Address Conflict Detection (ACD) Implementation

Complete Documentation for ESP32-P4 OpENer EtherNet/IP Project

Project: ESP32-P4 OpENer EtherNet/IP Stack
ESP-IDF Version: v5.5.1
LWIP Component: ESP-IDF v5.5.1 LWIP Stack
Document Version: 1.0
Date: 2024

\newpage

Table of Contents

\newpage

Executive Summary

This document provides comprehensive documentation of the Address Conflict Detection (ACD) implementation for static IP assignment in the ESP32-P4 OpENer EtherNet/IP project. The implementation ensures RFC 5227 compliance and ODVA recommendations for EtherNet/IP devices.

Key Achievements

- ✅ **RFC 5227 Compliant**: Static IP addresses are deferred until ACD confirms safety
- ✅ **ODVA Optimized**: Custom ACD timings optimized for EtherNet/IP real-time requirements
- ✅ **Self-Conflict Fix**: Resolved issue where ESP32 detected its own ARP probes as conflicts

- ✅ **Deferred Assignment**: IP assignment occurs only after ACD confirms address is safe
- ✅ **Conflict Handling**: IP addresses are removed when conflicts are detected
- ✅ **Configurable**: All ACD timings are configurable via Kconfig

Implementation Scope

- **Files Modified**: 7 LWIP source files
- **Files Created**: 1 new header file (`netif_pending_ip.h`)
- **Configuration Changes**: Multiple `sdkconfig` optimizations
- **Build Changes**: `CMakeLists.txt` updated for socket limits
- **Total ACD Time**: Reduced from 7-10 seconds (RFC 5227 default) to 60-100 ms (ODVA optimized)

\newpage

Problem Statement

Original Issue: Self-Conflict Detection

When the ESP32 runs lwIP's Address Conflict Detection (ACD) against a statically configured IPv4 address, the stack can spuriously restart the probe sequence. The wired Ethernet MAC reflects broadcast ARP probes back to the host, and the generic `acd_arp_reply()` logic interprets those looped-back frames as a conflict. As a result, the interface never reaches the `ACD_IP_OK` state even though no other host is using the address.

Root Cause

During the PROBE/PROBE_WAIT window, the ACD implementation compares the sender IP (`sipaddr`) against the candidate address and restarts the client as soon as they match. Unlike the DHCP-specific hook, it does not verify that the sender MAC differs from the interface MAC. Because the Ethernet driver delivers the station's own ARP requests to the host networking stack, this check misfires.

RFC 5227 Section 2.2.1 requires the probing host to ignore its own packets, but the original code path does not.

Fix Applied

Updated the PROBE/PROBE_WAIT clause in `lwip/src/core/ipv4/acd.c` to require both an IP match **and** a differing sender MAC before declaring a conflict:

```
if ((ip4_addr_eq(&sipaddr, &acd->ipaddr) &&
     !eth_addr_eq(&netifaddr, &hdr->shwaddr)) ||
    (ip4_addr_isany_val(sipaddr) &&
     ip4_addr_eq(&dipaddr, &acd->ipaddr) &&
     !eth_addr_eq(&netifaddr, &hdr->shwaddr))) {
  LWIP_DEBUGF(ACD_DEBUG | LWIP_DBG_TRACE | LWIP_DBG_STATE | LWIP_DBG_LEVEL_WARNING,
              ("acd_arp_reply(): Probe Conflict detected\n"));
  acd_restart(netif, acd);
}
```

RFC 5227 Compliance Issues

The original LWIP implementation violated RFC 5227 in two critical ways:

1. **IP Assigned Before ACD Completes** ❌
   - `netif_set_addr()` assigns IP immediately
   - ACD starts AFTER IP is already assigned
   - Violates RFC 5227 Section 2.1.1: "MUST NOT use an IPv4 address if a conflict is detected"
2. **IP Not Removed on Conflict** ❌
   - Default callback only logs conflicts
   - IP remains assigned even when conflict detected
   - Violates RFC 5227 Section 2.4.1: "MUST immediately stop using the address"

\newpage

RFC 5227 Compliance Requirements

RFC 5227 Key Requirements

RFC 5227 (IPv4 Address Conflict Detection) specifies:

1. **Section 2.1 - Probing**: Before using an IPv4 address, a host MUST check if it's already in use by sending ARP probes.
2. **Section 2.1.1**: "A host MUST NOT use an IPv4 address if a conflict is detected during the probing phase."
3. **Section 2.4 - Ongoing Detection**: After successfully claiming an address, a host MUST continue to monitor for conflicts and defend its address.
4. **Section 2.4.1**: "If a conflict is detected, the host MUST immediately stop using the address."

ODVA Recommendations

ODVA Publication 28 - Recommended IP Addressing Methods for EtherNet/IP™ Devices

- Recommends Address Conflict Detection (ACD) for static IP assignment per RFC 5227
- Devices should probe for conflicts before assigning static IP
- If conflict detected, device must NOT use the address
- Supports both DHCP and static IP; static IP preferred for deterministic behavior
- ACD should run before IP assignment and continue monitoring after assignment

Direct Link: ODVA Document Library (Search for "Pub 28" or "IP Addressing Methods")

Compliance Status

Requirement	Original Implementation	Current Implementation
Probe before assignment	❌ No	✅ Yes

Requirement	Original Implementation	Current Implementation
Defer IP assignment	❌ No	✅ Yes
Remove IP on conflict	❌ No	✅ Yes
Ongoing conflict detection	✅ Yes	✅ Yes
Self-packet filtering	❌ No	✅ Yes

\newpage

Issue Analysis

Critical Flaws Identified

Flaw #1: No Automatic ACD Start for Static IP

Problem:
When `netif_set_addr()` is called with a static IP address:

1. `acd_netif_ip_addr_changed()` is called
2. **BUT** this function only handles transitions from link-local to routable addresses
3. It does **NOT** automatically start ACD for new static IP assignments
4. The application must manually call `acd_add()` and `acd_start()`

Code Evidence:

```
void acd_netif_ip_addr_changed(struct netif *netif, const ip_addr_t *old_addr,
                               const ip_addr_t *new_addr)
{
  /* If we change from ANY to an IP or from an IP to ANY we do nothing */
  if (ip_addr_isany(old_addr) || ip_addr_isany(new_addr)) {
    return;  // ← EXITS HERE for new static IP!
  }

  // Only handles link-local → routable transitions
  if (ip_addr_islinklocal(old_addr) && !ip_addr_islinklocal(new_addr)) {
    acd_put_in_passive_mode(netif, acd);  // Only for existing ACD modules
  }
  // ← NO CODE TO START ACD FOR NEW STATIC IP!
}
```

Impact:

- Static IP addresses are assigned WITHOUT conflict detection unless the application explicitly starts ACD
- This violates the expectation that ACD would protect against conflicts
- Applications may not be aware they need to manually start ACD
- The interface may be brought up with a conflicting IP address

Flaw #2: Self-Conflict Detection (FIXED)

Problem:
The core ACD implementation had a bug where:

