位址，0~9，A~F不包括"："。</small>
	<input type="text"/> <input type="button" value="刪除"/>



802.11 Network Deployment

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\SChang>ping 10.240.98.1

Pinging 10.240.98.1 with 32 bytes of data:

Reply from 10.240.98.1: bytes=32 time<1ms TTL=255

Ping statistics for 10.240.98.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\SChang>
```

Wireless AP Features

- **MAC Address Control**
- **Bandwidth Management**
- **Port Trigger**
- **Virtual Server**
- **Dynamic DNS**
- **FireWall**
- **3G Broadband Router**
- **Power-over-Ethernet**

MAC Address Control

一般設定 WPS 無線橋接 **無線存取控制** RADIUS設定 專業設定

無線網路 - 無線存取控制

此功能能夠控制WL-520GU無線區域網路中，特定網路卡實體位址（MAC Address）的存取。

MAC存取模式：	<input type="button" value="未啟用"/> ▾
	<input type="button" value="未啟用"/> <input type="button" value="允許模式"/> <input type="button" value="拒絕模式"/>
MAC位址：	<input type="text"/> <small>請輸入完整包含12個16進位制字母的MAC位址，0~9，A~F不包括"："。</small>
	<input type="button" value="新增"/>
MAC存取控制名單：	<div style="border: 1px solid gray; height: 150px; width: 100%;"></div> <input type="button" value="刪除"/>
<input type="button" value="套用本頁面設定"/>	

Bandwidth Management

網際網路設定 | **頻寬管理** | 通訊埠觸發程式 | 虛擬伺服器 | DMZ | DDNS

頻寬管理 - 使用者指定服務

提供高、一般與低三個優先權。例如：您可設定來源IP位址為192.168.1.3的使用者，在FTP服務上，其21連接埠的優先權為最高。

頻寬狀態

偵測總上傳頻寬： Kb/s

手動指定總上傳頻寬： Kb/s

使用者指定規則表

服務名稱	來源 IP 位址	目的地連接埠	優先權	
<input type="text"/>	<input type="text"/>	<input type="text"/>	一般 ▾	<input type="button" value="新增"/>
目前沒有資料				

長封包分割

Port Trigger

[網際網路設定](#)
[頻寬管理](#)
[通訊埠觸發程式](#)
[虛擬伺服器](#)
[DMZ](#)
[DDNS](#)

NAT設定 - 通訊埠觸發程式

「通訊埠觸發程式」功能讓您可以開啓特定的TCP或UDP通訊埠來與連接 WL-520GU 的電腦進行通訊。而藉由指定觸發程式通訊埠及內傳通訊埠的方式即可完成此一操作。一旦偵測到觸發程式通訊埠，送往指定內傳通訊埠號的上傳封包便會轉向您的電腦。

觸發程式通訊埠清單

啓用通訊埠觸發程式？ Yes No

常見的應用：

說明	觸發程式通訊埠	通訊協定	內傳通訊埠	通訊協定	
<input type="text"/>	<input type="text"/>	TCP	<input type="text"/>	TCP	<input type="button" value="新增"/>
目前沒有資料					
					<input type="button" value="套用本頁面設定"/>



Virtual Server

網際網路設定 頻寬管理 通訊埠觸發程式 **虛擬伺服器** DMZ DDNS

NAT設定 - 虛擬伺服器

要由您所在網路當中的伺服器對網外用戶提供像是WWW、FTP等的網路服務時，您應該先要指定一連結伺服器的本地IP位址。接著，將IP位址及網路通訊協定的類型、通訊埠編號及網路服務名稱新增至下列的清單之中。以此清單為前題，閘道器會將網路服務要求由網外用戶轉遞給相對的本地伺服器。

啟用虛擬伺服器？ Yes No
 內建的伺服器應用： FTP
 內建的遊戲應用： 請選擇

虛擬伺服器清單			
服務名稱	通訊埠範圍	本地IP	本地通訊埠
FTP Server	20:21	192.168.1.10	
目前沒有資料			

請選擇
 FTP
 TELNET
 SMTP
 DNS
 FINGER
 HTTP
 POP3
 SNMP
 SNMP TRAP

請選擇
 Age of Empires
 BitTorrent
 Counter Strike(TCP)
 Counter Strike(UDP)
 PlayStation2
 Warcraft III
 WOW
 Xbox Live

目前沒有資料

新增

套用本頁面設定



DDNS

網際網路設定 頻寬管理 通訊埠觸發程式 虛擬伺服器 DMZ **DDNS**

外部網路 - DDNS

動態DNS (DDNS) 讓您即使在沒有靜態IP位址的情況下，仍可將伺服器連同特有名稱一併匯出至國際網路上。目前，在 WL-520GU 之中已有數台嵌入的DDNS客戶端。您可點選下方的「免費試用」，來啟用一個免費試用帳號。

啟用DDNS Client?	<input type="radio"/> Yes <input checked="" type="radio"/> No
伺服器:	<input type="text" value="WWW.DYNDNS.ORG"/> 免費試用
用戶名稱或E-mail帳號:	<input type="text"/>
密碼或DDNS金鑰:	<input type="text"/>
主機名稱:	<input type="text"/> <input type="button" value="查詢"/>
啟用萬用字元 (wildcard)?	<input type="radio"/> Yes <input checked="" type="radio"/> No
手動更新:	<input type="button" value="更新"/>

- WWW.DYNDNS.ORG
- WWW.ASUS.COM
- WWW.DYNDNS.ORG
- WWW.DYNDNS.ORG(CUSTOM)
- WWW.DYNDNS.ORG(STATIC)
- WWW.TZO.COM
- WWW.ZONEEDIT.COM



FireWall

一般設定 網址過濾 MAC存取控制 封包過濾

防火牆 - 一般設定

「啓用防火牆」功能可為 WL-520GU 及在其之後的裝置提供基本的保護。如果您要過濾出一些特定的封包，請採用在次頁中所述的LAN及WAN過濾功能。

啓用防火牆？	<input checked="" type="radio"/> Yes <input type="radio"/> No
啓動DoS防護？	<input type="radio"/> Yes <input checked="" type="radio"/> No
紀錄的封包類型：	None <input type="button" value="v"/>
從網際網路設定 WL-520GU？	<input type="radio"/> Yes <input checked="" type="radio"/> No
網際網路設定通訊埠：	8080
回應LPR要求？	<input type="radio"/> Yes <input checked="" type="radio"/> No
回應PING要求？	<input type="radio"/> Yes <input checked="" type="radio"/> No

套用本頁面設定



3G Broadband Router

