CC32xx NWP Filter Application

Overview

Return to CC31xx & CC32xx Home Page

Return to CC31xx Sample Applications

The Rx-Filters feature enables the user to simply define, and manage the Rx-filtering process. It reduces the amount of traffic transferred to the host, and achieves an efficient power management.

General Description

- Every Rx frame traverses through a series of decision trees that determine how the frame should be handled.
- The decision trees are composed of filter nodes. Each node has its “filter rule”, “trigger” and “action”. The frame filtering process starts with the tree root nodes: for every node, if its “filter rule” and “trigger” are TRUE - the node’s action will be performed, and the frame will continue to the node’s child nodes.
- Possible **filter rule** is:

1. Protocol header rule: header type + compare function + arguments

- Possible **triggers** are:

1. When role is… (STA/AP/Promiscuous)
2. When connection state is… (connected/disconnected)
3. When device IP... (acquired/not acquired)
4. When counter reaches a certain threshold

- Possible **actions** are:

1. DROP the packet (do not pass it to the host)
2. Increment a counter (counter can be used in a child's filter to perform a certain action, such as dropping a frame when counter reaches some threshold)
3. Decrement a counter (counter can be used in a child's filter to perform a certain action, such as dropping a frame when counter reaches some threshold)
4. Reset a counter

The actions are executed only if the filter is matched to a received packet. Up to 2 actions can be defined per filter.

- There is an option to define a combined-filter-node, which its rule is a logical function on one or two other filters-nodes. For example: (node_1 OR node_2).
- Trees traversal is stopped when the frame reaches a DROP action in one of the trees. Traversing is done layer by layer among all the trees.
- A quick example
  Suppose a user has the following requirements:

1. Receive data broadcast frames only from two specific MAC addresses
2. Receive all unicast frames, except for frames with a certain SRC_IP address range
3. If a unicast frame is received from MAC address ‘AA.AA.AA’, turn on GPIO1
4. If a unicast frame is received from MAC address ‘BB.BB.BB’, turn on GPIO2
5. If a unicast UDP frame is received from MAC address ‘AA.AA.AA’ or ‘BB.BB.BB’, pass only packets from port ‘5001’

Creating Trees

- Trees are created by the user. The user adds the filter nodes and defines the filter's tree hierarchy
- Trees can be saved and loaded from the FLASH memory
- The maximal number of filter nodes is limited to 64

Host Interface

- The host supplies a set of APIs to enable filter creation\editing. The APIs include the following operations:

1. Add nodes - The user may request to add new node to the database, the node is created and a node unique ID is returned back to the user. The user should disable all filters before adding a node.
2. Remove nodes - The user may request to remove a node from the filters' database, the node is removed. The user should disable all filters before removing a node.
3. Enable nodes - The user can request to enable one or more filters, filters which are enabled, take part in the matching process.
4. Disable nodes - The user can request to disable one or more filters, filters which are disabled, don't take part in the matching process.
5. Get filter node
6. Update filter node

- The user's host application is responsible for adding filter nodes, and for defining the filters' tree hierarchy.
- If the user is interested in storing a filter node in the flash memory, he should set the persistence flag of the node. A filter can be defined as persistent only if its parent is persistent. Persistent filter nodes will be loaded automatically upon system startup.
- The user can choose one or more filters to disable\enable, filters which are disabled don't take part in the matching process.

Examples

The application creates and enables two filters to filter packets according to:
- Remote MAC address
- Remote IP address

It then connects to an AP and starts listening on PORT_NUM. TCP connection will be refused if remote MAC/IP is of the filtered. It's also possible to enable the filters after TCP connection, resulting in TCP packets to not being received. Please download the latest SDK for the complete example code.