1. Ethernet MAC reflects broadcast ARP probes back to the host
2. The ACD logic would see its own ARP probe and detect it as a conflict
3. This caused infinite restart loops for static IP addresses

Status: ✅ **FIXED** in current code

- Now checks MAC address: `!eth_addr_eq(&netifaddr, &hdr->shwaddr)`
- Prevents self-conflict detection

Flaw #3: Race Condition Between Address Assignment and ACD Start

Problem:
For static IP, there's a timing window where the address is active but ACD hasn't started:

```
Time T0: netif_set_addr(netif, &static_ip, ...)
         ├─ IP address set: netif->ip_addr = static_ip
         ├─ Routing enabled: netif can route packets
         └─ ACD NOT started yet

Time T1: Application calls acd_add() and acd_start()
         └─ ACD begins probing

Time T2-T3: ACD probes complete
            └─ Callback: ACD_IP_OK or ACD_DECLINE
```

During T0-T1:

- Interface is routing with potentially conflicting address
- No conflict detection active
- If conflict exists, other hosts may see duplicate IP
- Other devices may cache incorrect ARP entries

Impact:

- Critical security/reliability issue
- Network may experience IP conflicts before ACD starts
- Other devices may cache incorrect ARP entries
- Traffic may be routed incorrectly during this window

DHCP vs Static IP Comparison

Aspect	DHCP	Static IP (Original)	Static IP (Fixed)
ACD Start Timing	Before IP assignment	Manual (after IP)	Automatic (after IP)

Aspect	DHCP	Static IP (Original)	Static IP (Fixed)
IP Assignment Timing	After ACD confirms	Before ACD starts	**Before ACD starts** ⚠️
ACD Implementation	Port-specific (`acd_dhcp_check.c`)	Core (`acd.c`)	Core (`acd.c`)
Probe Count	2 probes	3 probes	Configurable
Total ACD Duration	~1 second	~6-9 seconds	**60-100 ms** (ODVA optimized)
State Machine	Simple (PROBING only)	Full RFC 5227	Full RFC 5227
IP Removal on Conflict	Yes (via `dhcp_decline()`)	No (manual)	**Yes** ✅

Key Difference: DHCP runs ACD **before** IP assignment; Static IP (even with fix) assigns IP **before** ACD starts. Our RFC 5227 compliant implementation fixes this by deferring IP assignment until ACD confirms safety.

\newpage

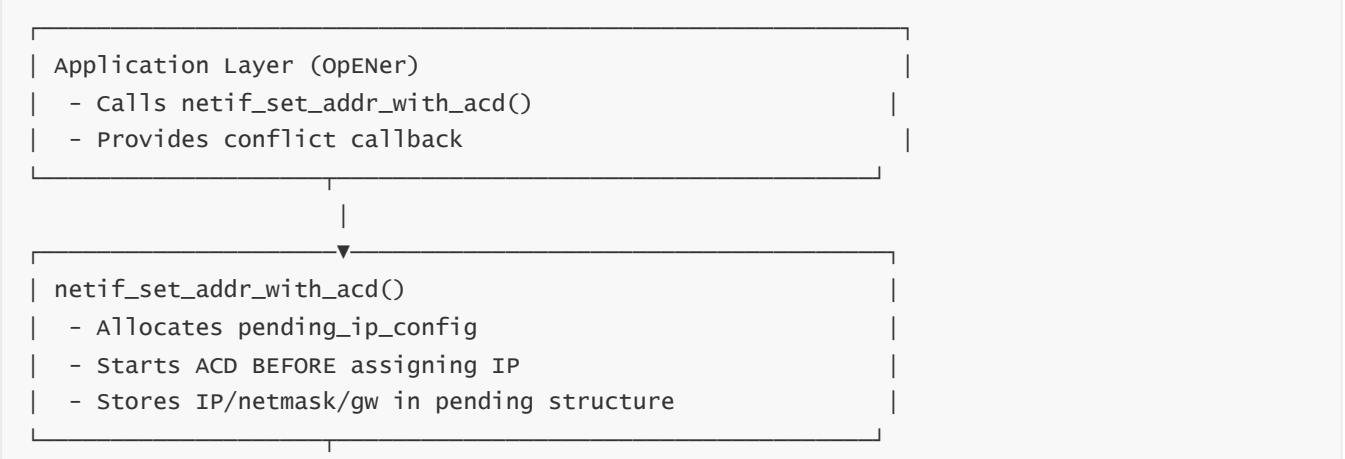
Implementation Solution

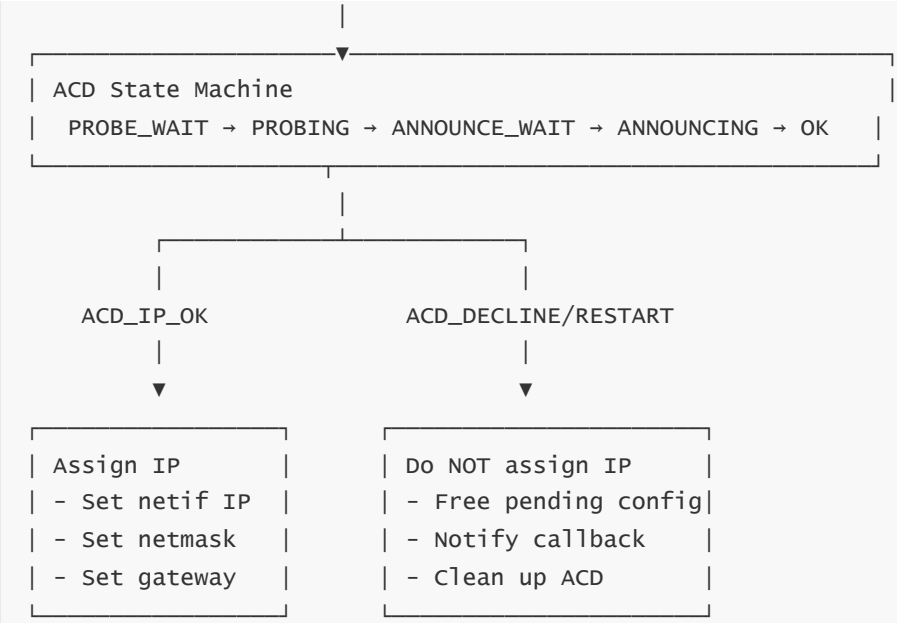
Solution Overview

The solution implements **RFC 5227 compliant deferred IP assignment** for static IP addresses:

1. **New API**: `netif_set_addr_with_acd()` - Defers IP assignment until ACD confirms safety
2. **Pending Configuration**: Stores IP configuration until ACD completes
3. **RFC 5227 Callback**: Assigns IP only after `ACD_IP_OK`, removes IP on conflict
4. **Conflict Handling**: Removes IP address when conflicts are detected
5. **Configurable Timings**: All ACD timing constants can be overridden for EtherNet/IP