The screenshot shows a web interface for a 3G Broadband Router. It is divided into two main sections: **Mobile Network Status** and **3G/UMTS Card Status**. The **Mobile Network Status** section includes fields for PIN Code, Mobile Network Bearer, Network Name, Signal Strength (No Signal), Connection Time (None), and Current Session Usage (None). A **Select Network** button is present. The **3G/UMTS Card Status** section includes fields for Card Model, Card Revision, Card Firmware, and IMSI. A **Refresh** button is located at the bottom right. A blue sidebar on the right contains instructions for Home Network, APN Name, and PIN Code. Three yellow callout boxes with arrows provide additional guidance: one points to the PIN Code input field, another points to the Select Network button, and a third points to the Card Model field.

Mobile Network Status

PIN Code :

Mobile Network Bearer :

Network Name :

Signal Strength : **No Signal**

Connection Time : **None**

Current Session Usage : **None**

Select Network

3G/UMTS Card Status

Card Model :

Card Revision :

Card Firmware :

IMSI :

Refresh

Home Network : Please choose the Vodafone network which provides your service.

APN Name : We recommend... as told to do so by Vodafone or Linksys support or your IT manager, you do change this setting. If you have access to a private Access Point Name you can enter it here.

PIN Code : If required, please enter your SIM card's PIN code. The code will be stored here for future reference by the router and you will not have re-enter this unless you change SIM cards.

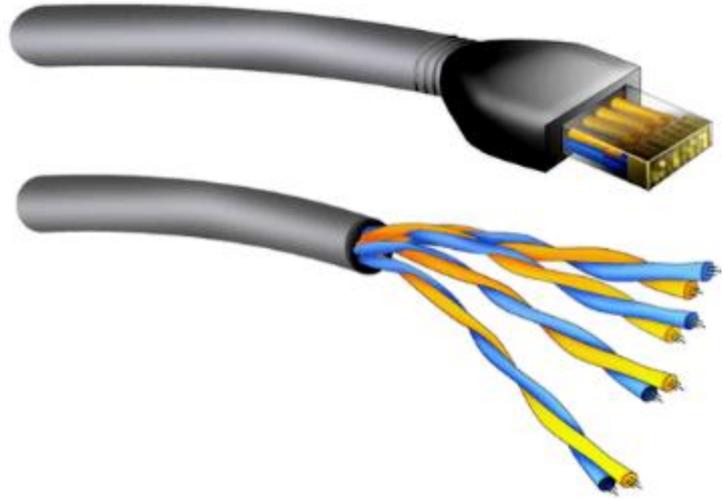
More...

Enter the SIM card's PIN code.

This will display the **Mobile Network Status**.

This will display the **3G/UMTS Card Status**.

Power-over-Ethernet



PINS on Switch	10/100 DC on Spares
Pin 1	Rx +
Pin 2	Rx -
Pin 3	Tx +
Pin 4	DC +
Pin 5	DC +
Pin 6	Tx -
Pin 7	DC -
Pin 8	DC -



Wireless AP Setting (Handon)



MICROCHIP

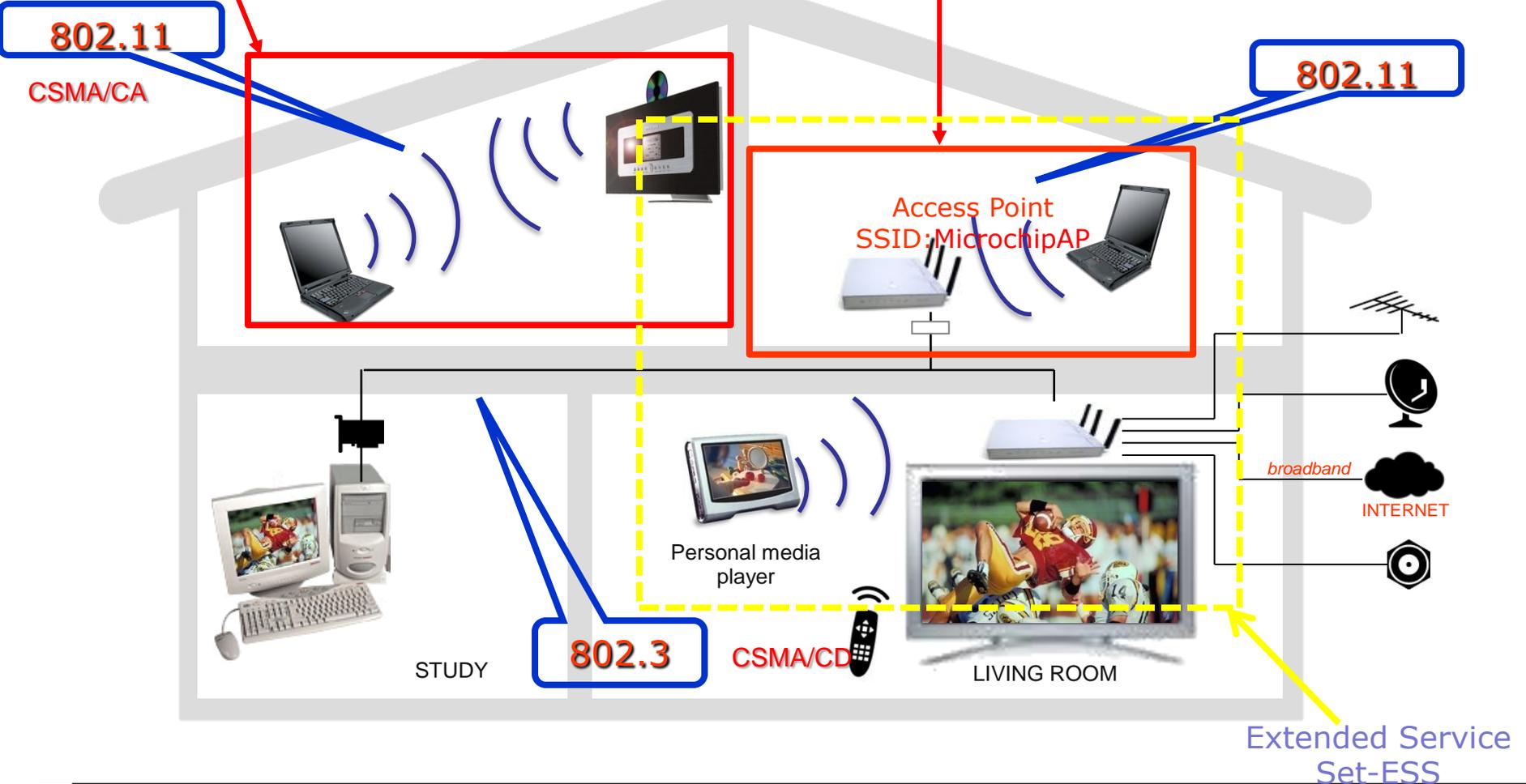
Regional Training Centers

5. Overview of 802.11 Networks

Overview of 802.11 Network

Ad-Hoc Mode (Independent Basic Service Set-IBSS)

Infrastructure mode
(Basic Service Set-BSS)



802.11 Media Access and Frame Control

- Challenges for the MAC
- Frame Transmission & Association States
 - Frame types of 802.11 protocol
 - 802.11 Frame Format in Detail

Challenges for the MAC

- Positive acknowledgment of data transmission

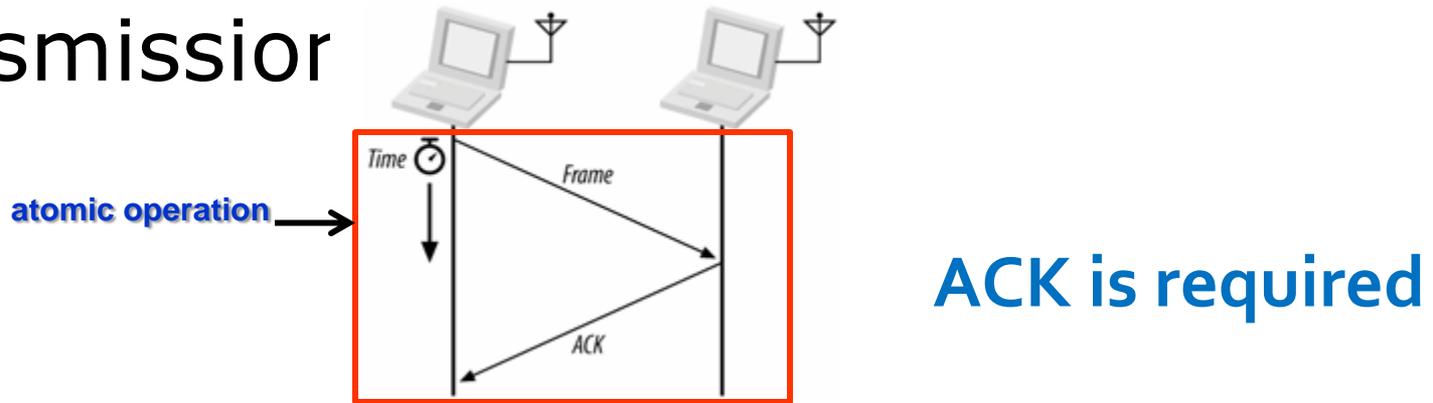


Figure- Positive acknowledgment of data transmissions

- The Hidden Node Problem

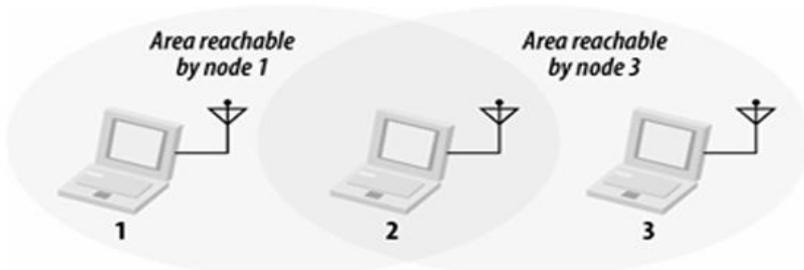


Figure- Nodes 1 and 3 are "hidden"

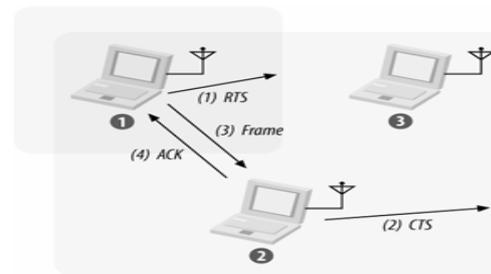
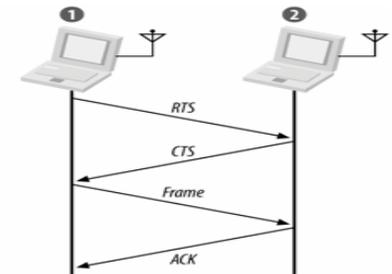


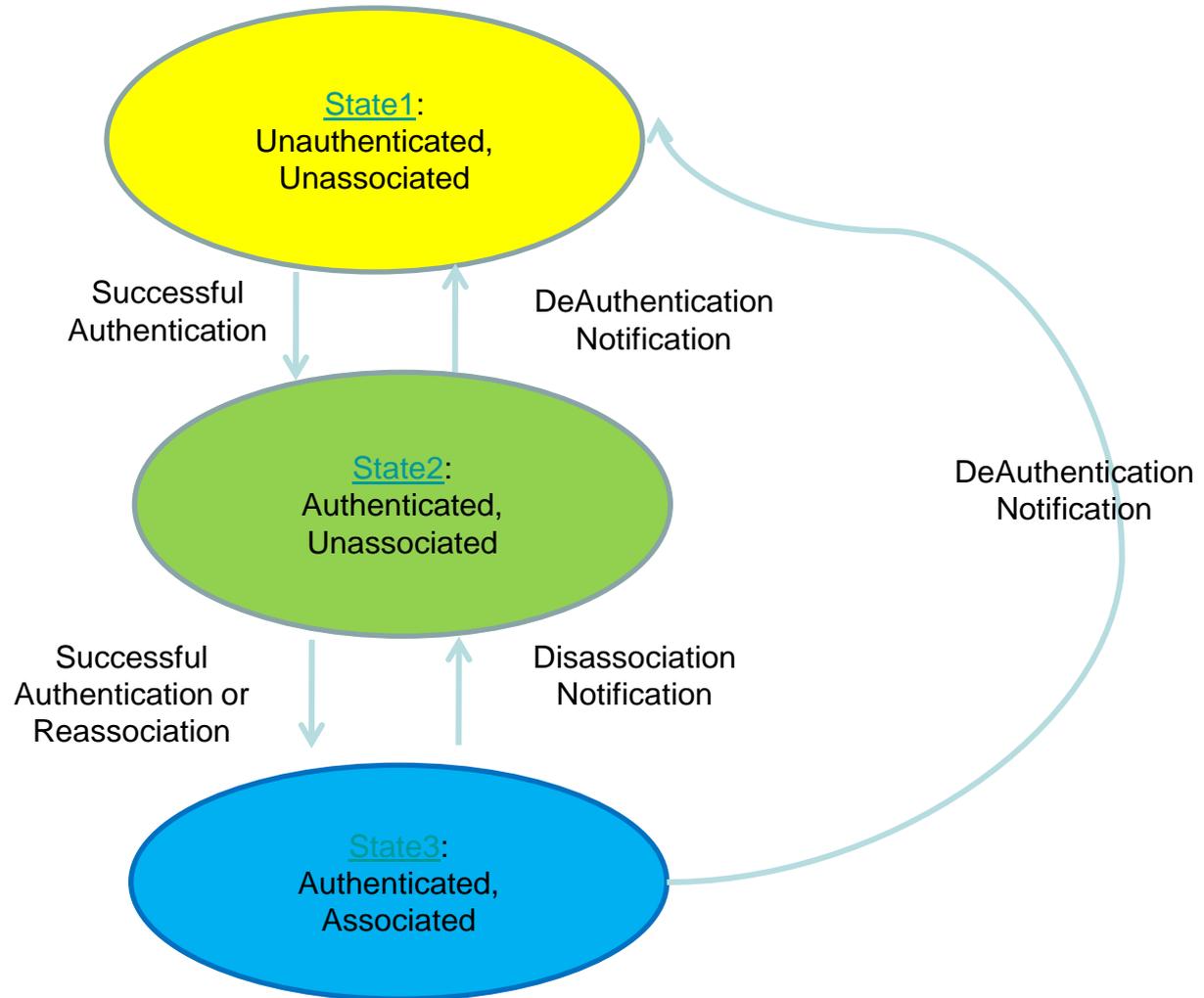
Figure- RTS/CTS clearing





Frame Transmission & Association States

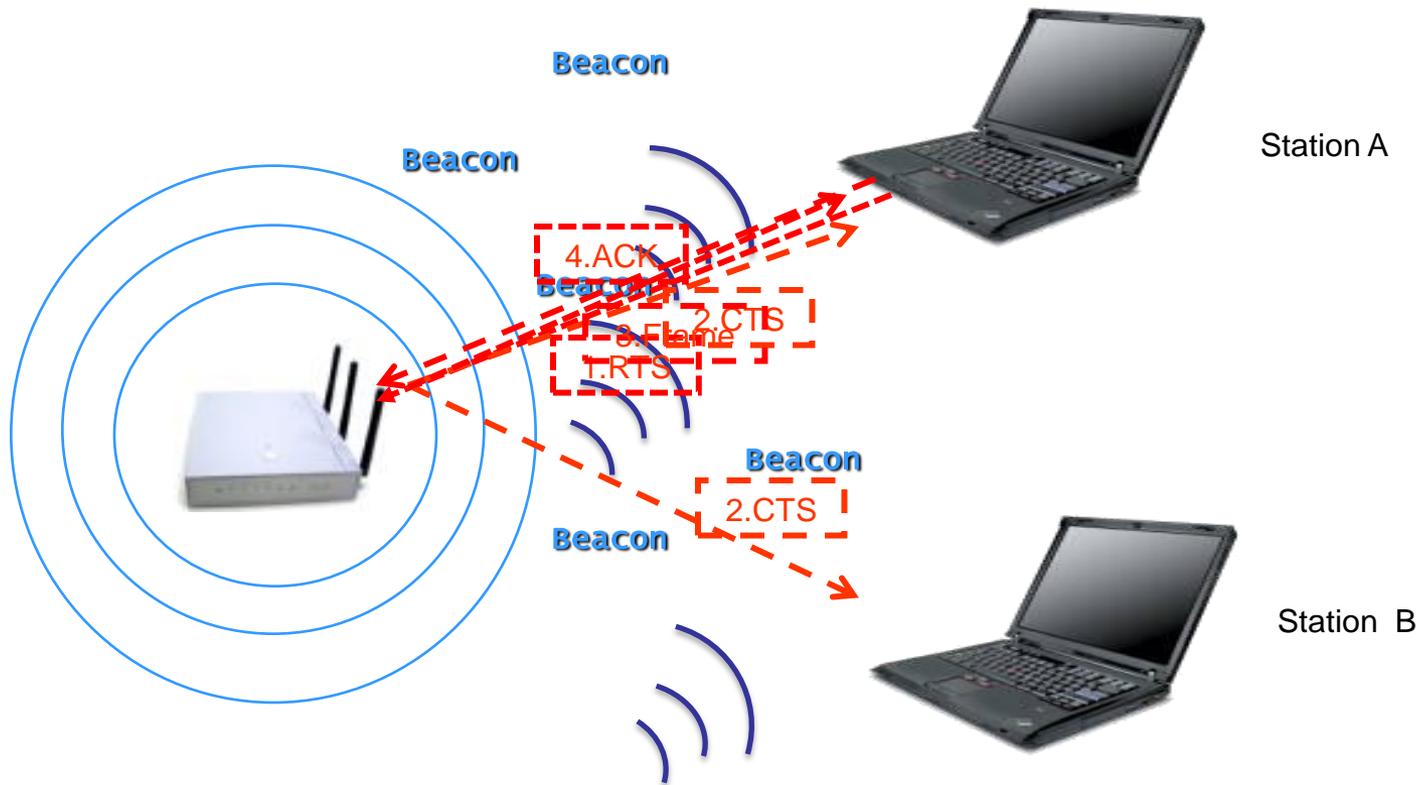
- Station connect to AP(Access Point)





Frame Transmission & Association States

- State 1(Unauthenticated, Unassociated)





Beacon-Management Frame

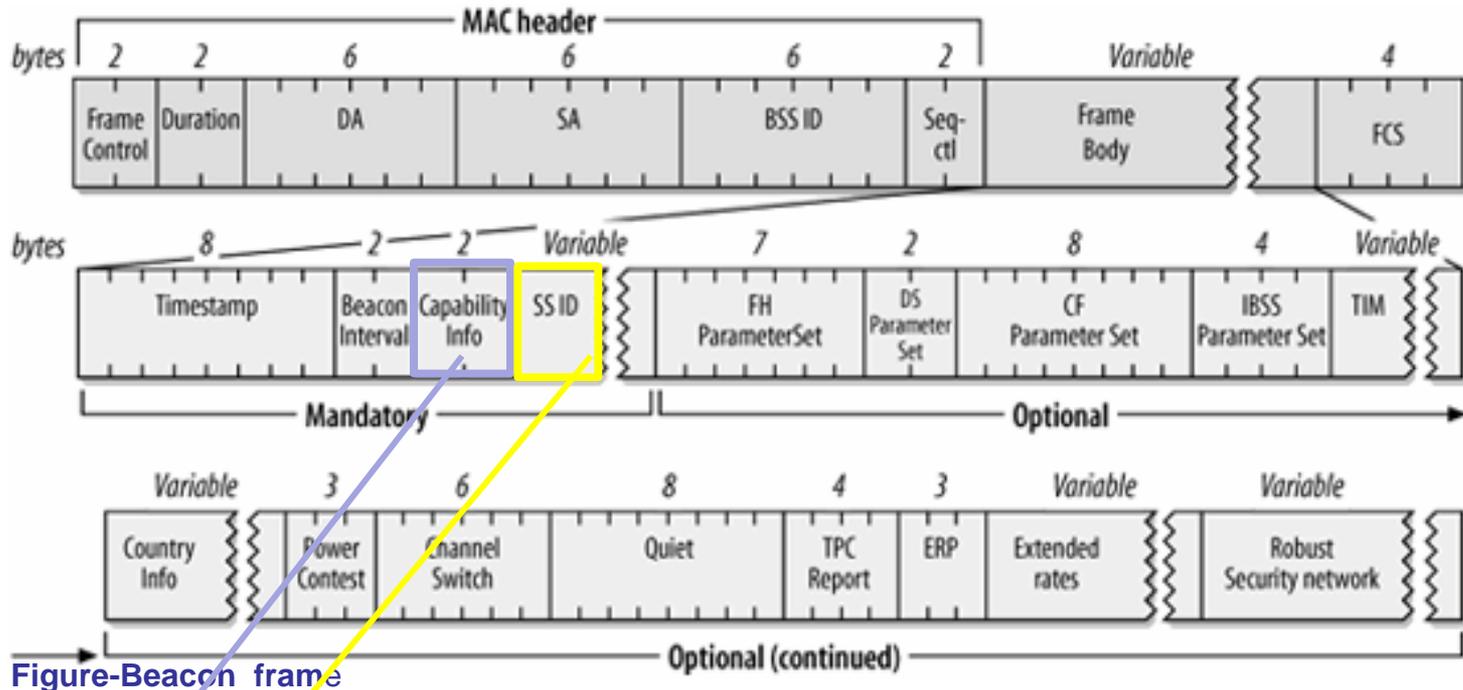


Figure-Beacon frame

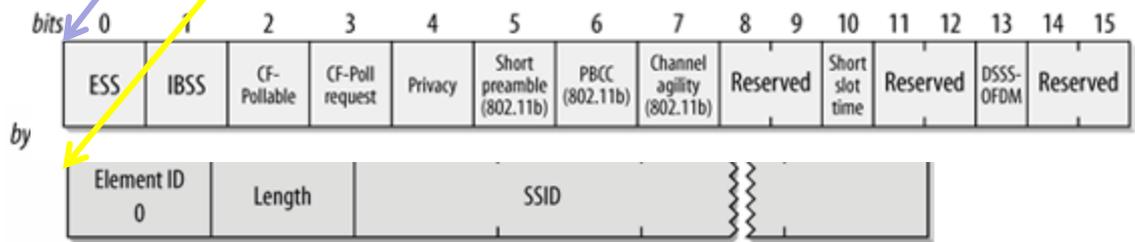


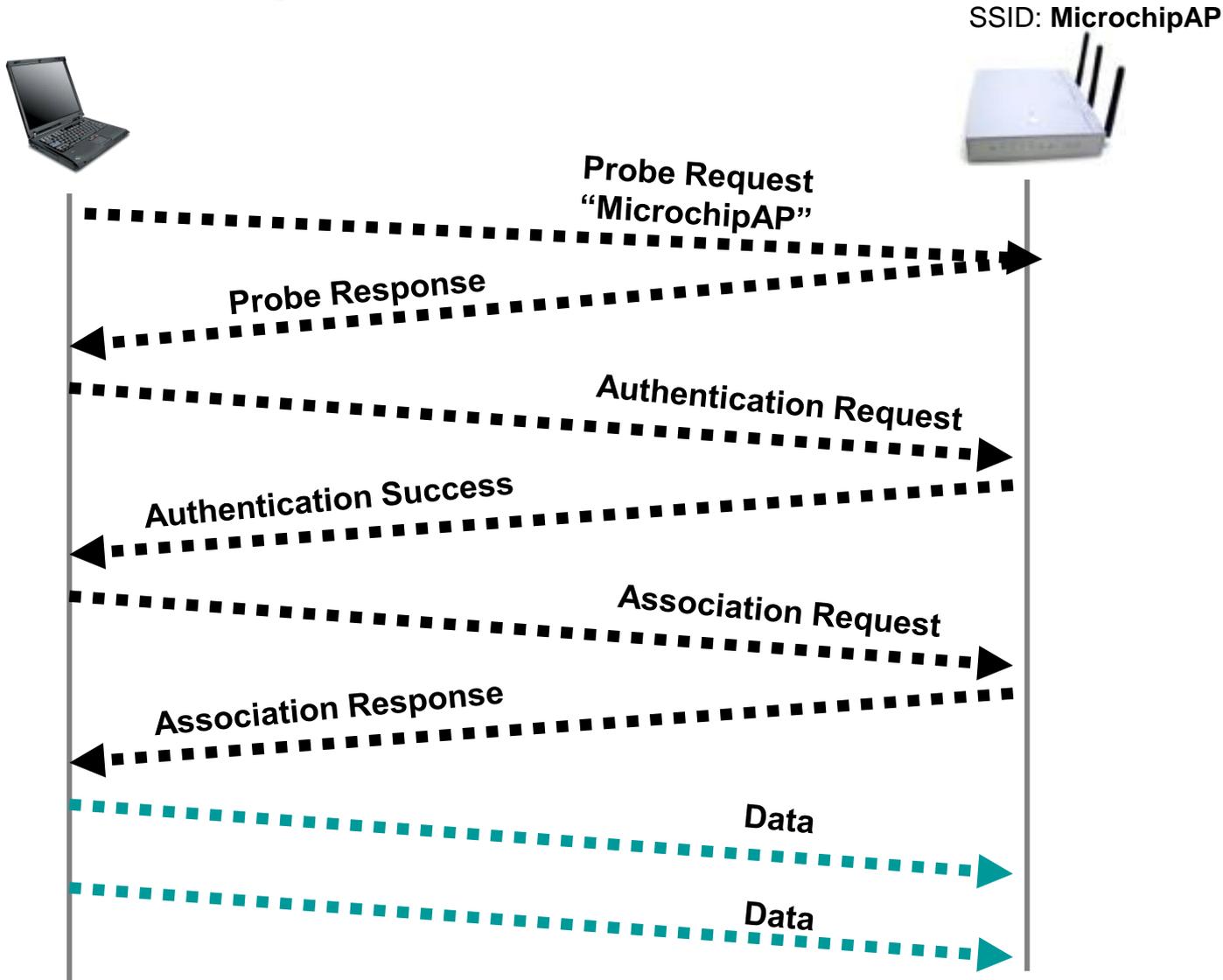
Figure-Service Set Identity information element

***Most products require that the string be a garden variety, null-terminated ASCII string. The length of SSID ranges 0 ~ 32 bytes.**

Capability Information field
 * ESS/IBSS:1/0 -> Infrastructure mode
 0/1-> Ad-Hoc mode
 * Privacy=1, WEP required

Probe Request/Response (Management Frame)

SSID: MicrochipAP

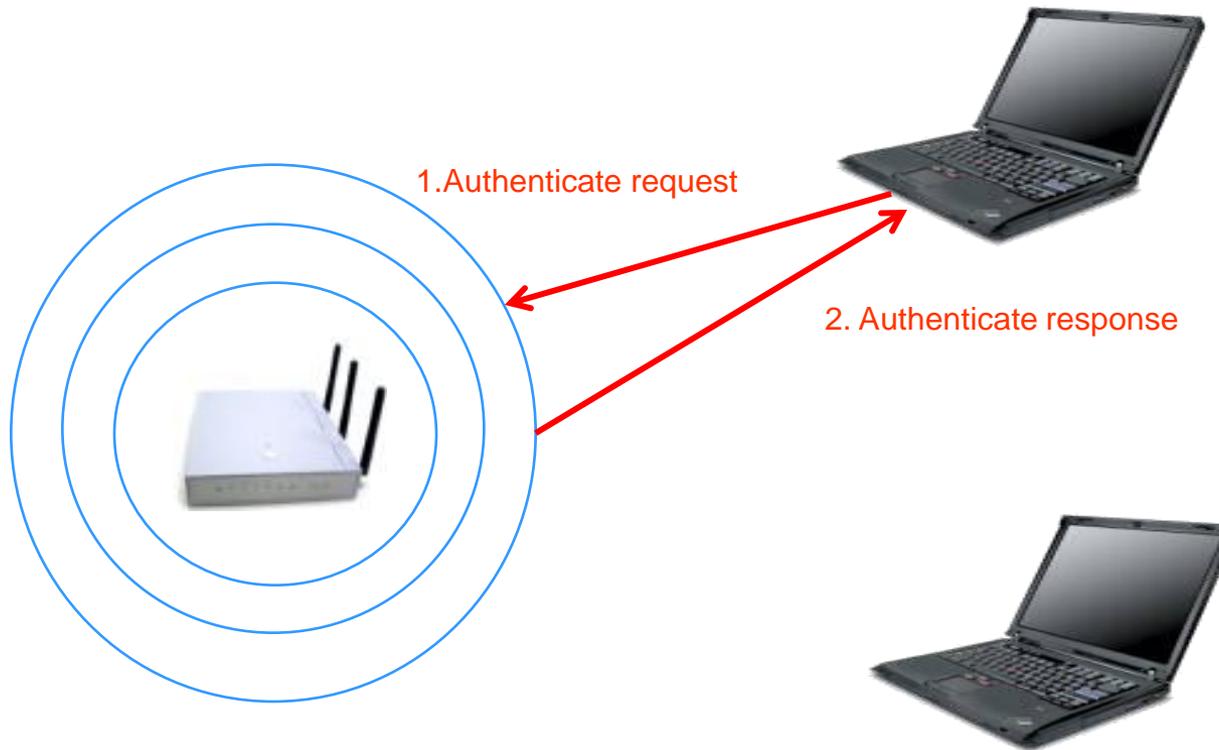


[Back State2](#)



802.11 Frame State

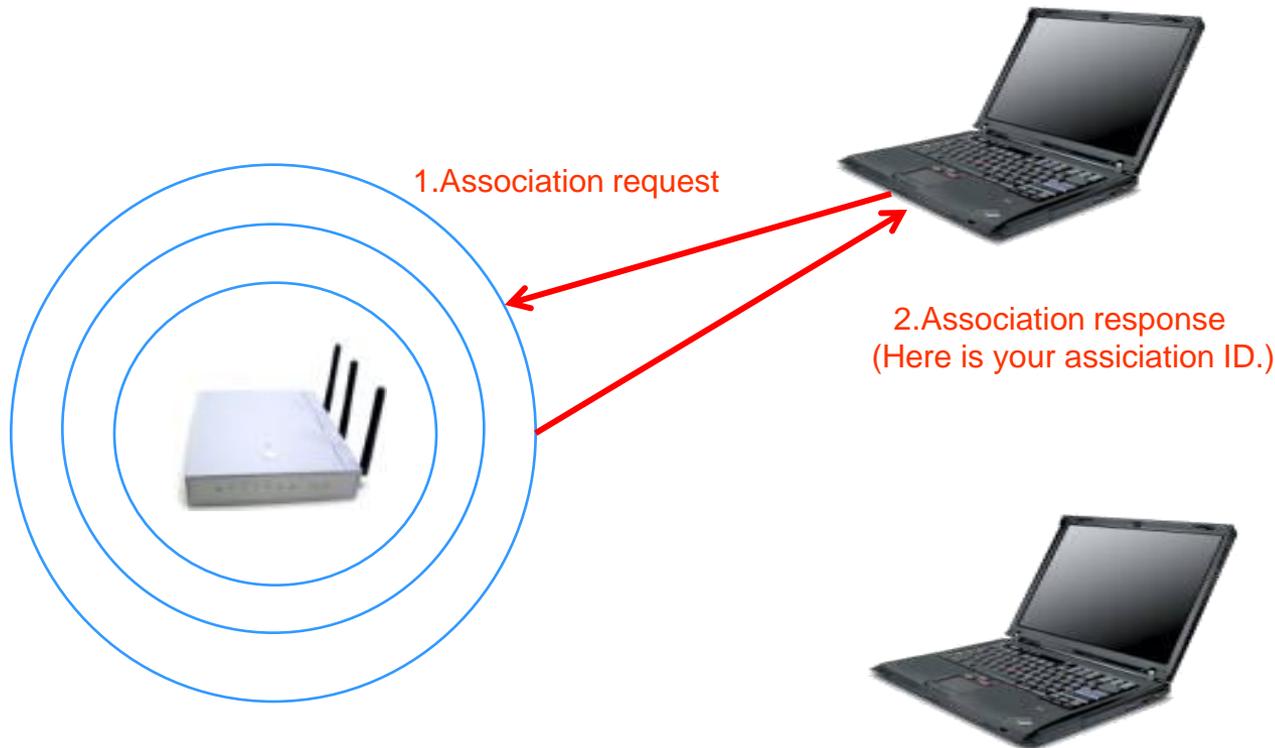
- State 2(Authenticated, Unassociated)





802.11 Frame State

- State 3(Authenticated, Associated)



Disassociation and de-authentication (Management Frame)

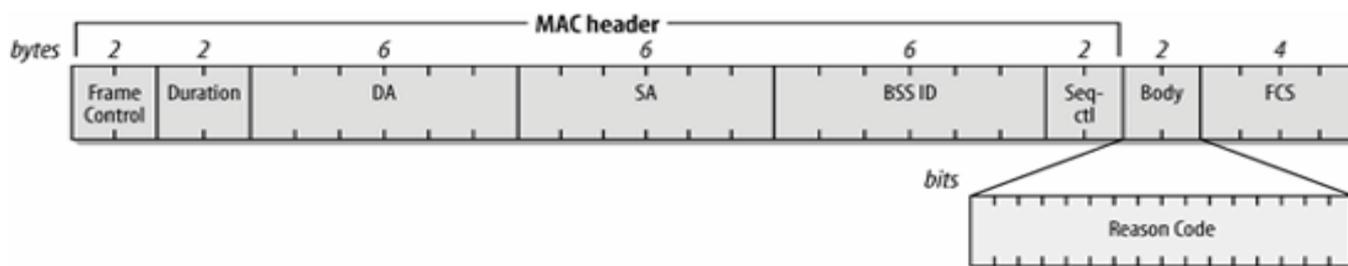


Figure-Disassociation and Deauthentication frames

Code	Explanation
3	Prior authentication is no valid
