Documentation Multicast IP et Implémentation C Embarqué

1. Principes du Multicast IP

1.1 Qu'est-ce que le Multicast?

Le multicast IP est un mode de transmission réseau permettant d'envoyer des données d'une source vers plusieurs destinataires simultanément, de manière efficace. Contrairement à l'unicast (un émetteur vers un récepteur) ou au broadcast (un émetteur vers tous), le multicast cible un groupe spécifique d'hôtes intéressés.

Avantages:
- Économie de bande passante (un seul flux pour N destinataires)
- Réduction de la charge CPU sur l'émetteur
- Idéal pour le streaming, la découverte de services, les mises à jour distribuées

1.2 Adresses Multicast

Plage IPv4: 224.0.0.0 à 239.255.255.255 (classe D)

- **224.0.0.0 - 224.0.0.255:** Réservé (usage local, jamais routé)
  - 224.0.0.1: Tous les hôtes du sous-réseau
  - 224.0.0.2: Tous les routeurs
  - 224.0.0.251: mDNS (Multicast DNS)
  - 224.0.0.252: LLMNR

- **239.0.0.0 - 239.255.255.255:** Usage administratif local (organization-local)

1.3 Mapping Adresse IP ↔ Adresse MAC

Une adresse multicast IP est mappée vers une adresse MAC multicast selon la formule:

```
MAC = 01:00:5E:0x:xx:xx
      └──────┘ └──────┘
     Préfixe  23 bits de poids faible de l'IP
```

Exemple:
```
IP:  224.1.2.3 = 0xE0.01.02.03
                 └─ 25 bits à ignorer
Derniers 23 bits: 0x01.02.03
```

MAC: 01:00:5E:01:02:03
```

**Problème:** Collision possible car on perd 5 bits (32 IPs différentes peuvent mapper vers la même MAC).

## 2. Protocoles Multicast

### 2.1 IGMP (Internet Group Management Protocol)

Protocole permettant aux hôtes de signaler aux routeurs leur appartenance à des groupes multicast.

**Versions:**
- IGMPv1: Join/Leave basique
- IGMPv2: Ajout de messages Leave explicites
- IGMPv3: Source-specific multicast (SSM)

**Messages principaux (IGMPv2):**
- **Membership Query (0x11):** Routeur demande qui écoute quoi
- **Membership Report (0x16):** Hôte déclare rejoindre un groupe
- **Leave Group (0x17):** Hôte quitte un groupe

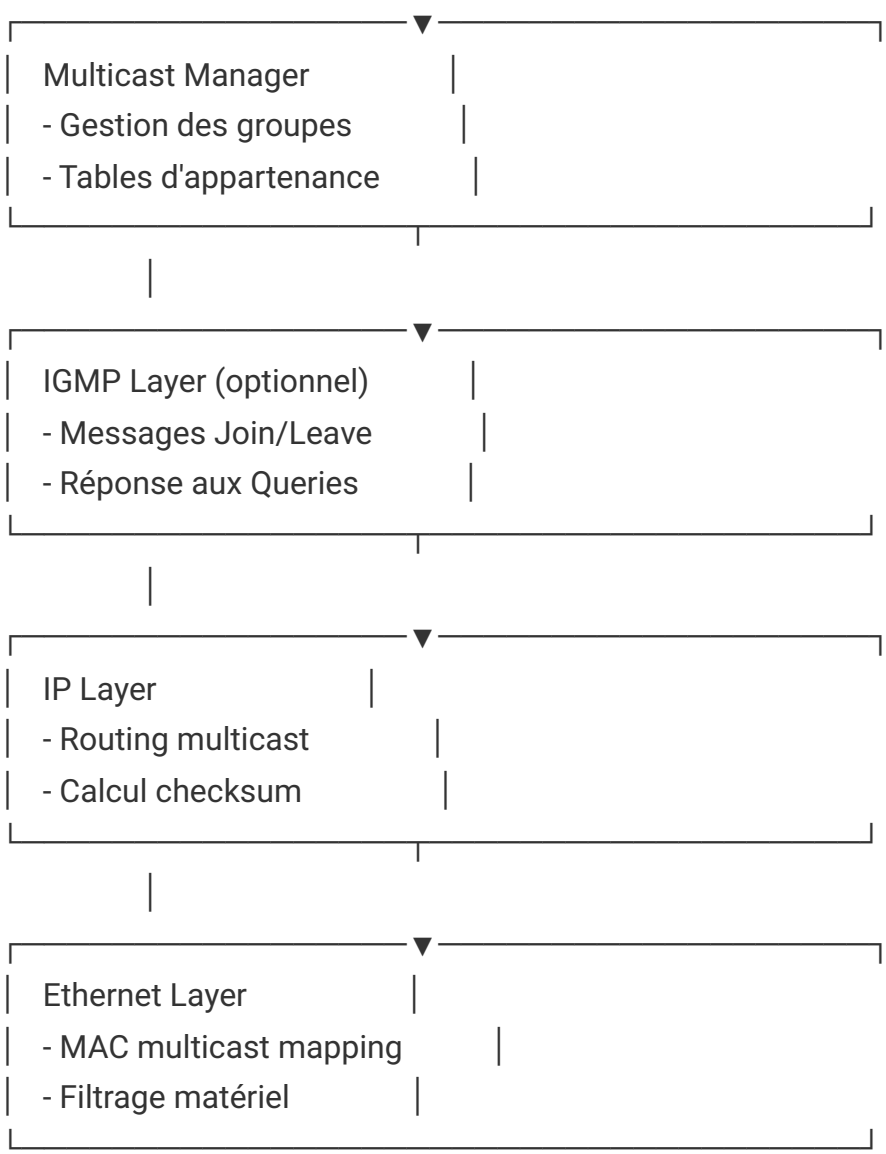
### 2.2 Structure d'un paquet IGMP