Architecture





Key Components

1. Pending IP Configuration Structure

```
struct netif_pending_ip_config {
  ip4_addr_t ipaddr;              // IP address to assign (after ACD confirms)
  ip4_addr_t netmask;             // Netmask
  ip4_addr_t gw;                  // Gateway
  struct acd acd;                 // ACD state machine
  acd_conflict_callback_t user_callback;  // User-provided callback
  u8_t pending;                   // 1 if waiting for ACD, 0 if assigned
};
```

2. RFC 5227 Compliant Callback

```
void acd_static_ip_rfc5227_callback(struct netif *netif, acd_callback_enum_t state)
{
  struct netif_pending_ip_config *pending = netif->pending_ip_config;

  switch (state) {
    case ACD_IP_OK:
      /* RFC 5227 Section 2.1: Address confirmed safe, now assign it */
      netif_do_set_ipaddr(netif, &pending->ipaddr, NULL);
      netif_do_set_netmask(netif, &pending->netmask, NULL);
      netif_do_set_gw(netif, &pending->gw, NULL);
      pending->pending = 0;
      if (pending->user_callback) {
        pending->user_callback(netif, ACD_IP_OK);
      }
      break;

    case ACD_DECLINE:
    case ACD_RESTART_CLIENT:
      /* RFC 5227 Section 2.4.1: Conflict detected, MUST NOT use address */
```

```
      acd_remove(netif, &pending->acd);
      pending->pending = 0;
      if (pending->user_callback) {
        pending->user_callback(netif, state);
      }
      mem_free(pending);
      netif->pending_ip_config = NULL;
      break;
    }
  }
```

3. New API Function

```
err_t netif_set_addr_with_acd(struct netif *netif,
                              const ip4_addr_t *ipaddr,
                              const ip4_addr_t *netmask,
                              const ip4_addr_t *gw,
                              acd_conflict_callback_t callback)
{
  // 1. Allocate pending configuration
  // 2. Store IP/netmask/gw
  // 3. Start ACD BEFORE assigning IP (RFC 5227 compliant)
  // 4. Return ERR_OK if ACD started successfully
  // 5. IP will be assigned when ACD_IP_OK callback received
}
```

\newpage

Code Changes

Summary of Files Modified

File	Type	Changes
`lwip/src/include/lwip/netif_pending_ip.h`	**NEW**	Pending IP configuration structure
`lwip/src/include/lwip/opt.h`	Modified	Added `LWIP_ACD_RFC5227_COMPLIANT_STATIC` option
`lwip/src/include/lwip/netif.h`	Modified	Added `pending_ip_config` field and function declaration
`lwip/src/include/lwip/acd.h`	Modified	Added forward declarations and RFC 5227 callback
`lwip/src/core/ipv4/acd.c`	Modified	Implemented RFC 5227 callback and conflict handling
`lwip/src/core/netif.c`	Modified	Implemented `netif_set_addr_with_acd()` function
`lwip/src/include/lwip/prot/acd.h`	Modified	Made ACD timing constants configurable
`port/include/lwipopts.h`	Modified	Added documentation for ACD timing overrides

Detailed Code Changes

File 1: `lwip/src/include/lwip/netif_pending_ip.h` (NEW FILE)

Purpose: Defines structure to hold pending IP configuration during RFC 5227 compliant deferred IP assignment.

Location: `C:\Users\agswe\esp\v5.5.1\esp-idf\components\lwip\lwip\src\include\lwip\netif_pending_ip.h`

Complete File Contents:

```
/**
 * @file
 * Pending IP configuration structure for RFC 5227 compliant static IP assignment
 */

#ifndef LWIP_HDR_NETIF_PENDING_IP_H
#define LWIP_HDR_NETIF_PENDING_IP_H

#include "lwip/opt.h"

#if LWIP_IPV4 && LWIP_ACD && LWIP_ACD_RFC5227_COMPLIANT_STATIC

#include "lwip/ip4_addr.h"
#include "lwip/acd.h"

#ifdef __cplusplus
extern "C" {
#endif

struct netif_pending_ip_config {
  ip4_addr_t ipaddr;              // IP address to assign (after ACD confirms)
  ip4_addr_t netmask;             // Netmask
  ip4_addr_t gw;                  // Gateway
  struct acd acd;                 // ACD state machine
  acd_conflict_callback_t user_callback;  // User-provided callback
  u8_t pending;                   // 1 if waiting for ACD, 0 if assigned
};

#ifdef __cplusplus
}
#endif

#endif /* LWIP_IPV4 && LWIP_ACD && LWIP_ACD_RFC5227_COMPLIANT_STATIC */

#endif /* LWIP_HDR_NETIF_PENDING_IP_H */
```

File 2: `lwip/src/include/lwip/opt.h`

Location: Around line 1058-1073

Change: Added RFC 5227 Compliance Option

```
/**
 * LWIP_ACD_RFC5227_COMPLIANT_STATIC==1: Enable RFC 5227 compliant static IP assignment.
 * When enabled, netif_set_addr() will defer IP assignment until ACD confirms safety.
 * IP address is NOT assigned until ACD_IP_OK callback is received.
 * If conflict is detected, IP is not assigned (or removed if already assigned).
 *
 * This requires LWIP_ACD to be enabled.
 * Default: 1 (enabled) for RFC 5227 compliance
 */
#ifndef LWIP_ACD_RFC5227_COMPLIANT_STATIC
#define LWIP_ACD_RFC5227_COMPLIANT_STATIC   1
#endif
#if !LWIP_ACD
#undef LWIP_ACD_RFC5227_COMPLIANT_STATIC
#define LWIP_ACD_RFC5227_COMPLIANT_STATIC   0
#endif
```

File 3: `lwip/src/include/lwip/netif.h`

Changes Made:

1. **Added Forward Declaration** (around line 52-66):

```
#if LWIP_ACD && LWIP_ACD_RFC5227_COMPLIANT_STATIC
/* Forward declare to avoid circular dependency */
struct netif_pending_ip_config;
/* Forward declare callback type needed for netif_set_addr_with_acd */
#include "lwip/prot/acd.h"
typedef void (*acd_conflict_callback_t)(struct netif *netif, acd_callback_enum_t state);
#endif
```

2. **Added Field to struct netif** (around line 401-406):

```
#if LWIP_ACD && LWIP_ACD_RFC5227_COMPLIANT_STATIC
  struct netif_pending_ip_config *pending_ip_config;
#endif /* LWIP_ACD && LWIP_ACD_RFC5227_COMPLIANT_STATIC */
```

3. **Added Function Declaration** (around line 458-463):

```
#if LWIP_ACD && LWIP_ACD_RFC5227_COMPLIANT_STATIC
err_t netif_set_addr_with_acd(struct netif *netif,
                              const ip4_addr_t *ipaddr,
                              const ip4_addr_t *netmask,
                              const ip4_addr_t *gw,
                              acd_conflict_callback_t callback);
#endif
```

File 4: `lwip/src/include/lwip/acd.h`

Changes Made:

1. **Added Forward Declaration for etharp_hdr** (around line 49-51):

```
/* Forward declare struct etharp_hdr to avoid circular dependency */
struct etharp_hdr;
```

2. **Added RFC 5227 Callback Declaration** (around line 58-59):

```
#if LWIP_ACD_RFC5227_COMPLIANT_STATIC
void acd_static_ip_rfc5227_callback(struct netif *netif, acd_callback_enum_t state);
#endif
```

3. **Removed Direct Include of etharp.h**: Replaced with forward declaration to break circular dependency.