4	Inactivity time expired and station was disassociated
5	Disassociated due to insufficient resources at the access point

[Back State3](#)

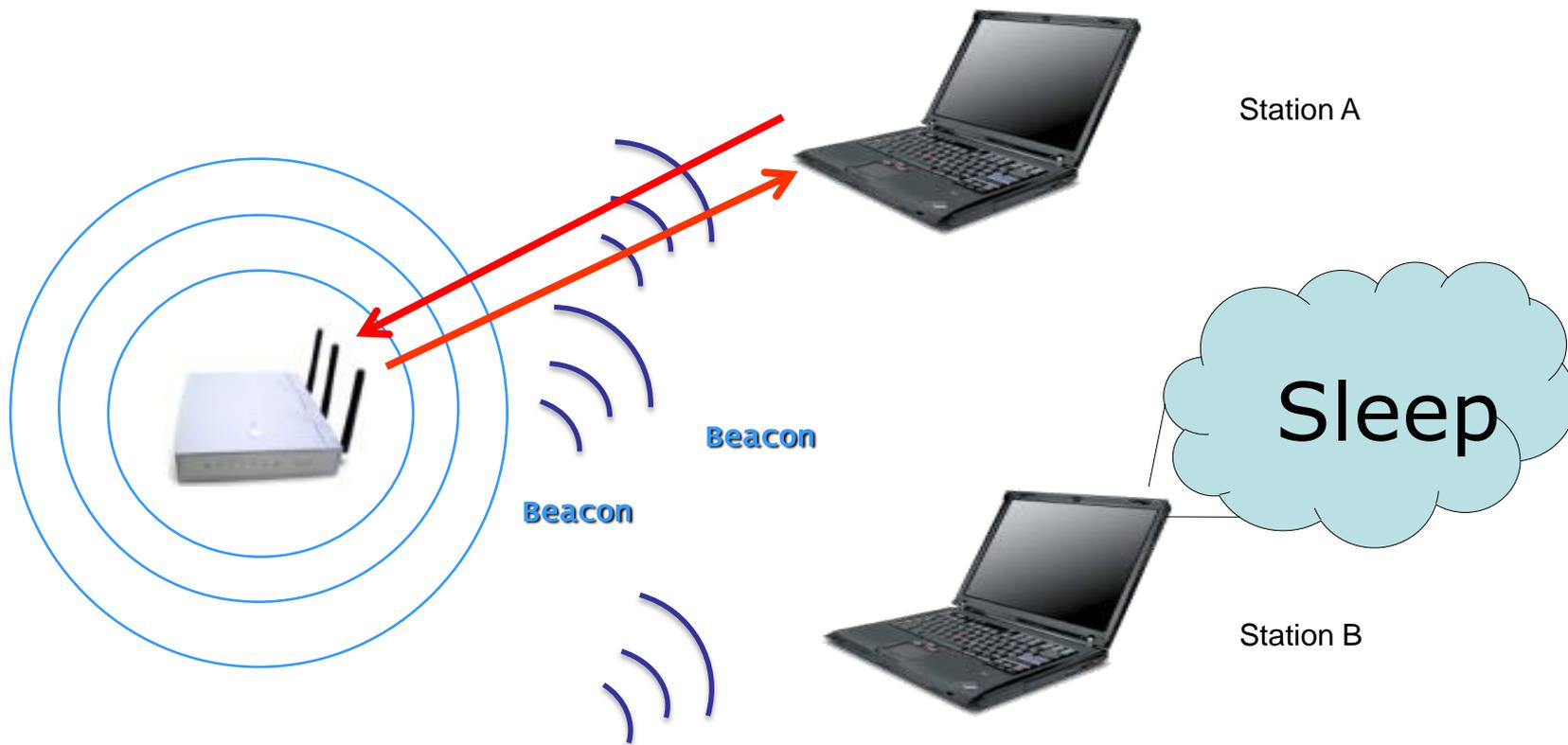


MICROCHIP

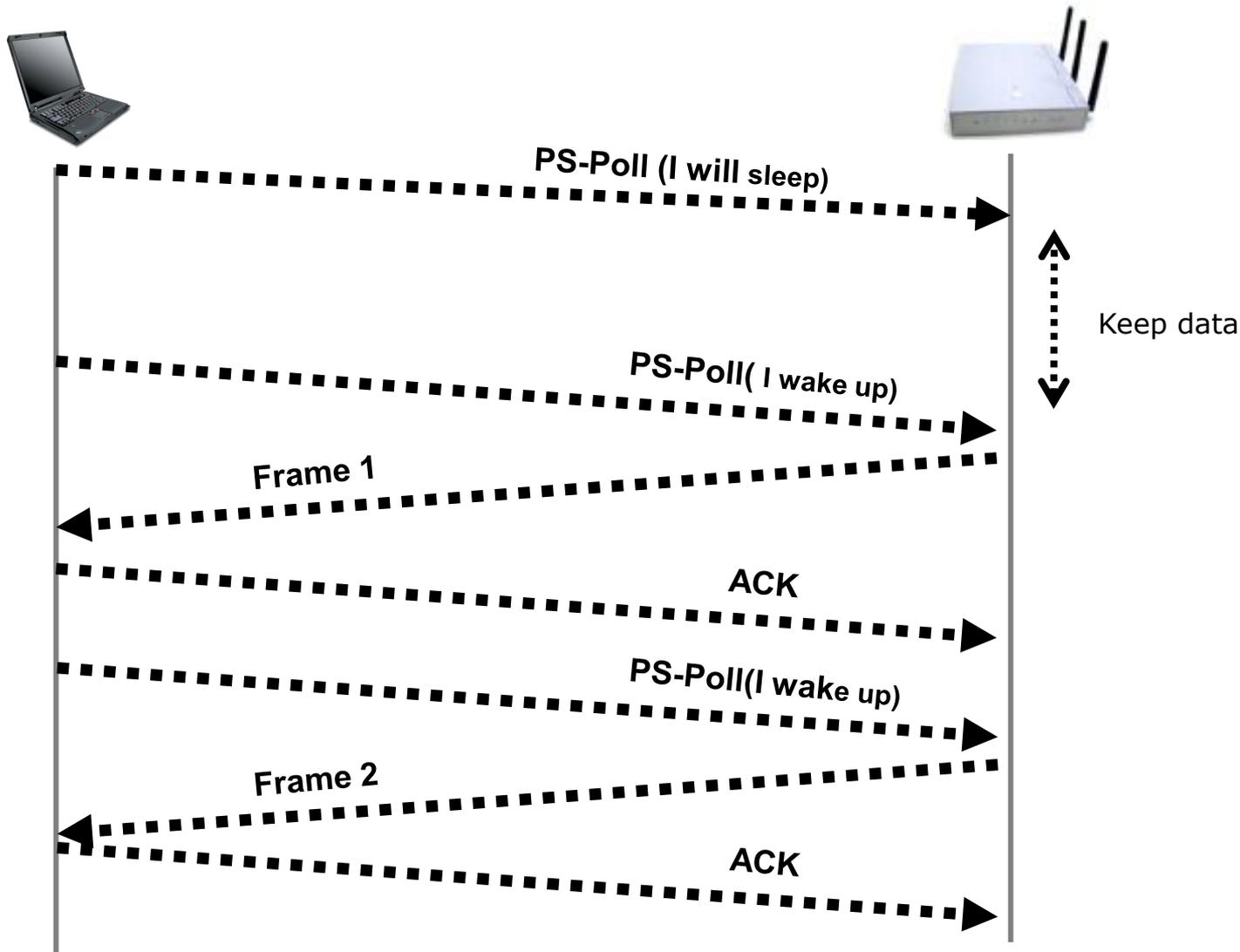
Regional Training
Centers

Frame Transmission & Association States

- State 3 (Authenticated/Associated)



Power-Save Poll (PS-Poll) Control Frame





MICROCHIP

Regional Training Centers

Low Power Operation
Power Management



Power Management

- **There are 4 modes of operation**
 - **Active (On)**
 - ◆ Transmitting or receiving
 - ◆ Few hours battery life with Rx on
 - **802.11PS**
 - ◆ Maintains connection, transparent disconnect
 - ◆ 2 months battery life
 - **Saved State**
 - ◆ Connection info stored after first commissioning
 - ◆ Can reconnect transmit and sleep in 100mS
 - ◆ 1+ year battery life
 - **Hibernate**
 - ◆ Disconnected, session lost
 - ◆ 2+ years battery life



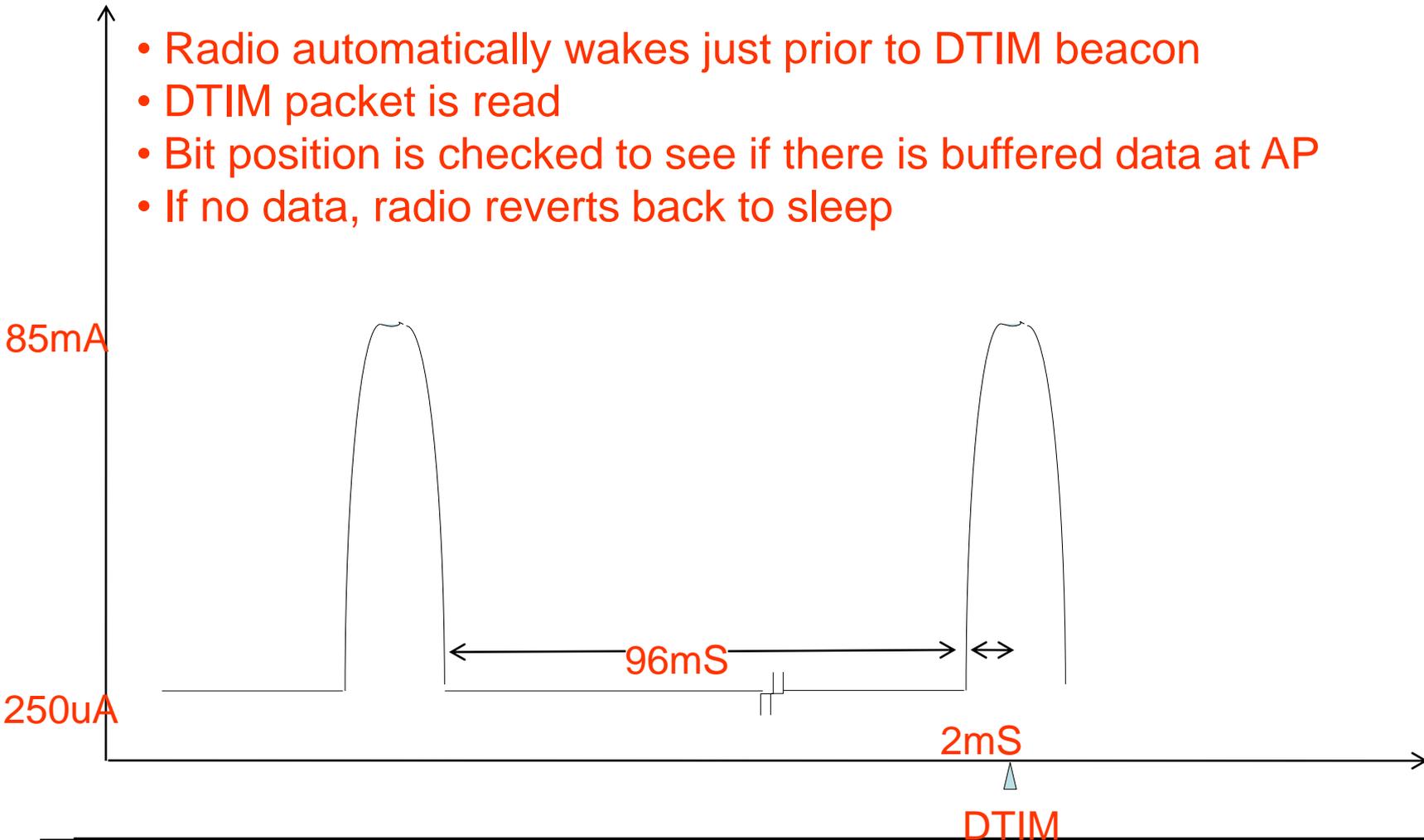
802.11 PS

- **Power Save or PS-Poll**
- **Use and limits**
 - Provides virtual “always on” power saving mode
 - Very low latency to send and receive
 - Requires PS supported Access Point
- **Power example**
 - Case: always connected w/sleep, hourly access for 500B transfer
 - ◆ 2 week battery life (just radio)



802.11PS

- Radio automatically wakes just prior to DTIM beacon
- DTIM packet is read
- Bit position is checked to see if there is buffered data at AP
- If no data, radio reverts back to sleep





Saved State

- **Use and limits**
 - After first connection, saves configuration data in NVM
 - Provides very low power fast on/off activity
 - Very low latency (100ms) to restart, send turn off
 - Many routers will not allow indefinite sleep
 - Best for sending data more often than every minute
- **Power**
 - Can achieve years of battery life for hourly connection with dynamic IP
 - RN Sleep: 4uA w/ RTC & Active Sensors



Hibernate

- **Hibernate use and limits**
 - Radio is turned off on-chip
 - Need to reconnect when turned on
 - Takes 1-10s to restart
- **Power example**
 - Case: unit powers on hourly for 500B transfer, static IP
 - ◆ MRF24WB0M: Hibernate/sleep
 - 4 year dual AA battery life (just radio)

IEEE 802.11 Security

- Wireless Security Setting on Access Point
- Two subsystems:
 - A data encapsulation technique called WEP (Wired Equivalent Privacy)
 - An authentication algorithm called Shared Key Authentication
- Severe security weakness in WEP.
- WPA(Wi-Fi Protected Access), WPA2





Wireless Security Setting

LINKSYS®
A Division of Cisco Systems, Inc. Firmware Version: v8.00.0

Wireless-G Broadband Router WRT54G

Wireless

Setup Wireless Security Access Restrictions Applications & Gaming Administration Status

Basic Wireless Settings | Wireless Security | Wireless MAC Filter | Advanced Wireless Settings

Wireless Security

Security Mode: WEP

Default Transmit Key: 1 2 3 4

WEP Encryption: 64 bits 10 hex digits 64 bits 10 hex digits 128 bits 26 hex digits

Passphrase:

Key 1:

Key 2:

Key 3:

Key 4:

Security Mode: You may choose from **Disable**, **WPA Personal**, **WPA Enterprise**, **WPA2 Personal**, **WPA2 Enterprise**, **RADIUS**, **WEP**. All devices on your network must use the same security mode in order to communicate. **More...**

CISCO SYSTEMS

LINKSYS®
A Division of Cisco Systems, Inc. Firmware Version: v8.00.0

Wireless-G Broadband Router WRT54G

Wireless

Setup Wireless Security Access Restrictions Applications & Gaming Administration Status

Basic Wireless Settings | Wireless Security | Wireless MAC Filter | Advanced Wireless Settings

Wireless Security

Security Mode: WPA Personal

WPA Algorithms: TKIP

WPA Shared Key:

Group Key Renewal: 3600 seconds