- Change the Source Address and IP Address to match with the machine on which iperf is running.

```
//
// IP Address and MAC Address used in filters
//
unsigned char g_MacAddress[SL_MAC_ADDR_LEN] = {0xC0, 0xCB, 0x38, 0x1D, 0x09, 0x18};
unsigned char g_IpAddress[IP_ADDR_LENGTH] = {0xC0, 0xA8, 0x01, 0x78};
```

- Delete all stored filters. For deleting a specific filter, filter's ID should be set as input parameter

```
/* Delete all profiles (0xFF) stored */
sl_WlanProfileDel(0xFF);
```

- Build filter to drop incoming packets according to source MAC address

```
char                    cRetVal;
SlrxFilterID_t                  FilterId = 0;
```

```
SlrxFilterRuleType_t           RuleType;

SlrxFilterFlags_t                 FilterFlags;

SlrxFilterRule_t              Rule;

SlrxFilterTrigger_t            Trigger;

SlrxFilterAction_t             Action;


unsigned char ucMacMask[6]      = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

unsigned char ucIPMask[4]      = {0xFF, 0xFF, 0xFF, 0xFF};


/**********************************************************************/

/* Build filter to drop incoming packets according to source MAC address */

/**********************************************************************/


//

//define filter as parent

//

Trigger.ParentFilterID = 0;

//

//no trigger to activate the filter.

//

Trigger.Trigger = NO_TRIGGER;

//

//connection state and role

//

Trigger.TriggerArgConnectionState.IntRepresentation = RX_FILTER_CONNECTION_STATE_STA_CONNECTED;

Trigger.TriggerArgRoleStatus.IntRepresentation = RX_FILTER_ROLE_STA;

//

// header and Combination types are only supported. Combination for logical AND/OR between two filters

//

RuleType = HEADER;

Rule.HeaderType.RuleHeaderfield = MAC_SRC_ADDRESS_FIELD;

memcpy( Rule.HeaderType.RuleHeaderArgsAndMask.RuleHeaderArgs.RxFilterDB6BytesRuleArgs[0], g_ucMacAddress , SL_MAC_ADDR_LEN);

memcpy( Rule.HeaderType.RuleHeaderArgsAndMask.RuleHeaderArgsMask, ucMacMask , SL_MAC_ADDR_LEN);

Rule.HeaderType.RuleCompareFunc = COMPARE_FUNC_EQUAL;

//

//Action

//

Action.ActionType.IntRepresentation = RX_FILTER_ACTION_DROP;

FilterFlags.IntRepresentation = RX_FILTER_BINARY;

cRetVal = sl_WlanRxFilterAdd(RuleType, FilterFlags, &Rule, &Trigger, &Action, &FilterId);

if (cRetVal != 0)

    return -1;
```

- Build filter to drop incoming packets according to source IP address

```
Trigger.ParentFilterID = 0;

//

//no trigger to activate the filter.
```

```
//
Trigger.Trigger = NO_TRIGGER;
//
//connection state and role
//
Trigger.TriggerArgConnectionState.IntRepresentation = RX_FILTER_CONNECTION_STATE_STA_CONNECTED;
Trigger.TriggerArgRoleStatus.IntRepresentation = RX_FILTER_ROLE_STA;
//
// header and Combination types are only supported. Combination for logical AND/OR between two filters
//
RuleType = HEADER;
Rule.HeaderType.RuleHeaderfield = IPV4_SRC_ADRRESS_FIELD;
memcpy( Rule.HeaderType.RuleHeaderArgsAndMask.RuleHeaderArgs.RxFilterDB4BytesRuleArgs[0], g_ucIpAddress , IP_ADDR_LENGTH);
memcpy( Rule.HeaderType.RuleHeaderArgsAndMask.RuleHeaderArgsMask, ucIPMask , IP_ADDR_LENGTH);
Rule.HeaderType.RuleCompareFunc = COMPARE_FUNC_EQUAL;
//
//Action
//
Action.ActionType.IntRepresentation = RX_FILTER_ACTION_DROP;
FilterFlags.IntRepresentation = RX_FILTER_BINARY;
cRetVal = sl_WlanRxFilterAdd(RuleType, FilterFlags, &Rule, &Trigger, &Action, &FilterId);
if (cRetVal != 0)
    return -1;
```

- Enable all filters. For enabling a specific filter, filter's ID should be set as input parameter

```
_WlanRxFilterOperationCommandBuff_t     RxFilterIdMask ;


//Enable all 64 filters (8*8) - bit=1 represents enabled filter
memset( RxFilterIdMask.FilterIdMask, 0xFF , 8);
sl_WlanRxFilterSet( SL_ENABLE_DISABLE_RX_FILTER, (unsigned char *)&RxFilterIdMask, sizeof(_WlanRxFilterOperationCommandBuff_t));
```