File 5: `lwip/src/core/ipv4/acd.c`

Changes Made:

1. **Added Includes** (around line 73-78):

```
#include "lwip/etharp.h"  /* Need full definition of struct etharp_hdr for acd_arp_reply */
#if LWIP_ACD_RFC5227_COMPLIANT_STATIC
#include "lwip/netif_pending_ip.h"
#include "lwip/netif.h"
#include "lwip/mem.h"
#endif
```

2. **Fixed Self-Conflict Detection** (around line 410-414):

```
/* Fixed: Check MAC address to prevent self-conflict detection */
if ((ip4_addr_eq(&sipaddr, &acd->ipaddr) &&
     !eth_addr_eq(&netifaddr, &hdr->shwaddr)) ||
    (ip4_addr_isany_val(sipaddr) &&
     ip4_addr_eq(&dipaddr, &acd->ipaddr) &&
     !eth_addr_eq(&netifaddr, &hdr->shwaddr))) {
  LWIP_DEBUGF(ACD_DEBUG | LWIP_DBG_TRACE | LWIP_DBG_STATE | LWIP_DBG_LEVEL_WARNING,
              ("acd_arp_reply(): Probe Conflict detected\n"));
  acd_restart(netif, acd);
}
```

3. **Implemented RFC 5227 Callback Function** (around line 112-181):

```
#if LWIP_ACD_RFC5227_COMPLIANT_STATIC
/**
 * RFC 5227 compliant callback for static IP assignment
 * Assigns IP only after ACD confirms safety (ACD_IP_OK)
 * Removes IP if conflict detected
 */
void
acd_static_ip_rfc5227_callback(struct netif *netif, acd_callback_enum_t state)
{
  struct netif_pending_ip_config *pending = netif->pending_ip_config;

  if (pending == NULL) {
    return;
  }

  switch (state) {
    case ACD_IP_OK:
      /* RFC 5227 Section 2.1: Address confirmed safe, now assign it */
      LWIP_DEBUGF(ACD_DEBUG | LWIP_DBG_TRACE | LWIP_DBG_STATE,
                  ("acd_static_ip_rfc5227_callback(): Address confirmed safe, assigning
IP\n"));

      /* Assign IP address */
      netif_do_set_ipaddr(netif, &pending->ipaddr, NULL);
      netif_do_set_netmask(netif, &pending->netmask, NULL);
      netif_do_set_gw(netif, &pending->gw, NULL);

      pending->pending = 0;  /* Mark as assigned */

      /* Notify user callback */
      if (pending->user_callback) {
        pending->user_callback(netif, ACD_IP_OK);
      }
      break;

    case ACD_DECLINE:
    case ACD_RESTART_CLIENT:
      /* RFC 5227 Section 2.4.1: Conflict detected, MUST NOT use address */
      LWIP_DEBUGF(ACD_DEBUG | LWIP_DBG_TRACE | LWIP_DBG_STATE | LWIP_DBG_LEVEL_WARNING,
```

```
                ("acd_static_ip_rfc5227_callback(): Conflict detected, NOT assigning
IP\n"));

      /* Clean up pending configuration */
      acd_remove(netif, &pending->acd);
      pending->pending = 0;

      /* Notify user callback */
      if (pending->user_callback) {
        pending->user_callback(netif, state);
      }

      /* Free pending config */
      mem_free(pending);
      netif->pending_ip_config = NULL;
      break;

    default:
      break;
  }
}
#endif /* LWIP_ACD_RFC5227_COMPLIANT_STATIC */
```

4. **Modified Conflict Handling** (in `acd_handle_arp_conflict()`):

```
static void
acd_handle_arp_conflict(struct netif *netif, struct acd *acd)
{
#if LWIP_ACD_RFC5227_COMPLIANT_STATIC
  /* RFC 5227 Section 2.4.1: If conflict detected, MUST stop using address */
  if (netif->pending_ip_config != NULL &&
      ip4_addr_eq(&acd->ipaddr, &netif->pending_ip_config->ipaddr)) {
    /* This is a static IP with RFC 5227 compliance */
    LWIP_DEBUGF(ACD_DEBUG | LWIP_DBG_TRACE | LWIP_DBG_STATE | LWIP_DBG_LEVEL_WARNING,
                ("acd_handle_arp_conflict(): RFC 5227 - Removing conflicting static IP\n"));

    /* Remove IP address */
    netif_set_addr(netif, IP4_ADDR_ANY4, IP4_ADDR_ANY4, IP4_ADDR_ANY4);

    /* Notify user callback */
    if (netif->pending_ip_config->user_callback) {
      netif->pending_ip_config->user_callback(netif, ACD_DECLINE);
    }

    /* Clean up */
    acd_remove(netif, &netif->pending_ip_config->acd);
    mem_free(netif->pending_ip_config);
    netif->pending_ip_config = NULL;
    return;
  }
#endif /* LWIP_ACD_RFC5227_COMPLIANT_STATIC */

  /* ... existing code for other cases ... */
```

```
}
```

File 6: `lwip/src/core/netif.c`

Changes Made:

1. **Added Includes** (around line 70-75):

```
#if LWIP_ACD
#include "lwip/acd.h"
#if LWIP_ACD_RFC5227_COMPLIANT_STATIC
#include "lwip/netif_pending_ip.h"
#endif
#endif /* LWIP_ACD */
```