Security Mode: You may choose from **Disable**, **WPA Personal**, **WPA Enterprise**, **WPA2 Personal**, **WPA2 Enterprise**, **RADIUS**, **WEP**. All devices on your network must use the same security mode in order to communicate. **More...**

CISCO SYSTEMS

LINKSYS®
A Division of Cisco Systems, Inc. Firmware Version: v8.00.0

Wireless-G Broadband Router WRT54G

Wireless

Setup Wireless Security Access Restrictions Applications & Gaming Administration Status

Basic Wireless Settings | Wireless Security | Wireless MAC Filter | Advanced Wireless Settings

Wireless Security

Security Mode: WPA2 Personal

WPA Algorithms: AES

WPA Shared Key:

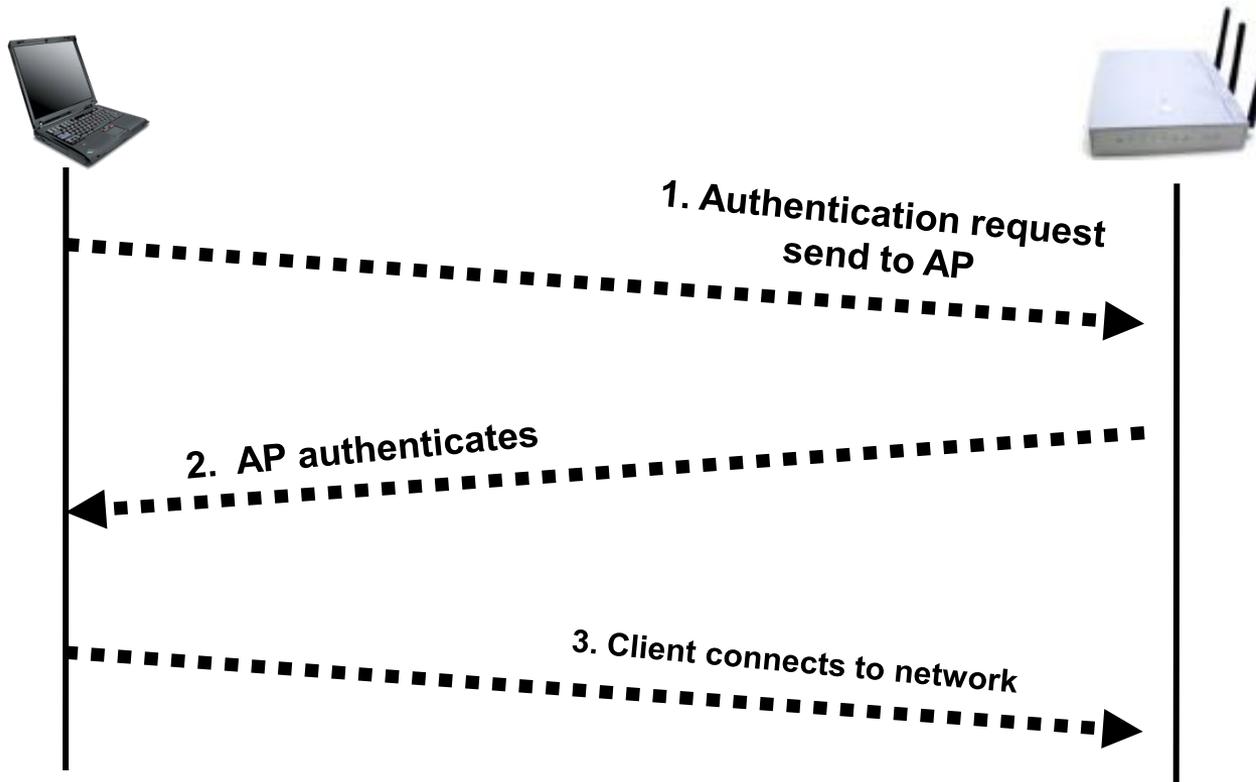
Group Key Renewal: 3600 seconds

Security Mode: You may choose from **Disable**, **WPA Personal**, **WPA Enterprise**, **WPA2 Personal**, **WPA2 Enterprise**, **RADIUS**, **WEP**. All devices on your network must use the same security mode in order to communicate. **More...**

CISCO SYSTEMS

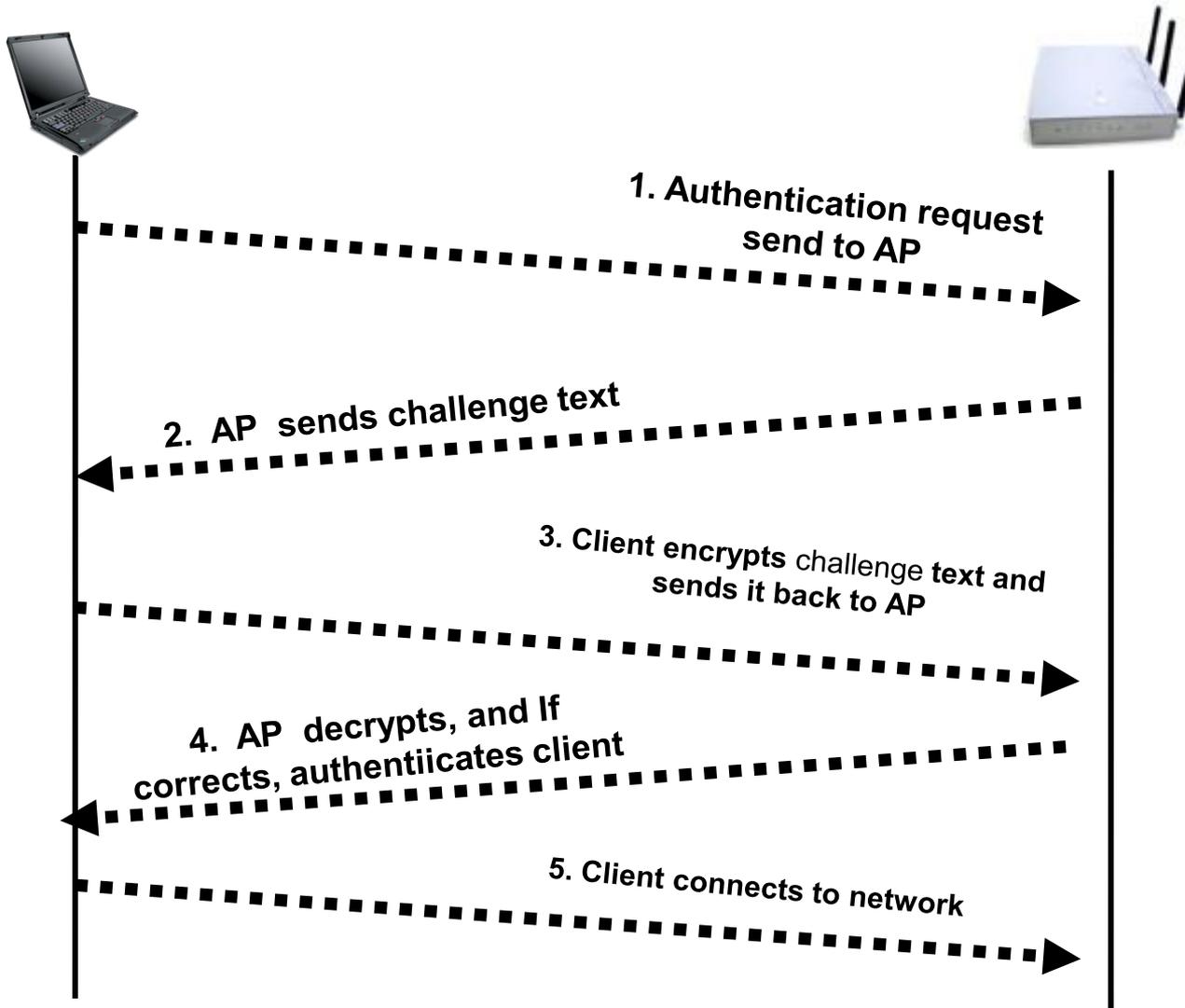


Open System Authentication

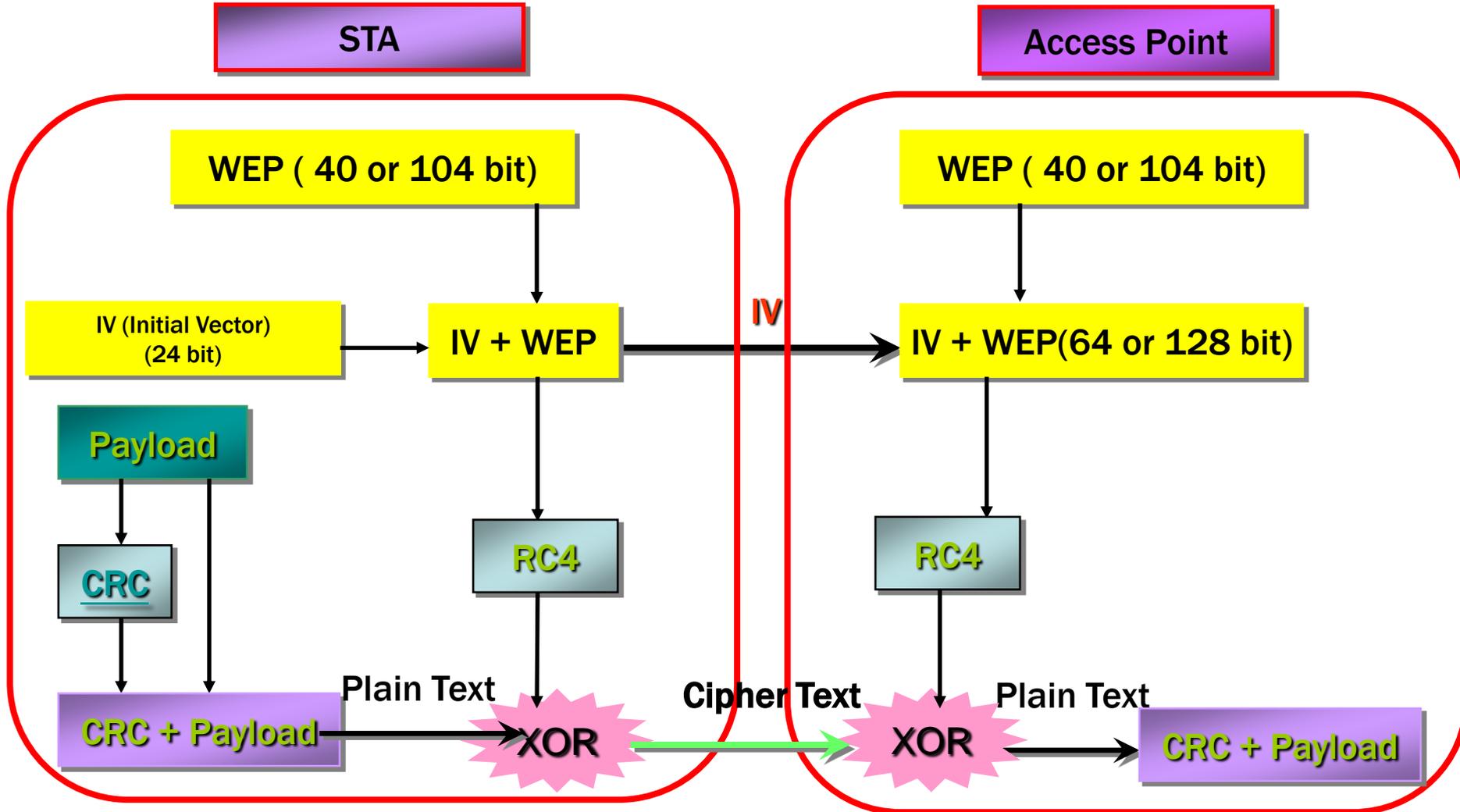




WEP Shared Key Authentication



WEP-Encryption/Decryption



RC4(Rivest Cipher), is the most widely-used software stream cipher .

IV: 24 bit initialization vector, $2^{24} = 16777216$



WEP Weaknesses

- The IV is too small and in cleartext.
- The IV is static (0 → 16777216)
- The IV makes the key stream vulnerable.
- The IV is a part of the RC4 encryption key
- WEP provides no cryptographic integrity protection

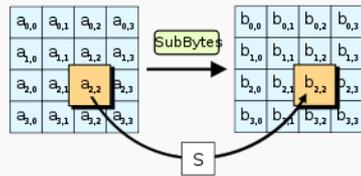


WEP VS WPA/WPA2

	WEP	WPA	WPA2
Cipher	RC4	RC4	AES
Key Size	40 bits	128 bits	128 bits
Key Life	24-bit IV	48-bit IV	48-bit IV
Data Integrity	CRC-32	Michael	CCM
Header Integrity	None	Michael	CCM

WI-FI Alliance: http://www.wi-fi.org/knowledge_center/webcast-wpa-061103/

AES(Advanced Encryption Standard)



Michael: Message Integrity Code(MIC)

CCM (Counter-Mode/CBC-MAC)

Electromagnetic Spectrum

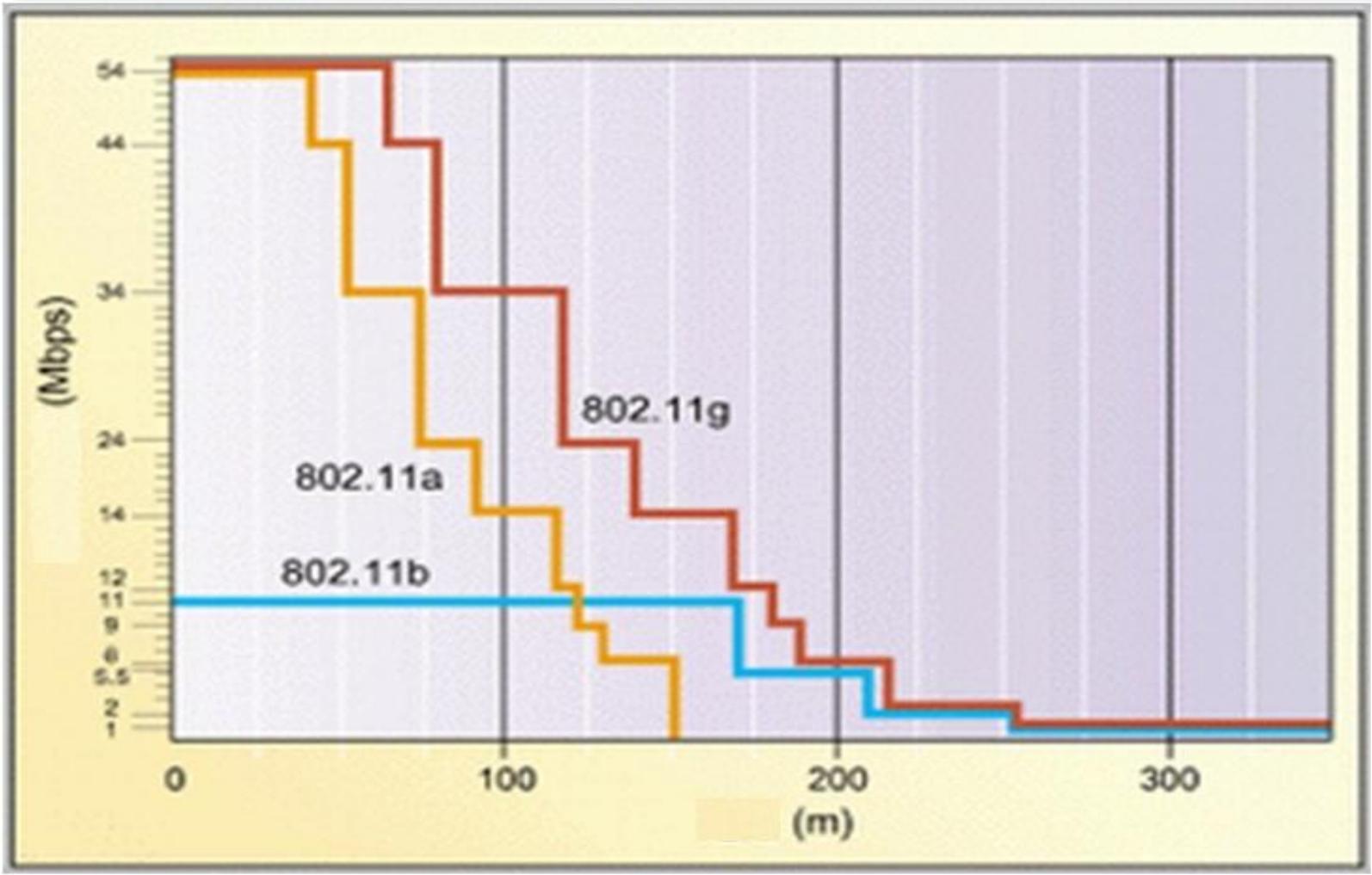
European
Union (Spain
and France
not included)

CHNL_ID	Frequency	Regulatory domains					
		USA domain= FCC	Canada domain= IC	domain= ETSI	domain= Spain	domain= France	domain= JapanA/B
1	2412 MHz	X	X	X	—	—	—
2	2417 MHz	X	X	X	—	—	—
3	2422 MHz	X	X	X	—	—	—
4	2427 MHz	X	X	X	—	—	—
5	2432 MHz	X	X	X	—	—	—
6	2437 MHz	X	X	X	—	—	—
7	2442 MHz	X	X	X	—	—	—
8	2447 MHz	X	X	X	—	—	—
9	2452 MHz	X	X	X	—	—	—
10	2457 MHz	X	X	X	X	X	—
11	2462 MHz	X	X	X	X	X	—
12	2467 MHz	—	—	X	—	X	—
13	2472 MHz	—	—	X	—	X	—
14	2484 MHz	—	—	—	—	—	X

JapanB=domain

JapanA=domain

Comparison of 802.11 Protocol





MICROCHIP

Regional Training Centers

6. Microchip TCP/IP Stack
(Software Architecture)

Microchip's TCP/IP Protocol Stack

- **Component of the Microchip TCPIP Framework**
- **FREE!! Royalty-free, no license fees**
- **Latest version available for download at:**
www.microchip.com/mla
- **Modular design**
 - **Compile only what you need**
- **No lower layer design required within the TCP/IP protocol**
- **Optimized for PIC18, PIC24, dsPIC[®] DSCs and PIC32**
 - **Includes BSD RTOS support via a wrapper**
- **Source code provided**
- **Support Documentation**
 - **Compiled HTML page distributed with the stack**

Application Examples (Services)

HTTP

Hyper-Text Transfer Protocol (Server)

Serves your web pages and processes web form input

SMTP

Simple Mail Transfer Protocol (Client)

Sends e-mail messages

Telnet

Telnet (Server)

Command-line interface

SNMP

Simple Network Management Protocol (Server)

Enterprise monitoring and control (AN870)

SNTP

Simple Network Time Protocol (Client)

Determines the absolute time

DHCP

Dynamic Host Configuration Protocol (Both)

Automatic IP and network configuration

DNS

Domain Name Service (Client)

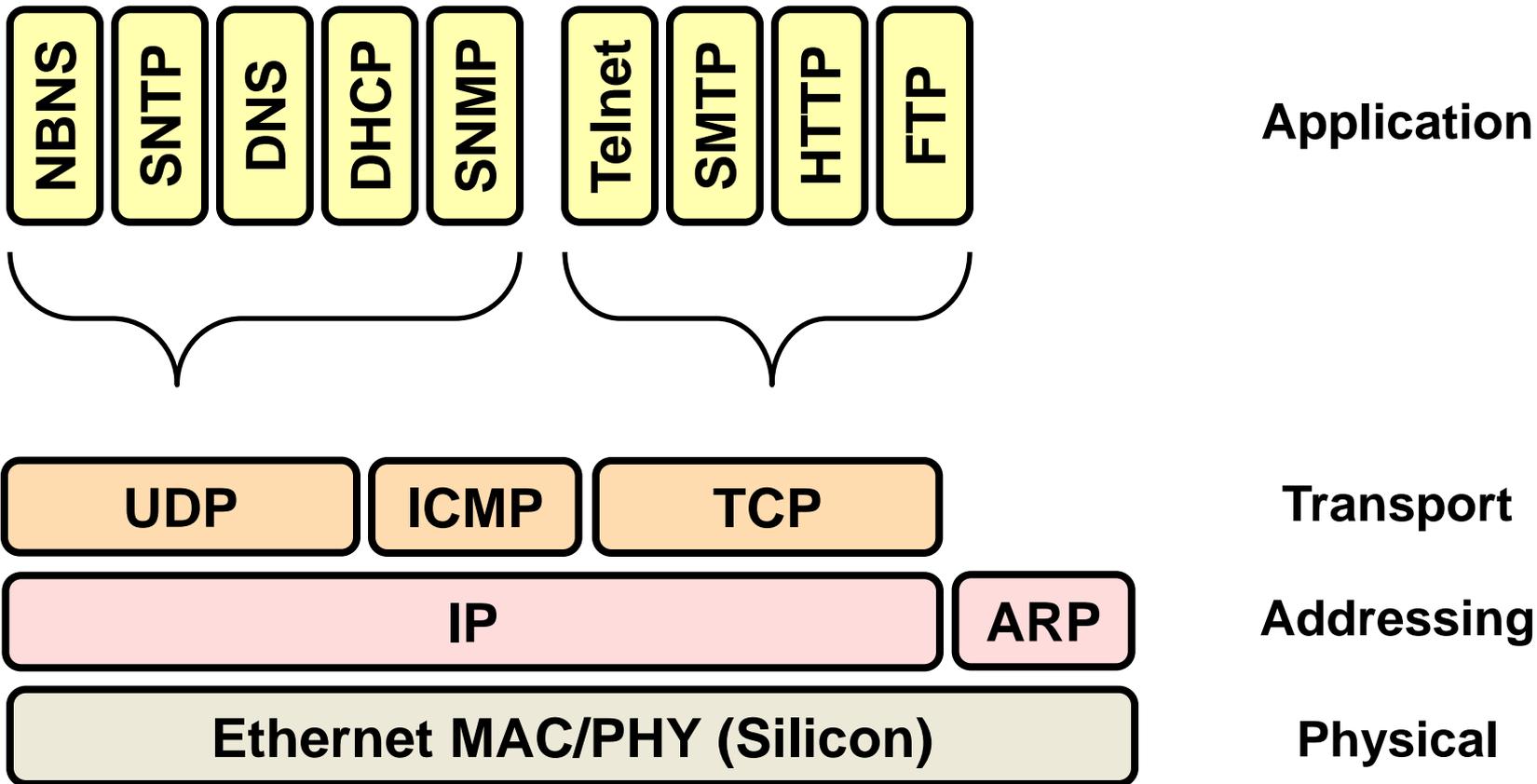
Global host name resolution

IPv6

IPv6

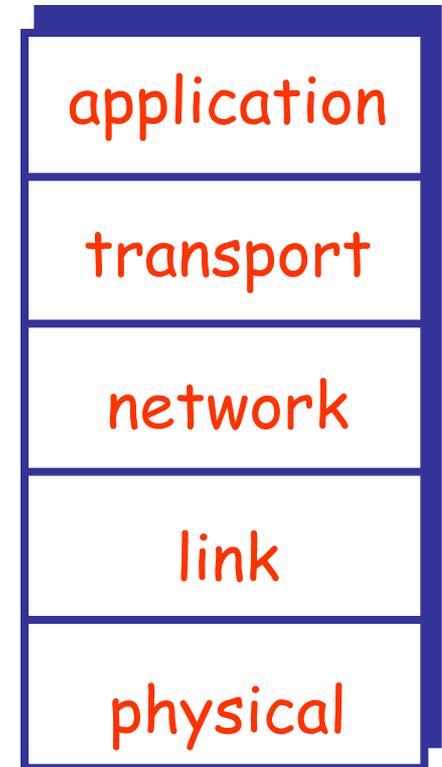


What's Included?



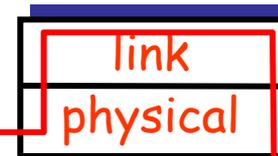
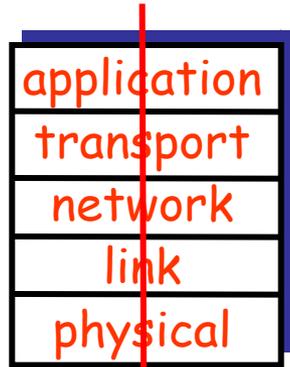