```
0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Type (8)     | Max Resp (8)  |      Checksum (16)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Group Address (32)                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3. Architecture d'une Stack Multicast Légère

3.1 Composants Principaux

```
┌─────────────────────────────────────────────────┐
│   Application (send/receive)        │
└─────────────────────────────────────────────────┘
                    │
                    │
```
```




```

### 3.2 Structures de Données

```c
// Entrée de groupe multicast
typedef struct mcast_group {
    uint32_t group_addr;      // Adresse IP du groupe
    uint8_t mac_addr[6];      // Adresse MAC correspondante
    uint16_t ref_count;       // Nombre d'abonnés locaux
    uint32_t last_query;      // Timestamp dernière query IGMP
    struct mcast_group *next;
} mcast_group_t;

// Gestionnaire multicast global
typedef struct {
    mcast_group_t *groups;    // Liste chaînée des groupes
    uint8_t igmp_enabled;     // Support IGMP actif
    uint32_t query_interval;  // Intervalle des queries (ms)
} mcast_manager_t;
```
```

```c
// Configuration interface réseau
typedef struct {
    uint32_t ip_addr;
    uint32_t netmask;
    uint8_t mac_addr[6];
    uint8_t ttl;            // TTL pour paquets multicast
    uint8_t loopback;       // Loopback multicast local
} net_config_t;
```

4. Implémentation C Minimale

4.1 Fonctions Utilitaires

```c
#include <stdint.h>
#include <string.h>

// Calcul du checksum IP/IGMP
uint16_t ip_checksum(const uint8_t *data, uint16_t len) {
    uint32_t sum = 0;

    while (len > 1) {
        sum += (data[0] << 8) | data[1];
        data += 2;
        len -= 2;
    }

    if (len > 0) {
        sum += data[0] << 8;
    }

    while (sum >> 16) {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }

    return ~sum;
}

// Conversion IP multicast → MAC multicast
void mcast_ip_to_mac(uint32_t ip, uint8_t *mac) {
    mac[0] = 0x01;
```

```
    mac[1] = 0x00;
    mac[2] = 0x5E;
    mac[3] = (ip >> 16) & 0x7F;  // Masque 23 bits
    mac[4] = (ip >> 8) & 0xFF;
    mac[5] = ip & 0xFF;
}