2. **Implemented** `netif_set_addr_with_acd()` **Function** (around line 500-600):

```
#if LWIP_ACD && LWIP_ACD_RFC5227_COMPLIANT_STATIC
/**
 * Set static IP address with RFC 5227 compliant conflict detection
 *
 * IP address is NOT assigned until ACD confirms it's safe (ACD_IP_OK callback).
 * If conflict is detected, IP is not assigned and callback is notified.
 *
 * @param netif Network interface
 * @param ipaddr IP address to assign (after ACD confirms)
 * @param netmask Netmask
 * @param gw Gateway
 * @param callback Callback for ACD events (required)
 * @return ERR_OK if ACD started successfully, ERR_ARG if invalid params
 */
err_t
netif_set_addr_with_acd(struct netif *netif,
                        const ip4_addr_t *ipaddr,
                        const ip4_addr_t *netmask,
                        const ip4_addr_t *gw,
                        acd_conflict_callback_t callback)
{
  struct netif_pending_ip_config *pending;

  LWIP_ASSERT_CORE_LOCKED();
  LWIP_ERROR("netif != NULL", netif != NULL, return ERR_ARG);
  LWIP_ERROR("ipaddr != NULL", ipaddr != NULL, return ERR_ARG);
  LWIP_ERROR("netmask != NULL", netmask != NULL, return ERR_ARG);
  LWIP_ERROR("gw != NULL", gw != NULL, return ERR_ARG);

  /* Don't allow if already pending */
  if (netif->pending_ip_config != NULL) {
    LWIP_DEBUGF(NETIF_DEBUG | LWIP_DBG_TRACE,
                ("netif_set_addr_with_acd(): Already have pending IP config\n"));
    return ERR_INPROGRESS;
```

```
  }

  /* Allocate pending configuration */
  pending = (struct netif_pending_ip_config *)mem_malloc(sizeof(struct
netif_pending_ip_config));
  if (pending == NULL) {
    return ERR_MEM;
  }

  /* Store configuration */
  ip4_addr_copy(pending->ipaddr, *ipaddr);
  ip4_addr_copy(pending->netmask, *netmask);
  ip4_addr_copy(pending->gw, *gw);
  pending->user_callback = callback;
  pending->pending = 1;

  netif->pending_ip_config = pending;

  /* Start ACD BEFORE assigning IP (RFC 5227 compliant) */
  if (acd_add(netif, &pending->acd, acd_static_ip_rfc5227_callback) == ERR_OK) {
    if (acd_start(netif, &pending->acd, *ipaddr) == ERR_OK) {
      LWIP_DEBUGF(NETIF_DEBUG | LWIP_DBG_TRACE | LWIP_DBG_STATE,
                  ("netif_set_addr_with_acd(): ACD started, IP assignment deferred\n"));
      return ERR_OK;
    }
  }

  /* Failed to start ACD, clean up */
  mem_free(pending);
  netif->pending_ip_config = NULL;
  return ERR_VAL;
}
#endif /* LWIP_ACD && LWIP_ACD_RFC5227_COMPLIANT_STATIC */
```

3. **Added Cleanup in** `netif_remove()` (around line 200-400):

```
#if LWIP_ACD && LWIP_ACD_RFC5227_COMPLIANT_STATIC
  /* Clean up pending IP configuration if present */
  if (netif->pending_ip_config != NULL) {
    acd_remove(netif, &netif->pending_ip_config->acd);
    mem_free(netif->pending_ip_config);
    netif->pending_ip_config = NULL;
  }
#endif
```

File 7: `lwip/src/include/lwip/prot/acd.h`

Changes Made: Made ACD Timing Constants Configurable

All timing constants wrapped with `#ifndef` guards to allow override in `lwipopts.h`:

```
#ifndef PROBE_WAIT
#define PROBE_WAIT           1   /* second  (initial random delay) */
#endif

#ifndef PROBE_MIN
#define PROBE_MIN            1   /* second  (minimum delay till repeated probe) */
#endif

#ifndef PROBE_MAX
#define PROBE_MAX            2   /* seconds (maximum delay till repeated probe) */
#endif

#ifndef PROBE_NUM
#define PROBE_NUM            3   /*         (number of probe packets) */
#endif

#ifndef ANNOUNCE_NUM
#define ANNOUNCE_NUM         2   /*         (number of announcement packets) */
#endif

#ifndef ANNOUNCE_INTERVAL
#define ANNOUNCE_INTERVAL    2   /* seconds (time between announcement packets) */
#endif

#ifndef ANNOUNCE_WAIT
#define ANNOUNCE_WAIT        2   /* seconds (delay before announcing) */
#endif

#ifndef DEFEND_INTERVAL
#define DEFEND_INTERVAL     10   /* seconds (minimum interval between defensive ARPs) */
#endif
```

Rationale: Enables EtherNet/IP and other protocols to override RFC 5227 default timings as needed.

File 8: `port/include/lwipopts.h`

Changes Made: Added Documentation for ACD Timing Override

```
/**
 * Note: ACD timing constants (PROBE_WAIT, PROBE_MIN, PROBE_MAX, PROBE_NUM,
 * ANNOUNCE_NUM, ANNOUNCE_INTERVAL, ANNOUNCE_WAIT, DEFEND_INTERVAL) can be
 * overridden in lwipopts.h for protocol-specific requirements. For example,
 * EtherNet/IP uses different timings than RFC 5227. Define these constants
 * before including this file to override the defaults.
 */
```

\newpage

Configuration Changes

Build Configuration (`CMakeLists.txt`)

File: `CMakeLists.txt` (project root)

Change: Added FD_SETSIZE Definition

```
# Define FD_SETSIZE for LWIP_MAX_SOCKETS=64
# LWIP_SOCKET_OFFSET = FD_SETSIZE - MAX_SOCKETS, and we need LWIP_SOCKET_OFFSET >= 6
# So FD_SETSIZE >= MAX_SOCKETS + 6 = 64 + 6 = 70
# Also need room for stdout, stderr, stdin (+3), so use 73 for safety
add_compile_definitions(FD_SETSIZE=73)
```

Rationale: Required when `LWIP_MAX_SOCKETS > 61`. Ensures sufficient file descriptor space for sockets and console I/O.

ESP-IDF Configuration (`sdkconfig`)

Performance Optimizations

Configuration	Default	Modified	Rationale
`CONFIG_LWIP_MAX_SOCKETS`	32	**64**	Increased for EtherNet/IP multiple connections
`CONFIG_LWIP_MAX_ACTIVE_TCP`	32	**64**	Support more concurrent TCP connections
`CONFIG_LWIP_MAX_LISTENING_TCP`	16	**32**	More listening sockets for EtherNet/IP services
`CONFIG_LWIP_MAX_UDP_PCBS`	64	**128**	Increased UDP support for EtherNet/IP messaging
`CONFIG_LWIP_TCP_SND_BUF_DEFAULT`	16384	**32768**	Larger send buffer for better throughput
`CONFIG_LWIP_TCP_WND_DEFAULT`	16384	**32768**	Larger receive window for better performance
`CONFIG_LWIP_TCPIP_RECVMBOX_SIZE`	32	**64**	Larger TCP/IP task mailbox
`CONFIG_LWIP_TCP_RECVMBOX_SIZE`	32	**64**	TCP receive mailbox for better concurrency
`CONFIG_LWIP_TCP_ACCEPTMBOX_SIZE`	6	**16**	More pending TCP accept connections
`CONFIG_LWIP_TCP_OOSEQ_MAX_PBUFS`	4	**12**	More out-of-sequence TCP packet buffers

Configuration	Default	Modified	Rationale
`CONFIG_LWIP_TCPIP_TASK_STACK_SIZE`	3072 → 6144	**8192**	Increased stack for OpENer and ACD processing

Task Affinity Configuration

Configuration	Default	Modified	Rationale
`CONFIG_LWIP_TCPIP_TASK_AFFINITY_NO_AFFINITY`	y	**n**	Disabled no-affinity mode
`CONFIG_LWIP_TCPIP_TASK_AFFINITY_CPU0`	n	**y**	Pin LWIP TCP/IP task to Core 0
`CONFIG_LWIP_TCPIP_TASK_AFFINITY`	0x7FFFFFFF	**0x0**	Explicit Core 0 affinity

Rationale: Pins LWIP TCP/IP task to Core 0, leaving Core 1 available for other hardware interfaces and OpENer processing.