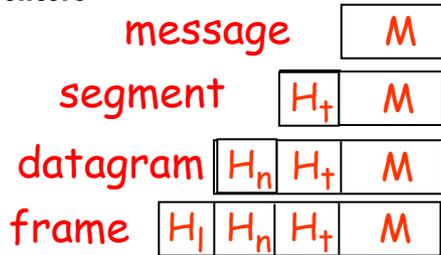
Internet (TCP/IP) Protocol Stack

- **Application:** supporting network applications
 - FTP, SMTP, HTTP, Custom
- **Transport:** process-process data transfer
 - TCP, UDP
- **Network:** routing of datagrams from source to destination
 - IP, routing protocols
- **Link:** data transfer between neighboring network elements
 - PPP, Ethernet
- **Physical:** bits “on the wire”



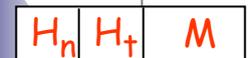
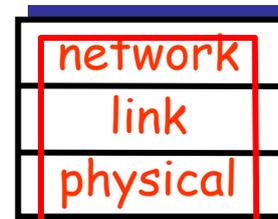
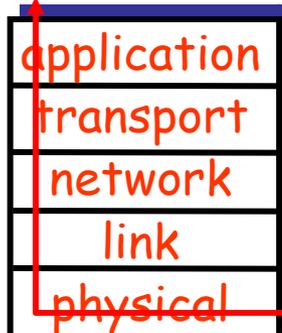
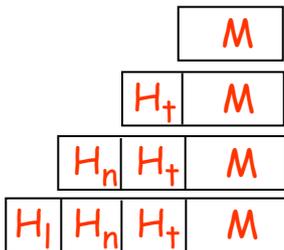


Encapsulation



switch

destination



router



About Addresses

- **Telephone System:**
 - **Must supply a number to the system before you can talk to someone**
- **Internet:**
 - **End-end communication requires**
 - ◆ **MAC Address**
 - ◆ **IP Address**
 - ◆ **Port Number**



MAC Addresses

- **MAC Addresses**
 - **Associated with hardware on the local network**
 - ◆ Only usable within the local network
 - ◆ Not globally routable
 - **Specific to IEEE 802.x networks**
 - ◆ Ethernet, ZigBee[®], Wi-Fi[®], etc.
 - **Six/Eight bytes: e.g. 00:04:A3:00:12:34**
 - ◆ IEEE “EUI-48/64” format
 - ◆ First 3 bytes are Vendor OUI code
 - ◆ Last 3/5 bytes are serialized in production
 - **Sold in blocks by the IEEE**
 - ◆ Globally unique
 - ◆ US\$550 / 4096 pcs or US\$1650 / 16M pcs
 - **Preprogrammed into MRF24WB0MA**



IP Addresses

- **Internet Protocol (IP) Addresses**
 - **Network Address: Globally Routable**
 - **Tied to the part of the network that the host is connected to**
 - **Assigned by software (DHCP or manual)**
 - ◆ Initial/Static address can be defined in TCPCConfig.h
 - **Four bytes as dotted-quad: 192.168.1.100**
 - ◆ New version (IPv6) will have 128 bits
 - **Administered by regional authorities**
 - ◆ Typically leased by your ISP
 - **Some allocated for private networks**
 - ◆ 192.168.*, 10.*, 169.254.*, and 172.16.*
 - ◆ Behind firewall and/or router - Not globally routable

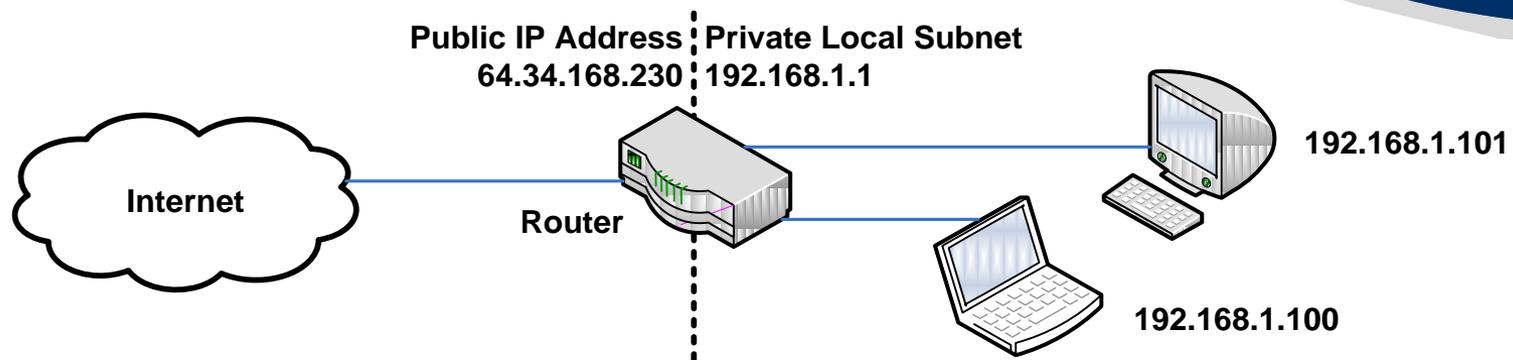


Port Number

- **Identifier of the target host application**
 - 16-bit unsigned integer (1 to 65535, 0 is reserved)
- **Some applications have standardized port #s**
 - **HTTP: 80**
- **Interpreted relative to an IP address**
 - E.g. To send HTTP message to `gaia.cs.umass.edu` web server:
 - ◆ **IP address:** 128.119.245.12
 - ◆ **Port number:** 80
- **Mail Analogy:**
 - **IP Address is the street address of the building**
 - **Port number is the room**

Private Network IP Addressing

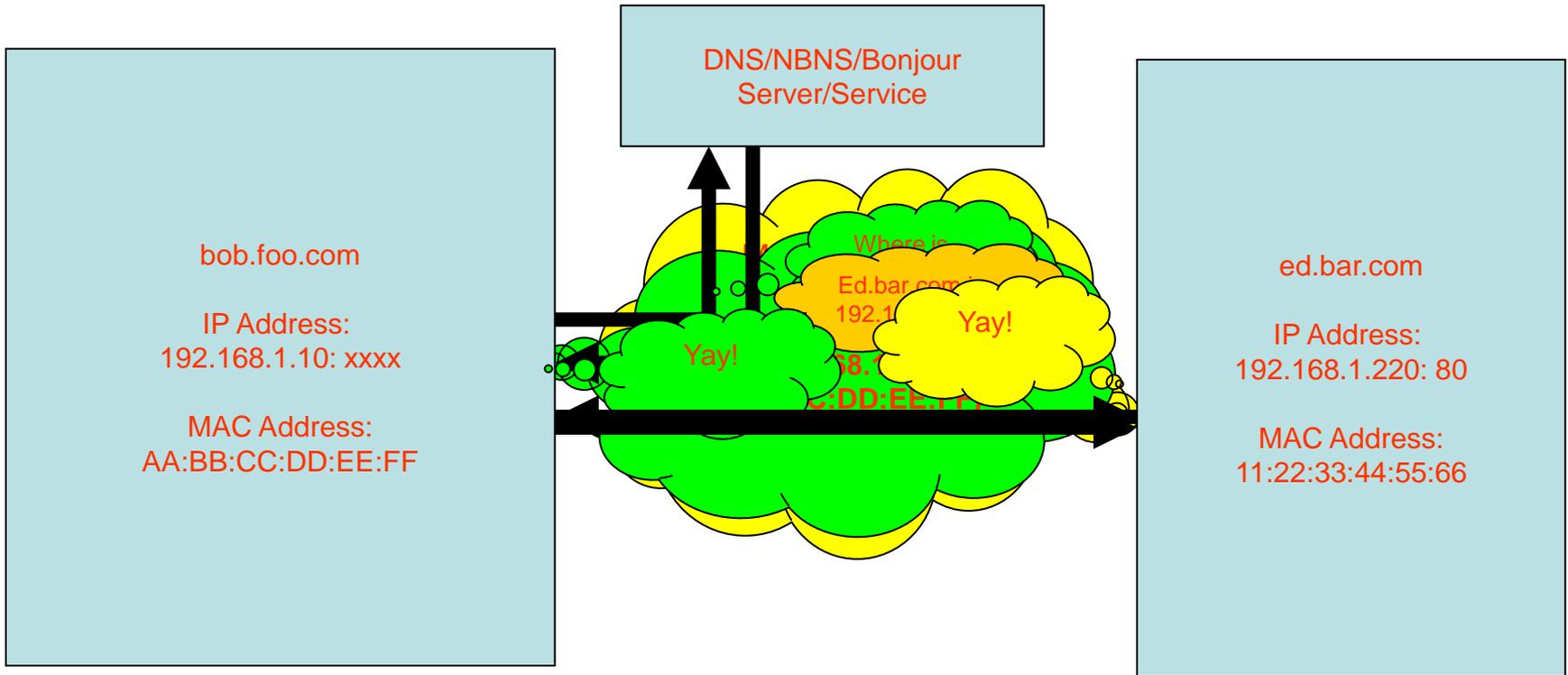
- 32-bit public address space insufficient
- NAT (Network Address Translation) provides one-to-many routing
 - Share one global IP among many machines
 - Router/gateway makes all outgoing connections
 - Can port-forward incoming connections
 - Also provides security





Name Resolution

Bob wants to send a Packet to Ed

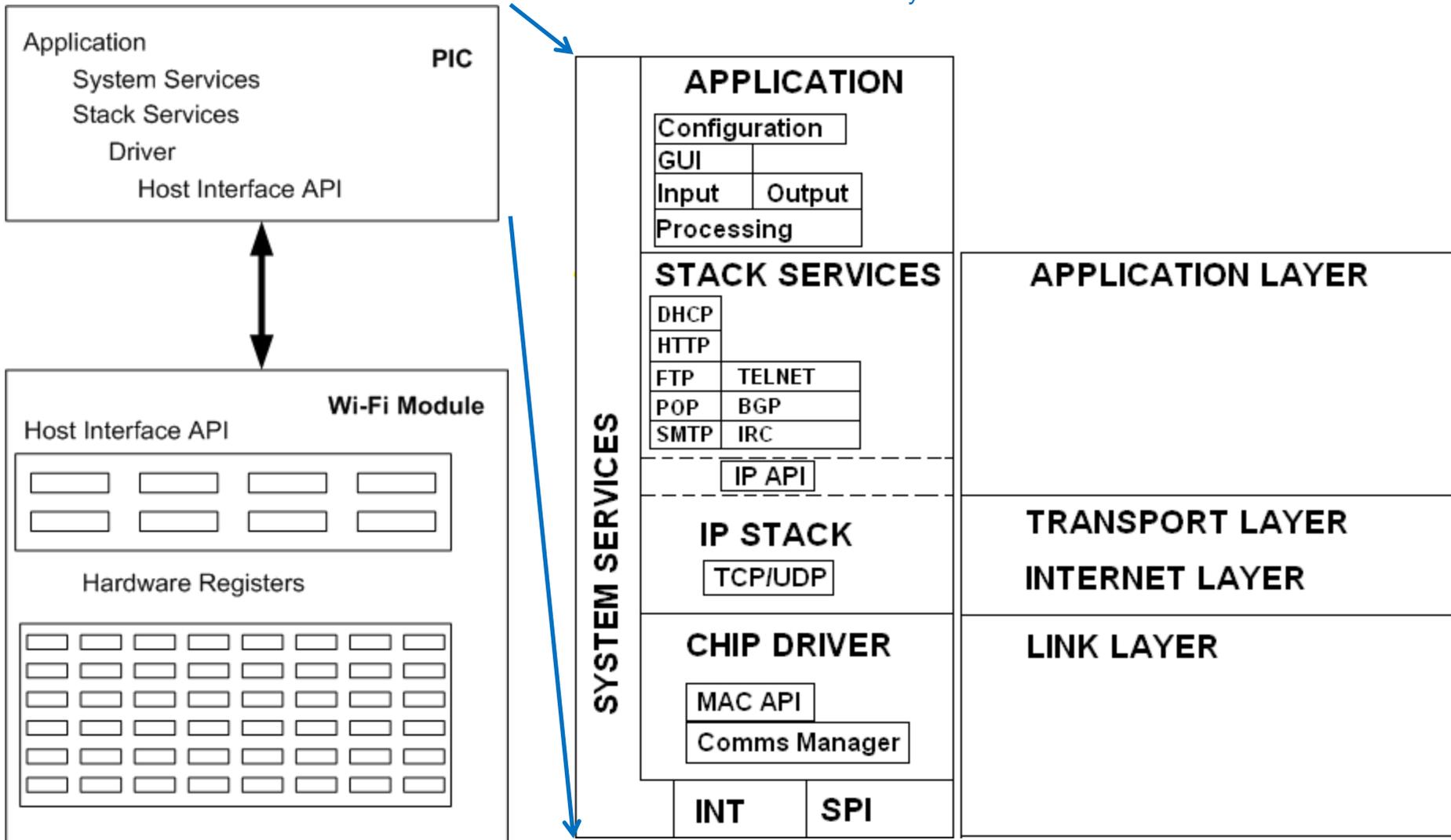


Bob sends an ARP looking for the hardware address associated with Ed's IP address.
 Bob sends a request to the DNS server asking for the address of Ed.
 Now both Bob and Ed know where to send packets of their conversation



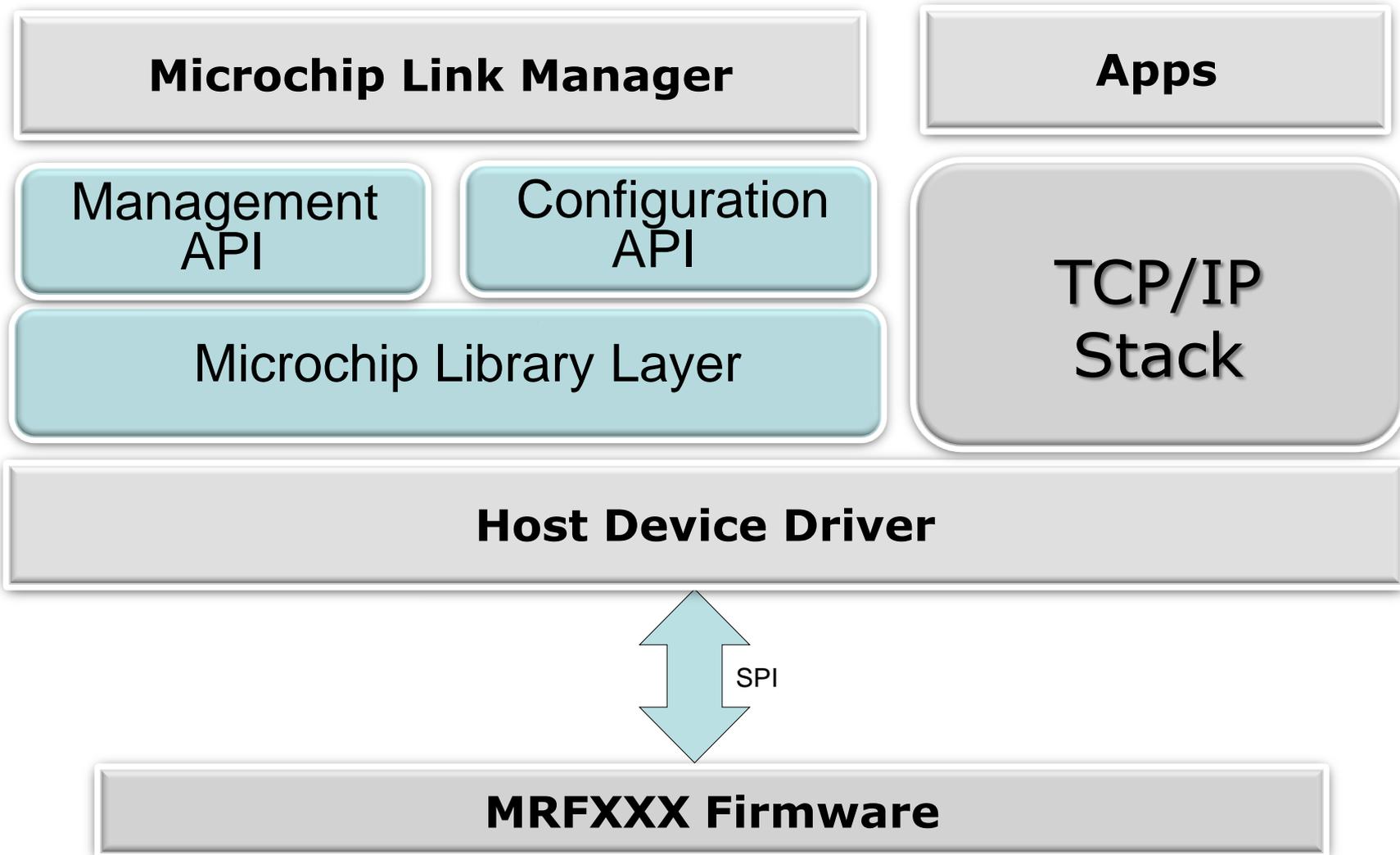
Microchip 802.11 Solution

System & Software Architecture





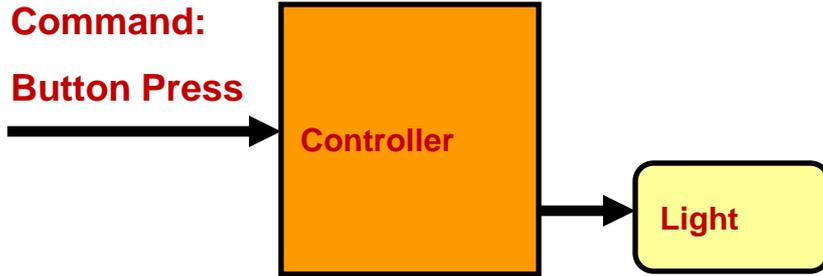
System Block Diagram



FSM(Finite-state machine) State Machine Basics

- **Definition:**
 - **A model of a system consisting of**
 - ◆ **States**
 - ◆ **Transitions between states**
 - ◆ **Actions**

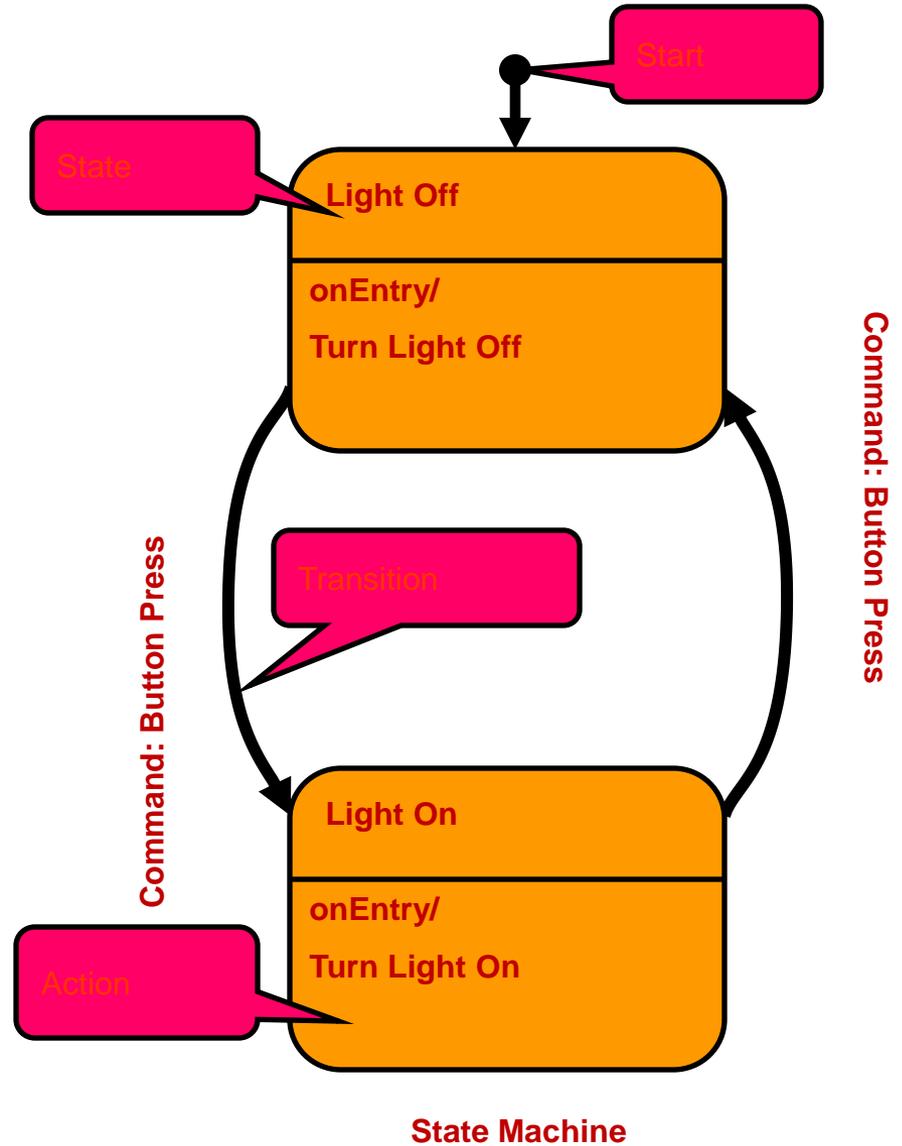
Example



Model of a push button flashlight

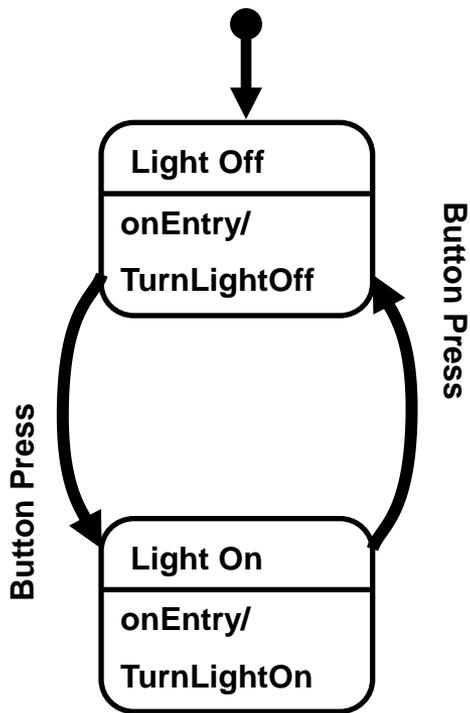
State \ Input	Light Off	Light On
Button Press	Light On	Light Off

Transition Table





Write the Code : Moore



Moore State Machine

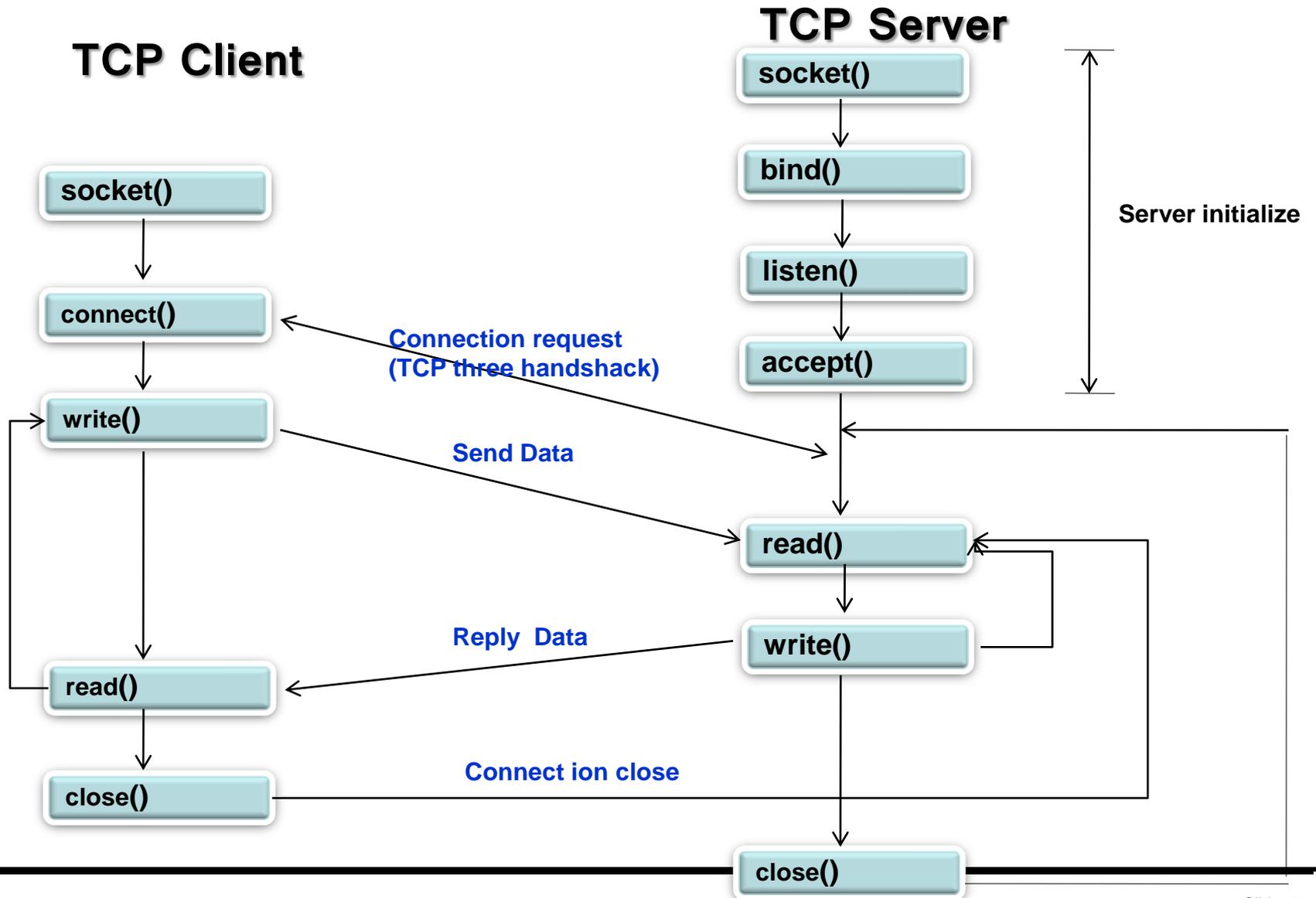
```
hw_init();
state=LightOff_Entry;