Source Files briefly explained

- **main.c** - The main file that creates Rx Filters and manage the traffic accordingly

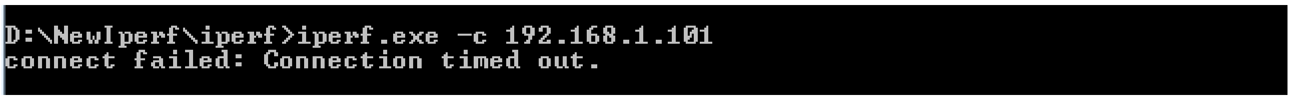
Supporting Files

- **startup_ccs.c** - CCS related functions
- **startup_ewarm.c** - IAR related functions

Usage

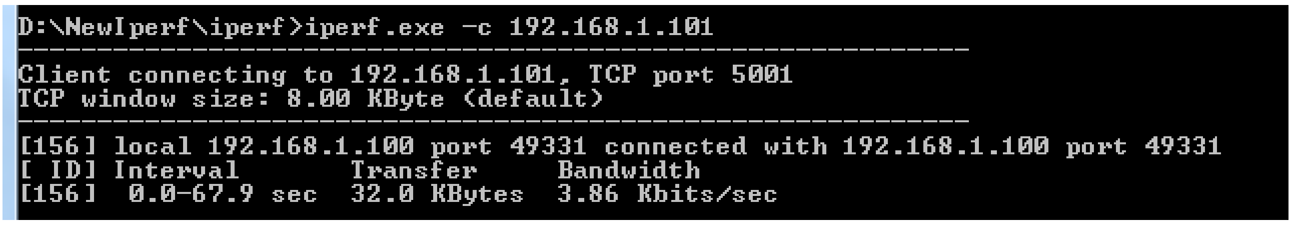
1. Change the IP Address and MAC Address to match with the machine on which iperf is running.
2. Change the SSID_NAME to required network to which you want to connect to.
3. Run the reference application
   - Open the Project in IAR/CCS. Build and download the application to the board
4. Once device connects to the network it acquires IP and you can view the IP address from the variable **pucCC31xx_Rx_Buffer**
5. Start iperf on the machine from where you want to send Packets.

6. The application Creates and Enables filters based on IP Address and MAC Address. Incoming packets are filtered based on the Rules defined.
7. Below screenshot shows you iperf result when you send Packets from Filtered IP or MAC



```
D:\NewIperf\iperf>iperf.exe -c 192.168.1.101
connect failed: Connection timed out.
```

8. Below screenshot shows you iperf result when you send packets from a machine which is not in filtered list.



```
D:\NewIperf\iperf>iperf.exe -c 192.168.1.101
------------------------------------------------------------
Client connecting to 192.168.1.101, TCP port 5001
TCP window size: 8.00 KByte (default)
------------------------------------------------------------
[156] local 192.168.1.100 port 49331 connected with 192.168.1.100 port 49331
[ ID] Interval       Transfer     Bandwidth
[156]  0.0-67.9 sec  32.0 KBytes  3.86 Kbits/sec
```

Limitations/Known Issues

None.

Article Sources and Contributors

CC32xx NWP Filter Application *Source*: http://processors.wiki.ti.com/index.php?oldid=178884 *Contributors*: A0221015, Codycooke, Jitgupta, Malokyle

Image Sources, Licenses and Contributors

File:Cc31xx cc32xx return home.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Cc31xx_cc32xx_return_home.png *License*: unknown *Contributors*: A0221015
File:Cc32xx return sample apps.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Cc32xx_return_sample_apps.png *License*: unknown *Contributors*: A0221015
Image:CC3101 NWP Filter1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:CC3101_NWP_Filter1.png *License*: unknown *Contributors*: Codycooke
Image:CC3101 NWP Filter2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:CC3101_NWP_Filter2.png *License*: unknown *Contributors*: Codycooke