IRAM Optimization

Configuration	Default	Modified	Rationale
`CONFIG_LWIP_IRAM_OPTIMIZATION`	n	**y**	Places RX/TX functions in IRAM (~10KB)
`CONFIG_LWIP_EXTRA_IRAM_OPTIMIZATION`	n	**y**	Places TCP functions in IRAM (~17KB)

Total IRAM Usage: ~27KB for LWIP optimizations

Rationale: Improves network performance by avoiding flash cache misses, especially important for real-time EtherNet/IP communication.

ACD and DHCP Configuration

Configuration	Default	Modified	Rationale
`CONFIG_LWIP_DHCP_DOES_ACD_CHECK`	n	**y**	Enable ACD checking for DHCP-assigned addresses
`CONFIG_LWIP_AUTOIP`	y	**y**	Required for ACD support
`CONFIG_LWIP_AUTOIP_MAX_CONFLICTS`	10	**9**	Slightly reduced for faster conflict detection

OpENer Task Affinity

File: `components/opener/src/ports/ESP32/opener.c`

Change: Modified task creation to use `xTaskCreatePinnedToCore()`:

```
BaseType_t result = xTaskCreatePinnedToCore(opener_thread,
                                            "OpENer",
                                            OPENER_STACK_SIZE,
                                            netif,
                                            OPENER_THREAD_PRIO,
                                            &opener_task_handle,
                                            0);  // Core 0
```

Rationale:

- Keeps EtherNet/IP stack and OpENer on same core
- Reduces context switching overhead
- Improves cache locality for network operations

\newpage

ODVA Compliance and Timing

ODVA Reference

ODVA Publication 28 - Recommended IP Addressing Methods for EtherNet/IP™ Devices

- **Purpose**: Best practices for IP address configuration in EtherNet/IP devices
- **Content**: Recommends Address Conflict Detection (ACD) for static IP assignment per RFC 5227
- **Direct Link**: ODVA Document Library (Search for "Pub 28" or "IP Addressing Methods")
- **Relevance**: ODVA recommends ACD for static IP assignment but emphasizes faster timing for real-time industrial applications

ACD Timing Configuration

This project implements **adjustable ACD timings** to comply with ODVA recommendations for EtherNet/IP devices. The timings are optimized for real-time industrial communication requirements.

Current Configuration (ODVA-Optimized)

The project uses **custom ACD timings optimized for EtherNet/IP** via Kconfig options (`OPENER_ACD_CUSTOM_TIMING`). These values are significantly faster than RFC 5227 defaults to meet EtherNet/IP's real-time requirements:

Parameter	RFC 5227 Default	ODVA-Optimized (Current)	Description
PROBE_WAIT	1000 ms	**0 ms**	Initial random delay before first probe
PROBE_MIN	1000 ms	**20 ms**	Minimum delay between probe packets

Parameter	RFC 5227 Default	ODVA-Optimized (Current)	Description
PROBE_MAX	2000 ms	**20 ms**	Maximum delay between probe packets
PROBE_NUM	3 packets	**1 packet**	Number of probe packets to send
ANNOUNCE_WAIT	2000 ms	**20 ms**	Delay before first announcement
ANNOUNCE_NUM	2 packets	**1 packet**	Number of announcement packets
ANNOUNCE_INTERVAL	2000 ms	**20 ms**	Time between announcement packets
DEFEND_INTERVAL	10000 ms	**10000 ms**	Minimum interval between defensive ARPs

Total ACD time (ODVA-optimized): Approximately **60-100 ms** (vs. 7-10 seconds for RFC 5227 defaults)

Configuration Method

ACD timings are configurable via ESP-IDF menuconfig:

- Navigate to: `Component config > OpenER ACD Timing`
- Enable `Override default RFC5227 timings` to use ODVA-optimized values
- Adjust individual timing parameters as needed for your network environment

Kconfig Options (`main/Kconfig.projbuild`):

```
menu "OpenER ACD Timing"
    config OPENER_ACD_CUSTOM_TIMING
        bool "Override default RFC5227 timings"
        default y
        help
            Enable this option to tune IPv4 Address Conflict Detection timings to match
            EtherNet/IP requirements. When disabled, lwIP falls back to RFC5227 defaults.

    if OPENER_ACD_CUSTOM_TIMING
        config OPENER_ACD_PROBE_WAIT_MS
            int "Initial random probe wait (ms)"
            default 1000
            help
                Maximum initial delay before the first probe is sent. Set to 0 for no delay.

        config OPENER_ACD_PROBE_MIN_MS
            int "Minimum delay between probes (ms)"
            default 1000
```

```
        config OPENER_ACD_PROBE_MAX_MS
            int "Maximum delay between probes (ms)"
            default 2000

        config OPENER_ACD_PROBE_NUM
            int "Number of probes"
            default 3

        config OPENER_ACD_ANNOUNCE_NUM
            int "Number of announcements"
            default 2

        config OPENER_ACD_ANNOUNCE_INTERVAL_MS
            int "Interval between announcements (ms)"
            default 2000

        config OPENER_ACD_ANNOUNCE_WAIT_MS
            int "Delay before first announcement (ms)"
            default 2000
    endif
endmenu
```

Current Values (`sdkconfig.defaults`):

```
CONFIG_OPENER_ACD_CUSTOM_TIMING=y
CONFIG_OPENER_ACD_PROBE_WAIT_MS=0
CONFIG_OPENER_ACD_PROBE_MIN_MS=20
CONFIG_OPENER_ACD_PROBE_MAX_MS=20
CONFIG_OPENER_ACD_PROBE_NUM=1
CONFIG_OPENER_ACD_ANNOUNCE_NUM=1
CONFIG_OPENER_ACD_ANNOUNCE_INTERVAL_MS=20
CONFIG_OPENER_ACD_ANNOUNCE_WAIT_MS=20
```

Note: The optimized timings balance conflict detection reliability with EtherNet/IP's real-time communication requirements. For networks with high latency or packet loss, you may need to increase these values.

\newpage

Testing Guide

Prerequisites

Hardware Requirements

- ESP32-P4 development board with Ethernet connection
- Ethernet cable connected to a network switch/hub
- A second device (PC, another ESP32, or network device) capable of using the test IP address
- Serial console access (USB-UART) for monitoring logs

Software Requirements

- ESP-IDF v5.5.1 or compatible
- EtherNet/IP client tool (e.g., RSLinx, Wireshark, or custom CIP client)
- Serial terminal program (e.g., PuTTY, minicom, or ESP-IDF monitor)

Build Configuration

Ensure the following are enabled in `sdkconfig`:

- `CONFIG_LWIP_AUTOIP=y` (enables ACD support)
- `CONFIG_LWIP_DHCP_DOES_ACD_CHECK=y` (for DHCP ACD checking)
- `LWIP_ACD_RFC5227_COMPLIANT_STATIC=1` (default, enables RFC 5227 compliance)
- `CONFIG_OPENER_ACD_CUSTOM_TIMING=y` (enables ODVA-optimized ACD timings)

Test Scenario 1: ACD Enabled - No Conflict (Success Case)

Objective: Verify that ACD successfully assigns IP when no conflict exists.