while(1) // Run the state machine forever
{
    switch(state)
    {
        case LightOff_Entry:
            TurnLightOff();
            state=LightOff;
        case LightOff:
            if(ButtonPress()) state = LightOn_Entry;
            break;

        case LightOn_Entry:
            TurnLightOn();
            state=LightOn;
        case LightOn:
            if(ButtonPress()) state = LightOff_Entry;
            break;
    }
}
```



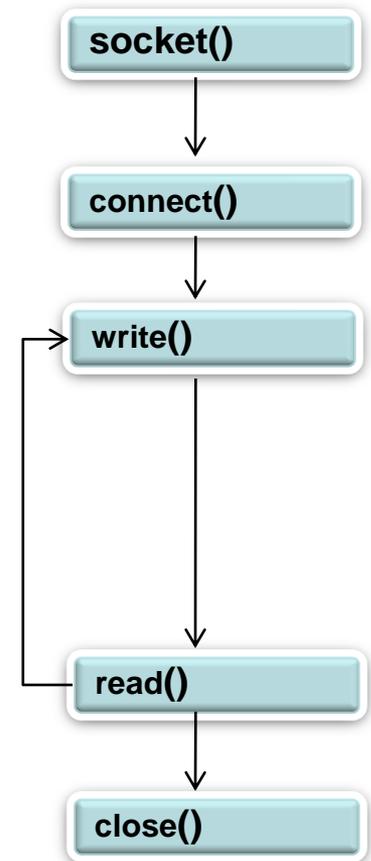
TCP Client-Server



Linux Client Source Code

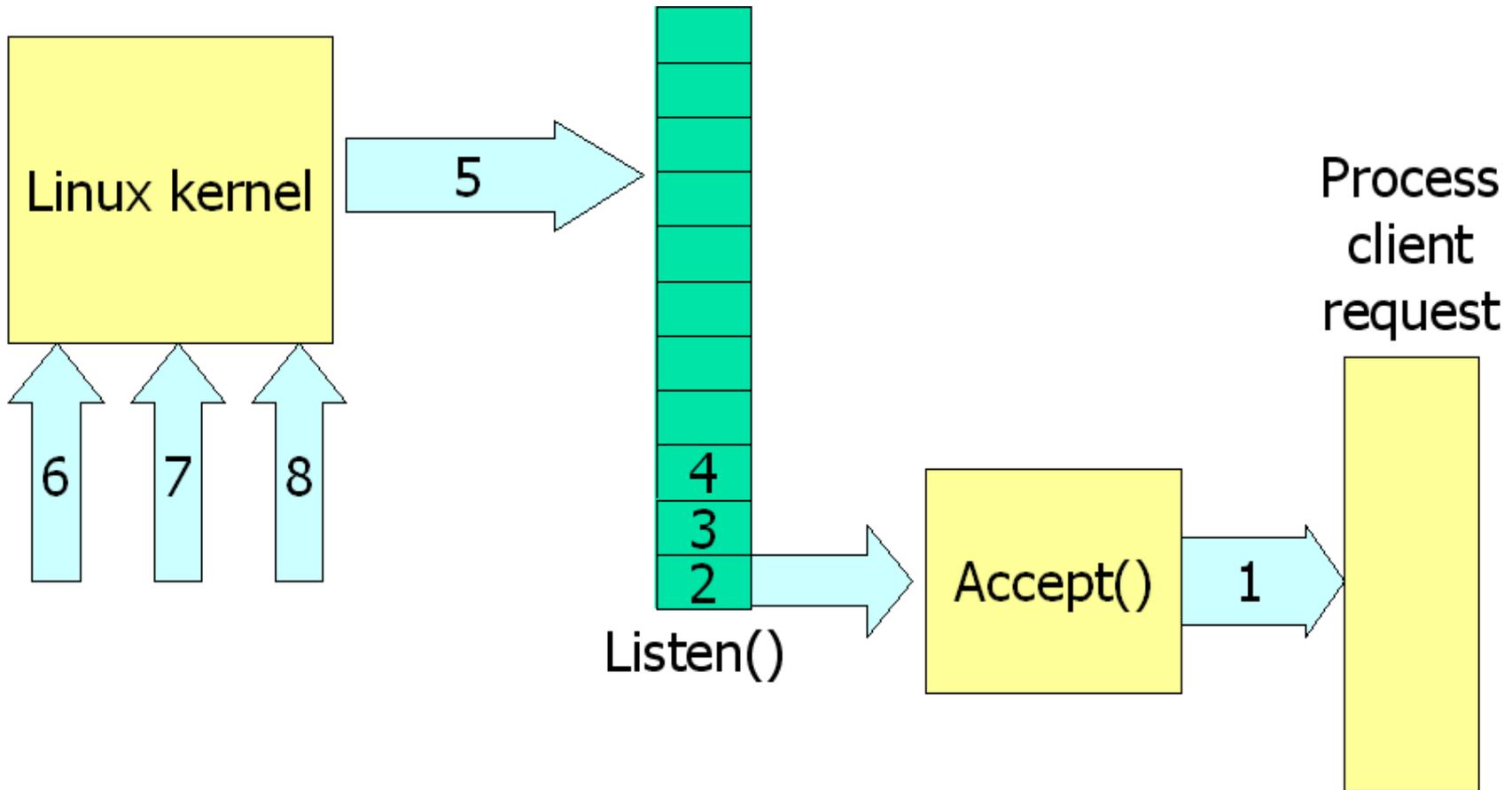
```
int main(int argc, char **argv)
{
    struct sockaddr_in addr_svr;
    int sockfd;
    char buffer[1024];
        // create server address
    memset(&addr_svr, 0, sizeof(addr_svr));
    addr_svr.sin_family= AF_INET;
    addr_svr.sin_port= htons(1234);
    addr_svr.sin_addr.s_addr = inet_addr("xxx.xxx.xxx.1");
    // create client socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    // connect
    connect(sockfd, (struct sockaddr *)&addr_svr, sizeof(addr_svr));
    // write to and read from server
    write(sockfd, buffer, strlen(buffer)+1);
    for(int len=0; ; ){
        len +=read(sockfd, buffer+len, 1024);
        if (len == 0) break;
    }
    close(sockfd);
    return 0;
}
```

TCP Client





Linux TCP/IP





Microchip-- Main Code

```
int main(void)
{
    // Initialize application specific hardware
    InitializeBoard();
    // Initialize stack and application related NV variables into AppConfig.
    InitAppConfig();
    StackInit();
    ZGConsoleInit( p_cmdStringPtrList, kZGNumCmdsInList);
    while(1)
    {
        // This task performs normal stack task including checking
        // for incoming packet, type of packet and calling
        // appropriate stack entity to process it.
        StackTask();

        // This tasks invokes each of the core stack application tasks
        StackApplications();
        ZGConsoleProcess();
        PingAppCall();
        BarcodeReaderDemo();
        ProcessIO();
    }
}
```

LOOP





Microchip—Client Source Code(1)

```
#define BARCodePORTNUM 1234
BYTE    sendBarCodeRequest[20] = "\0";
void BarCodeReaderDemo(void)
{
    #if defined(STACK_USE_DNS)
        static SOCKET bsdClientSocket = INVALID_SOCKET;
        static struct sockaddr_in addr;
        char recvBuffer[9];
        int i;
        int addrLen;
        IP_ADDR RemoteIP;
        static enum _BARCodeServerState
        {
            BARCode_START,
            BARCode_CONNECT,
            BARCode_SEND,
            BARCode_OPERATION,
            BARCode_CLOSE,
            BARCode_DONE,
        } BARCodeClientState = BARCode_DONE;

        switch(BARCodeClientState)
        {
```



Microchip—Client Source Code(2)

```
case BARDCode_START:
if (ZGConsoleIsConsoleMsgReceived() == kZGBoolTrue)
{
    if (isIPAddress(argv[1], address))
    {
        RemoteIP.v[0] = address[0]; RemoteIP.v[1] = address[1];
        RemoteIP.v[2] = address[2]; RemoteIP.v[3] = address[3];
        addr.sin_addr.S_un.S_addr=RemoteIP.Val;
        // Create a socket for this client to connect with
        if((bsdClientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP))
            == INVALID_SOCKET )

            return;

        memset(sendBarCodeRequest,0,strlen((char*) sendBarCodeRequest));
        strcpy((char*) sendBarCodeRequest,(char*) argv[2]);
        BARCodeClientState = BARDCode_CONNECT;
    }
    else
    {
        ZG_PUTSUART( (char *) "Error sendbarcode command\n\r");
        ZGConsoleReleaseConsoleMsg();
    }
}

break;
```



Microchip-- Main Code

```
int main(void)
{
    // Initialize application specific hardware
    InitializeBoard();
    // Initialize stack and application related NV variables into AppConfig.
    InitAppConfig();
    StackInit();
    ZGConsoleInit( p_cmdStringPtrList, kZGNumCmdsInList);
    while(1)
    {
        // This task performs normal stack task including checking
        // for incoming packet, type of packet and calling
        // appropriate stack entity to process it.
        StackTask();

        // This tasks invokes each of the core stack application tasks
        StackApplications();
        ZGConsoleProcess();
        PingAppCall();
        BarCodeReaderDemo();
        ProcessIO();
    }
}
```

LOOP





Microchip—Client Source Code(3)

```
case BARDCode_CONNECT:
// addr.sin_addr.S_un.S_addr destination IP address was set earlier in DNS step
addr.sin_port = BARDCodePORTNUM; //PORTNUM;
addrLen = sizeof(struct sockaddr);
if(connect(bsdClientSocket, (struct sockaddr*)&addr, addrLen) < 0)
    return;

BARDCodeClientState = BARDCode_SEND;
break;

case BARDCode_SEND:
//send TCP data
send(bsdClientSocket, (const char*)sendBarcodeRequest, strlen((char*)
                                                                    sendBarcodeRequest), 0);

BARDCodeClientState = BARDCode_OPERATION;
break;
```



Microchip-- Main Code

```
int main(void)
{
    // Initialize application specific hardware
    InitializeBoard();
    // Initialize stack and application related NV variables into AppConfig.
    InitAppConfig();
    StackInit();
    ZGConsoleInit( p_cmdStringPtrList, kZGNumCmdsInList);
    while(1)
    {
        // This task performs normal stack task including checking
        // for incoming packet, type of packet and calling
        // appropriate stack entity to process it.
        StackTask();

        // This tasks invokes each of the core stack application tasks
        StackApplications();
        ZGConsoleProcess();
        PingAppCall();
        BarCodeReaderDemo();
        ProcessIO();
    }
}
```

LOOP





Microchip—Client Source Code(4)

```
case BARDCODE_OPERATION:
    if(recv(bsdClientSocket, recvBuffer, 0, 2) < 0) //get the connection status
        BARDCODE_CLIENT_STATE = BARDCODE_CLOSE;
    while(1)
    {
        i = recv(bsdClientSocket, recvBuffer, sizeof(recvBuffer)-1, 0);
        //get the data from the recv queue
        if(i <= 0)
        {
            //putsUART((char*) "receive null data\r\n");
            break;
        }
        putsUART((char*)recvBuffer);
        BARDCODE_CLIENT_STATE = BARDCODE_CLOSE;
        break;
    case BARDCODE_CLOSE:
        closesocket(bsdClientSocket);
        BARDCODE_CLIENT_STATE = BARDCODE_DONE;
        break;
    case BARDCODE_DONE:
        ZGConsoleReleaseConsoleMsg();
        BARDCODE_CLIENT_STATE = BARDCODE_START;
        break;
    default:
        return;
}
```

Hand on TCPIP WiFi Console App

- **Lab 1: Uart command → connect wifi**
 - step1: copy TCPIP Demo App → TCPIP Demo App_UartCommand**
 - step2: add uart command**
 - step3: disable auto connect wifi**
 - step4: add connect wifi command**
- **Lab 1: Ping command**
 - step5: modify PingDemo.c**
 - step6: #define STACK_USE_ICMP_CLIENT**



Modify PingDemo.c

C:\2012_04_26_CDFiles_WiFiClass\Microchip Solutions v2012-04-03\Board Support Package