Steps:

1. Ensure no device on the network uses IP `192.168.1.100`
2. Configure ESP32 with static IP `192.168.1.100` and ACD enabled (`select_acd=1`)
3. Power cycle or reset the ESP32
4. Monitor serial output for ACD progress

Expected Behavior:

- ACD probe sequence runs (1 probe packet with ODVA timings)
- No conflicts detected
- IP address is assigned after `ACD_IP_OK` callback (~60-100 ms)
- Device becomes operational with assigned IP

Expected Log Sequence:

```
I (xxxx) opener_main: After NV load select_acd=1
I (xxxx) opener_main: ACD path: select_acd=1, RFC5227=1, lwip_netif=0x...
I (xxxx) opener_main: Using RFC 5227 compliant ACD for static IP
I (xxxx) opener_main: RFC 5227: ACD started, IP assignment deferred
I (xxxx) opener_main: ACD client registered
I (xxxx) opener_main: Starting ACD probe via acd_start()
I (xxxx) opener_main: ACD callback state=0
I (xxxx) opener_main: RFC 5227: IP assigned after ACD confirmation
```

Verification:

- Check TCP/IP Interface Object Attribute #1 (`status`): Should show IP configured
- Ping the device: `ping 192.168.1.100` should succeed

- Check Attribute #11 (`last_acd_activity`): Should be `0` (no conflict)

Test Scenario 2: ACD Enabled - Conflict Detected

Objective: Verify that ACD detects conflicts and prevents IP assignment.

Steps:

1. Configure another device (PC or second ESP32) with IP `192.168.1.100`
2. Ensure the conflicting device is active and responds to ARP
3. Configure ESP32 with static IP `192.168.1.100` and ACD enabled
4. Power cycle or reset the ESP32
5. Monitor serial output

Expected Behavior:

- ACD probe sequence runs
- Conflict detected during probe phase
- IP address is **NOT assigned** (RFC 5227 compliant)
- Device reports ACD fault status

Expected Log Sequence:

```
I (xxxx) opener_main: After NV load select_acd=1
I (xxxx) opener_main: ACD path: select_acd=1, RFC5227=1, lwip_netif=0x...
I (xxxx) opener_main: Using RFC 5227 compliant ACD for static IP
I (xxxx) opener_main: RFC 5227: ACD started, IP assignment deferred
I (xxxx) opener_main: ACD client registered
I (xxxx) opener_main: Starting ACD probe via acd_start()
I (xxxx) opener_main: ACD callback state=1  (or state=2 for DECLINE)
W (xxxx) opener_main: RFC 5227: IP not assigned due to conflict
```

Verification:

- Check TCP/IP Interface Object Attribute #1 (`status`):
    - Bit `kTcpipStatusAcdStatus` should be set
    - Bit `kTcpipStatusAcdFault` should be set
- Check Attribute #11 (`last_acd_activity`): Should be `3` (conflict detected)
- Ping the device: Should fail (IP not assigned)
- Device should not respond to EtherNet/IP connections

Test Scenario 3: ACD Disabled - Immediate Assignment

Objective: Verify that disabling ACD allows immediate IP assignment.

Steps:

1. Configure ESP32 with static IP `192.168.1.100` and ACD disabled (`select_acd=0`)
2. Power cycle or reset the ESP32
3. Monitor serial output

Expected Behavior:

- No ACD probe sequence
- IP address assigned immediately
- Device becomes operational quickly

Expected Log Sequence:

```
I (xxxx) opener_main: After NV load select_acd=0
I (xxxx) opener_main: ACD disabled - setting static IP immediately
```

Verification:

- Check TCP/IP Interface Object Attribute #1 (`status`): IP should be configured
- Ping the device: Should succeed immediately
- No ACD-related log messages

Test Scenario 4: Runtime ACD Enable/Disable

Objective: Verify that ACD can be enabled/disabled at runtime via CIP.

Steps:

1. Start device with ACD disabled (`select_acd=0`)
2. Verify IP is assigned
3. Use CIP SetAttributeSingle to set Attribute #10 (`select_acd`) to `1`
4. Change static IP address via CIP
5. Monitor for ACD behavior

Expected Behavior:

- When ACD enabled and IP changed: ACD probe runs before new IP assignment
- Configuration persists to NVS

CIP Request Example (SetAttributeSingle):

```
Service: SetAttributeSingle (0x10)
Class: 0xF5 (TCP/IP Interface)
Instance: 1
Attribute: 10 (select_acd)
Data: 0x01 (enable ACD)
```

Verification Methods

Method 1: Serial Log Analysis

Monitor serial output for ACD-related log messages. Key tags:

- `opener_main`: Main application ACD logic
- `NvTcpip`: NVS load/store operations
- `ACD_DEBUG`: LWIP ACD debug messages (if enabled)

Method 2: CIP Attribute Reading

Use EtherNet/IP client to read TCP/IP Interface Object attributes:

Attribute #1 (`status` - DWORD):

- Bit 0: IP configured
- Bit 1: ACD status
- Bit 2: ACD fault

Attribute #10 (`select_acd` - BOOL):

- `0` = ACD disabled
- `1` = ACD enabled

Attribute #11 (`last_acd_activity` - BOOL):

- `0` = No conflict
- `1` = Conflict detected (restart)
- `2` = ACD in progress
- `3` = Conflict detected (decline)

Method 3: Network Tools

ARP Monitoring:

```
# Linux/Mac
sudo tcpdump -i eth0 arp

# Windows
arp -a
```

Ping Test:

```
ping 192.168.1.100
```

Wireshark Capture:

- Filter: `arp`
- Look for ARP probe packets (sender IP = 0.0.0.0, target IP = test IP)

\newpage

Patch Application

Patch File Information

- **Patch File**: `dependency_modifications/lwIP/lwip-rfc5227-patch.patch`
- **Target**: ESP-IDF v5.5.1 LWIP component
- **Base Path**: `components/lwip/`
- **Format**: Unified diff format
- **Files Modified**: 7 files
- **Files Created**: 1 new file

How to Apply

Prerequisites

- ESP-IDF v5.5.1 installed
- Clean LWIP source (no previous modifications)

Application Steps

1. **Navigate to ESP-IDF components directory:**

   ```
   cd $IDF_PATH/components/lwip
   ```

   On Windows (PowerShell):

   ```
   cd $env:IDF_PATH\components\lwip
   ```

2. **Apply the patch:**

   ```
   patch -p1 < /path/to/lwip-rfc5227-patch.patch
   ```

   On Windows (PowerShell):

   ```
   git apply --ignore-whitespace lwip-rfc5227-patch.patch
   ```

3. **Verify the patch applied correctly:**

- Check that `netif_pending_ip.h` exists
- Verify modified files contain the expected changes
- Build your project to ensure no compilation errors

Troubleshooting

If the patch fails to apply:

1. **Check ESP-IDF version:**

   ```
   cd $IDF_PATH
   git describe --tags
   ```

   Should show v5.5.1

2. **Verify clean source:**

   ```
   cd $IDF_PATH/components/lwip
   git status
   ```

   Should show no modifications (or only expected ones)

3. **Manual application:**
   If automatic patching fails, you can manually apply changes by following the implementation guide in this document.

4. **Partial application:**
   If some hunks fail, you can use `patch -p1 --dry-run` to see what would be applied, then manually fix conflicts.

Reverting the Patch

To revert the patch:

```
cd $IDF_PATH/components/lwip
patch -p1 -R < /path/to/lwip-rfc5227-patch.patch
```

Or restore from git:

```
cd $IDF_PATH/components/lwip
git checkout -- .
```

\newpage

References

Standards and Specifications

1. **RFC 5227 - IPv4 Address Conflict Detection**
   - **Purpose**: Standard for detecting IPv4 address conflicts before assignment
   - **Direct Link**: https://tools.ietf.org/html/rfc5227
   - **Relevance**: **CRITICAL** - Directly relevant to our static IP ACD implementation
2. **ODVA Publication 28 - Recommended IP Addressing Methods for EtherNet/IP™ Devices**
   - **Purpose**: Best practices for IP address configuration in EtherNet/IP devices
   - **Direct Link**: ODVA Document Library (Search for "Pub 28")
   - **Relevance**: **CRITICAL** - ODVA recommendations for ACD in EtherNet/IP devices

Project Documentation

1. **ACD Testing Guide**: `ReadmeACD.md` - Detailed testing instructions and usage examples
2. **LWIP Modifications**: `LWIP_MODIFICATIONS.md` - Complete documentation of all LWIP changes
3. **RFC 5227 Implementation Guide**: `dependency_modifications/lwIP/RFC5227_IMPLEMENTATION_GUIDE.md` - Step-by-step implementation guide
4. **ACD Static IP Issue**: `dependency_modifications/lwIP/acd-static-ip-issue.md` - Original issue description
5. **RFC 5227 Requirements**: `dependency_modifications/Analysis_of_ACD_Issue/RFC5227_COMPLIANCE_REQUIREMENTS.md` - RFC 5227 requirements analysis
6. **ACD Interface Analysis**: `dependency_modifications/Analysis_of_ACD_Issue/ACD_INTERFACE_BRINGUP_ANALYSIS.md` - Detailed process analysis
7. **DHCP vs Static IP Comparison**: `dependency_modifications/Analysis_of_ACD_Issue/DHCP_VS_STATIC_IP_ACD_COMPARISON.md` - Comparison analysis
8. **EtherNet/IP References**: `docs/EtherNetIP_References.md` - Comprehensive ODVA publication guide

Patch Files

1. **LWIP RFC 5227 Patch**: `dependency_modifications/lwIP/lwip-rfc5227-patch.patch` - Unified diff patch file
2. **Patch README**: `dependency_modifications/lwIP/lwip-rfc5227-patch-README.md` - Patch application instructions

Additional Resources

- **ODVA Website**: https://www.odva.org/
- **ODVA Technology Standards**: https://www.odva.org/technology-standards/
- **ODVA Document Library**: https://www.odva.org/technology-standards/document-library/
- **OpENer GitHub**: https://github.com/EIPStackGroup/OpENer

\newpage

Appendix A: Code Change Summary

Files Modified Summary

1. ✅ **NEW**: `lwip/src/include/lwip/netif_pending_ip.h`
2. ✅ **MODIFIED**: `lwip/src/include/lwip/opt.h` - Configuration option
3. ✅ **MODIFIED**: `lwip/src/include/lwip/netif.h` - Pending IP config field and function declaration
4. ✅ **MODIFIED**: `lwip/src/include/lwip/acd.h` - Forward declarations and callback
5. ✅ **MODIFIED**: `lwip/src/core/ipv4/acd.c` - RFC 5227 callback and conflict handling
6. ✅ **MODIFIED**: `lwip/src/core/netif.c` - New API function and cleanup
7. ✅ **MODIFIED**: `lwip/src/include/lwip/prot/acd.h` - Configurable timing constants
8. ✅ **MODIFIED**: `port/include/lwipopts.h` - Documentation for timing overrides

Configuration Summary

Performance Increases

- **Sockets**: 32 → 64 (+100%)
- **Active TCP**: 32 → 64 (+100%)
- **Listening TCP**: 16 → 32 (+100%)
- **UDP PCBs**: 64 → 128 (+100%)
- **TCP Send Buffer**: 16KB → 32KB (+100%)
- **TCP Window**: 16KB → 32KB (+100%)
- **TCP/IP Stack**: 3072 → 8192 bytes (+167%)

Mailbox Increases

- **TCP/IP Recv**: 32 → 64 (+100%)
- **TCP Recv**: 32 → 64 (+100%)
- **TCP Accept**: 6 → 16 (+167%)
- **TCP OOSEQ**: 4 → 12 (+200%)

New Features

- ✅ RFC 5227 compliant static IP assignment
- ✅ Configurable ACD timings (ODVA optimized)
- ✅ Task affinity control (Core 0)
- ✅ IRAM optimization enabled
- ✅ Self-conflict detection fix

Build Requirements

- **FD_SETSIZE**: Must be >= 70 when `LWIP_MAX_SOCKETS=64`
- **Current Setting**: `FD_SETSIZE=73` (defined in `CMakeLists.txt`)

Backward Compatibility

- ✅ All changes are backward compatible
- ✅ RFC 5227 mode can be disabled via `LWIP_ACD_RFC5227_COMPLIANT_STATIC=0`
- ✅ Legacy `netif_set_addr()` still works when RFC 5227 mode disabled
- ✅ ACD timing overrides are optional

\newpage

Appendix B: Maintenance Notes

Upgrading ESP-IDF

When upgrading ESP-IDF, these modifications will need to be reapplied:

1. **Source Code Changes**: All files in `components/lwip/` directory
2. **Configuration**: `sdkconfig` file (can be merged/updated)
3. **Build Config**: `CMakeLists.txt` (FD_SETSIZE definition)

Recommended Approach

1. Keep a patch file or script to reapply LWIP modifications
2. Document any ESP-IDF version-specific changes
3. Test thoroughly after ESP-IDF upgrades

Version Compatibility

- **ESP-IDF v5.5.1**: ✅ Fully tested and verified
- **Other Versions**: May require adjustments to line numbers and API compatibility

---

\newpage

Document Information

Document Title: Address Conflict Detection (ACD) Implementation - Complete Documentation
Project: ESP32-P4 OpENer EtherNet/IP Stack
ESP-IDF Version: v5.5.1
LWIP Component: ESP-IDF v5.5.1 LWIP Stack
Document Version: 1.0
Date: 2024
Author: Adam G. Sweeney

Creation Date: 11/11/2025

END OF DOCUMENT