```

44 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHERWISE
45 *
46 *
47 * Author          Date          Comment
48 * ~~~~~
49 * E. Wood          4/26/08      Moved from MainDem
50 *****
51 #define __PINGDEMO_C
52
53 #include "TCPIPConfig.h"
54
55 #if defined(STACK_USE_ICMP_CLIENT)
56
57 #include "TCPIP Stack/TCPIP.h"
58 #include "MainDemo.h"
59
60 extern int isPING;
61
62 #define HOST_TO_PING "192.168.1.2" // Address that
63
64 /*****
65  Function:
66      void PingDemo(void)
67
68  Summary:
69      Demonstrates use of the ICMP (Ping) client.
70
71  Description:
72      This function implements a simple ICMP client. The
73      periodically by the stack, and it checks if BUTTON0
74      If the button is pressed, the function sends an ICMP
75      to a Microchip web server. The round trip time is

```

C:\2012_04_26_CDFiles_WiFiClass\Microchip Solutions v2012-04-03\TCPIP Demo App

```

44 * (INCLUDING NEGLIGENCE), BREACH OF WARRANTY, OR OTHER
45 *
46 *
47 * Author          Date          Comment
48 * ~~~~~
49 * E. Wood          4/26/08      Moved from Ma
50 *****
51 #define __PINGDEMO_C
52
53 #include "TCPIPConfig.h"
54
55 #if defined(STACK_USE_ICMP_CLIENT)
56
57 #include "TCPIP Stack/TCPIP.h"
58 #include "MainDemo.h"
59
60
61 #define HOST_TO_PING "www.microchip.com" // Addr
62 //#define HOST_TO_PING "192.168.1.2" // Address that
63
64 /*****
65  Function:
66      void PingDemo(void)
67
68  Summary:
69      Demonstrates use of the ICMP (Ping) client.
70
71  Description:
72      This function implements a simple ICMP client.
73      periodically by the stack, and it checks if BUT
74      If the button is pressed, the function sends an
75      to a Microchip web server. The round trip time

```



Modify PingDemo.c

```
92     static enum
93     {
94         SM_HOME = 0,
95         SM_GET_ICMP_RESPONSE
96     } PingState = SM_HOME;
97     static DWORD Timer;
98     LONG ret;
99     BYTE tmpoutput[50];
100     switch(PingState)
101     {
102         case SM_HOME:
103             // Send a ping request out if the u
104             if(isPING)
105             {
106                 // Don't ping flood: wait at
107                 if(TickGet() - Timer > lul*
108                 {
109                     // Obtain ownership
110                     if(!ICMPBeginUsage(
111                         break;
112                     // Update anti-ping
113                     Timer = TickGet();
114                     // Send ICMP echo r
115                     #if defined(STACI
116                     ICMPSendPin
117                     #else
118                     ICMPSendPin
119                     #endif
120                     PingState = SM_GET_
121                 }
122     }
```

```
92     static enum
93     {
94         SM_HOME = 0,
95         SM_GET_ICMP_RESPONSE
96     } PingState = SM_HOME;
97     static DWORD Timer;
98     LONG ret;
99
100     switch(PingState)
101     {
102         case SM_HOME:
103             // Send a ping request out if the
104             if(BUTTONO_IO == Ou)
105             {
106                 // Don't ping flood: wait
107                 if(TickGet() - Timer > lu
108                 {
109                     // Obtain ownersh
110                     if(!ICMPBeginUsage
111                         break;
112                     // Update anti-pin
113                     Timer = TickGet();
114                     // Send ICMP echo
115                     #if defined(STACI
116                     ICMPSendP
117                     #else
118                     ICMPSendP
119                     #endif
120                     PingState = SM_GET_
121                 }
122     }
```



Modify PingDemo.c

```

131         {
132             // Do nothing: still waiting
133             break;
134         }
135     else if(ret == -1)
136     {
137         // Request timed out
138         putsUART("\r\n Ping Timed out\r\n");
139         isPING=0;
140         PingState = SM_HOME;
141     }
142     else if(ret == -3)
143     {
144         // DNS address not resolved
145         putsUART("\r\n Can't resolve IP\r\n");
146         isPING=0;
147         PingState = SM_HOME;
148     }
149     else
150     {
151         // Echo received. Time elapsed
152         memset(tmpoutput,0,sizeof(tmpoutput));
153         sprintf((char *) tmpoutput,
154             "\r\nReply: %dms\r\n", (WORD)TickConvertToMsec);
155         putsUART(tmpoutput);
156         isPING=0;
157         PingState = SM_HOME;
158     }
159 }
160 // Finished with the ICMP module, reinitialize
161 ICMPEndUsage();
162 break;
163

```

```

134     }
135     else if(ret == -1)
136     {
137         // Request timed out
138         #if defined(USE_LCD)
139         memcpypgm2ram((void*)&LCDText,
140             LCDUpdate());
141         #endif
142         PingState = SM_HOME;
143     }
144     else if(ret == -3)
145     {
146         // DNS address not resolved
147         #if defined(USE_LCD)
148         memcpypgm2ram((void*)&LCDText,
149             LCDUpdate());
150         #endif
151         PingState = SM_HOME;
152     }
153     else
154     {
155         // Echo received. Time elapsed
156         #if defined(USE_LCD)
157         memcpypgm2ram((void*)&LCDText,
158             uitoa((WORD)TickConvertToMsec,
159             strcatpgm2ram((char*)&LCDText,
160             LCDUpdate());
161         #endif
162         PingState = SM_HOME;
163     }
164 }
165 // Finished with the ICMP module, reinitialize
166 ICMPEndUsage();

```



Modify BerkerlyTCPClientDemo.c

```

47  * Author          Date          Comment
48  *-----
49  * Aseem Swalah    4/17/08        Original
50  *-----
51
52  #include "TCPIPConfig.h"
53
54  #if defined(STACK_USE_BERKELEY_API)
55
56  #include "TCPIP Stack/TCPIP.h"
57
58  /*
59  #define PORTNUM 80
60  static ROM BYTE ServerName[] = "www.google.com";
61  // This is specific to this HTTP Client example
62  static BYTE sendRequest[] = "GET /search?as_q=Microchip&as
63  */
64  IP_ADDR RemoteIP;
65  #define PORTNUM 443
66  // This is specific to this HTTP Client example
67  static BYTE sendRequest[] = "4710060006145";
68  extern int isGlobalSend;
69
70
71  /*****
72  * Function:        void BerkeleyTCPClientDemo(void)
73  *
74  * PreCondition:    Stack is initialized()
75  *
76  * Input:           None
77  *
78  * Output:          None
79  *

```

```

--
47  * Author          Date          Comment
48  *-----
49  * Aseem Swalah    4/17/08        Original
50  *-----
51
52  #include "TCPIPConfig.h"
53
54  #if defined(STACK_USE_BERKELEY_API)
55
56  #include "TCPIP Stack/TCPIP.h"
57
58
59  #define PORTNUM 80
60  static ROM BYTE ServerName[] = "www.google.com";
61  // This is specific to this HTTP Client example
62  static BYTE sendRequest[] = "GET /search?as_q=Microchip&as
63
64  /*****
65  * Function:        void BerkeleyTCPClientDemo(void)
66  *
67  * PreCondition:    Stack is initialized()
68  *
69  * Input:           None
70  *
71  * Output:          None
72  *
73  * Side Effects:    None
74  *
75  * Overview:        None
76  *
77  * Note:            None
78  *****/

```



Modify BerkeleyTCPClientDemo.c

```
104 } BSDClientState = BSD_DONE;
105
106 if(isGlobalSend == 1)
107 {
108     BSDClientState = DNS_START_RESOLUTION;
109     isGlobalSend=0;
110 }
111
112
113 switch(BSDClientState)
114 {
115     case DNS_START_RESOLUTION:
116
117
118         RemoteIP.v[0] = 192;
119         RemoteIP.v[1] = 168;
120         RemoteIP.v[2] = 1;
121         RemoteIP.v[3] = 2;
122         addr.sin_addr.S_un.S_addr=RemoteIP
123         BSDClientState = BSD_START;
124         break;
125
126
127     case BSD_START:
128         // Create a socket for this client to connect w
129         if((bsdClientSocket = socket(AF_INET, SOCK_STR
130             return;
131
132         BSDClientState = BSD_CONNECT;
133         break;
134
135     case BSD_CONNECT:
136         // addr.sin_addr.S_un.S_addr destination IP add
```

```
100 switch(BSDClientState)
101 {
102     case DNS_START_RESOLUTION:
103         if(DNSBeginUsage())
104         {
105             DNSResolverROM(ServerName, DNS_TYPE
106             BSDClientState = DNS_GET_RESULT;
107         }
108         break;
109
110     case DNS_GET_RESULT:
111         if(!DNSIsResolved((IP_ADDR*)&addr.sin_addr
112             break;
113
114         if(!DNSEndUsage())
115         {
116             #if defined(STACK_USE_UART)
117             putsUART((ROM char*)"Coul
118             #endif
119             BSDClientState = BSD_DONE;
120             break;
121         }
122
123         BSDClientState = BSD_START;
124         // No break; here.
125
126     case BSD_START:
127         // Create a socket for this client to connect
128         if((bsdClientSocket = socket(AF_INET, SOCK_STR
129             return;
130
131         #if defined(STACK_USE_UART)
132         putsUART((ROM char*)"r\nr\nComm
```



Modify BerkerlyTCPClientDemo.c

```
104 } BSDClientState = BSD_DONE;
105
106 if(isGlobalSend == 1)
107 {
108     BSDClientState = DNS_START_RESOLUTION;
109     isGlobalSend=0;
110 }
111
112
113 switch(BSDClientState)
114 {
115     case DNS_START_RESOLUTION:
116
117
118         RemoteIP.v[0] = 192;
119         RemoteIP.v[1] = 168;
120         RemoteIP.v[2] = 1;
121         RemoteIP.v[3] = 2;
122         addr.sin_addr.S_un.S_addr=RemoteIP
123         BSDClientState = BSD_START;
124         break;
125
126
127     case BSD_START:
128         // Create a socket for this client to connect w
129         if((bsdClientSocket = socket(AF_INET, SOCK_STR
130             return;
131
132         BSDClientState = BSD_CONNECT;
133         break;
134
135     case BSD_CONNECT:
136         // addr.sin_addr.S_un.S_addr destination IP add
```

```
100 switch(BSDClientState)
101 {
102     case DNS_START_RESOLUTION:
103         if(DNSBeginUsage())
104         {
105             DNSResolverROM(ServerName, DNS_TYPE
106             BSDClientState = DNS_GET_RESULT;
107         }
108         break;
109
110     case DNS_GET_RESULT:
111         if(!DNSIsResolved((IP_ADDR*)&addr.sin_addr
112             break;
113
114         if(!DNSEndUsage())
115         {
116             #if defined(STACK_USE_UART)
117             putsUART((ROM char*)"Coul
118             #endif
119             BSDClientState = BSD_DONE;
120             break;
121         }
122
123         BSDClientState = BSD_START;
124         // No break; here.
125
126     case BSD_START:
127         // Create a socket for this client to connect
128         if((bsdClientSocket = socket(AF_INET, SOCK_STR
129             return;
130
131         #if defined(STACK_USE_UART)
132         putsUART((ROM char*)"r\n\r\nComm
```



Modify BerkerlyTCPClientDemo.c

```
155         i = recv(bsdClientSocket, recvBuffer, 1024, 0);
156
157         if(i == 0)
158             break;
159
160         if(i < 0) //error condition
161         {
162             BSDClientState = BSD_CLOSE;
163             putsUART((char*)"Connect failed");
164             break;
165         }
166
167         #if defined(STACK_USE_UART)
168         recvBuffer[i] = '\0'; // Null terminate data
169         putsUART((char*)recvBuffer);
170         BSDClientState = BSD_CLOSE;
171         #endif
172
173         if(BSDClientState == BSD_OPERATION)
174             break;
175     }
176     break;
177
178     case BSD_CLOSE:
179         closesocket(bsdClientSocket);
180         BSDClientState = BSD_DONE;
181         // No break needed
182         break;
183     case BSD_DONE:
184         // isGlobalSend=0;
185         putsUART((char*)" \n");
186         break;
187
188     default:
189         break;
190 }
```

```
159     while(1)
160     {
161         i = recv(bsdClientSocket, recvBuffer, 1024, 0);
162
163         if(i == 0)
164             break;
165
166         if(i < 0) //error condition
167         {
168             BSDClientState = BSD_CLOSE;
169             break;
170         }
171
172         #if defined(STACK_USE_UART)
173         recvBuffer[i] = '\0'; // Null terminate data
174         putsUART((char*)recvBuffer);
175         #endif
176
177         if(BSDClientState == BSD_OPERATION)
178             break;
179     }
180     break;
181
182     case BSD_CLOSE:
183         closesocket(bsdClientSocket);
184         BSDClientState = BSD_DONE;
185         // No break needed
186         break;
187     case BSD_DONE:
188         if(BUTTON2_IO == 0)
189             BSDClientState = DNS_START_RESOLUTION;
190         break;
191 }
```



MICROCHIP

Regional Training Centers

Appendix

WiFi Remote Controller (SoftAP mode)



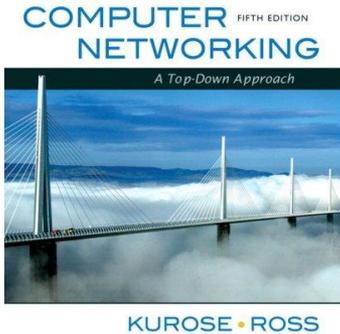
MICROCHIP

Regional Training Centers

Summary & References



References



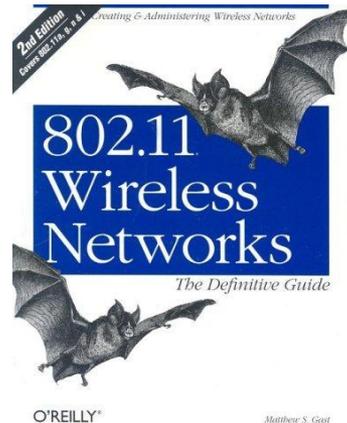
Computer Networking

A Top-Down Approach, 5th Edition

by James Kurose and Keith Ross

ISBN-10: 0136079679

ISBN-13: 978-0136079675



802.11 Wireless Networks

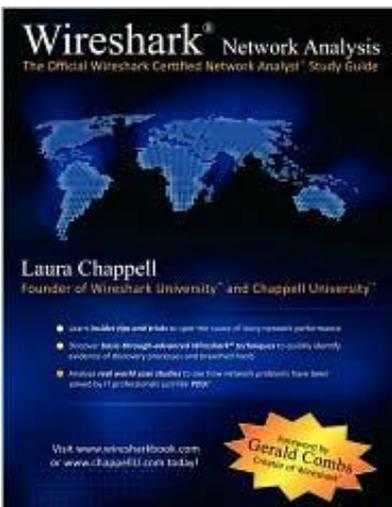
The Definitive Guide, 2nd Edition

by **Matthew Gast**

ISBN-10: 0596100523

ISBN-13: 978-0596100520

References



[Wireshark Network Analysis](#)

[The Official Wireshark Certified Network Analyst Guide](#)

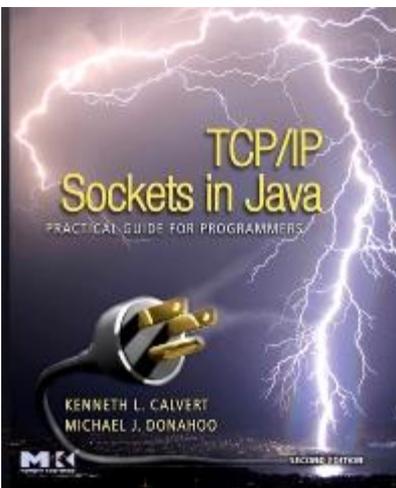
by Laura Chappell

ISBN-10: 1-893939-99-5

ISBN-13: 978-1-893939-8

www.wiresharkbook.com

www.wireshark.org



[TCP/IP Sockets in Java](#)

[Practical Guide for Programmers](#)

by Kenneth Calvert and Michael Donahoo

ISBN-10: 0123742552

ISBN-13: 978-0123742551

www.amazon.com



Additional Resources

- **Weblinks**

- www.microchip.com/wifi
- www.microchip.com/tcpip

- **Forums**

- forum.microchip.com

- **Training (HTTP, etc)**

- **Microchip RTC courses**

- **App Notes**

- **AN833 Original Microchip TCP/IP Stack**
- **AN1120 Ethernet Theory of Operation**



MICROCHIP

Regional Training Centers

Thank You

*Note: The Microchip name and logo, dsPIC, MPLAB and PIC are registered trademarks of Microchip Technology Inc. in the U.S.A. and other countries.
MIWi, PICDEM and PICtail are trademarks of Microchip Technology Inc. in the U.S.A. and other countries.
All other trademarks mentioned herein are property of their respective companies.*