// Vérification si adresse est multicast
int is_multicast(uint32_t ip) {
    return (ip & 0xF0000000) == 0xE0000000;
}
```

4.2 Gestion des Groupes

```c
static mcast_manager_t mcast_mgr = {0};

// Initialisation
void mcast_init(void) {
    mcast_mgr.groups = NULL;
    mcast_mgr.igmp_enabled = 1;
    mcast_mgr.query_interval = 125000; // 125 secondes
}

// Rejoindre un groupe
int mcast_join_group(uint32_t group_addr) {
    if (!is_multicast(group_addr)) {
        return -1;
    }

    // Recherche si groupe existe déjà
    mcast_group_t *grp = mcast_mgr.groups;
    while (grp) {
        if (grp->group_addr == group_addr) {
            grp->ref_count++;
            return 0;
        }
        grp = grp->next;
    }

    // Nouveau groupe
    grp = (mcast_group_t*)malloc(sizeof(mcast_group_t));
```

```
    if (!grp) return -1;

    grp->group_addr = group_addr;
    mcast_ip_to_mac(group_addr, grp->mac_addr);
    grp->ref_count = 1;
    grp->last_query = 0;
    grp->next = mcast_mgr.groups;
    mcast_mgr.groups = grp;

    // Configuration matérielle (filtre MAC)
    eth_add_multicast_filter(grp->mac_addr);

    // Envoi IGMP Membership Report si activé
    if (mcast_mgr.igmp_enabled) {
        igmp_send_report(group_addr);
    }

    return 0;
}

// Quitter un groupe
int mcast_leave_group(uint32_t group_addr) {
    mcast_group_t **grp_ptr = &mcast_mgr.groups;

    while (*grp_ptr) {
        mcast_group_t *grp = *grp_ptr;
        if (grp->group_addr == group_addr) {
            grp->ref_count--;

            if (grp->ref_count == 0) {
                // Retrait du filtre MAC
                eth_remove_multicast_filter(grp->mac_addr);

                // Envoi IGMP Leave
                if (mcast_mgr.igmp_enabled) {
                    igmp_send_leave(group_addr);
                }

                // Suppression de la liste
                *grp_ptr = grp->next;
                free(grp);
            }
            return 0;
```

```
        }
        grp_ptr = &(*grp_ptr)->next;
    }

    return -1;
}

// Vérification appartenance à un groupe
int mcast_is_member(uint32_t group_addr) {
    mcast_group_t *grp = mcast_mgr.groups;
    while (grp) {
        if (grp->group_addr == group_addr) {
            return 1;
        }
        grp = grp->next;
    }
    return 0;
}
```

4.3 Couche IGMP

```c
#define IGMP_QUERY              0x11
#define IGMP_V2_REPORT          0x16
#define IGMP_LEAVE_GROUP        0x17
#define IGMP_V3_REPORT          0x22

#define IGMP_ALL_HOSTS          0xE0000001  // 224.0.0.1
#define IGMP_ALL_ROUTERS        0xE0000002  // 224.0.0.2

// Structure paquet IGMP
typedef struct {
    uint8_t type;
    uint8_t max_resp_time;
    uint16_t checksum;
    uint32_t group_addr;
} __attribute__((packed)) igmp_packet_t;

// Envoi d'un Membership Report
void igmp_send_report(uint32_t group_addr) {
    igmp_packet_t pkt;
```

```
    pkt.type = IGMP_V2_REPORT;
    pkt.max_resp_time = 0;
    pkt.group_addr = htonl(group_addr);
    pkt.checksum = 0;
    pkt.checksum = ip_checksum((uint8_t*)&pkt, sizeof(pkt));

    // Envoi vers le groupe lui-même (particularité IGMP)
    ip_send_packet(group_addr, IPPROTO_IGMP,
                   (uint8_t*)&pkt, sizeof(pkt), 1); // TTL=1
}

// Envoi d'un Leave Group
void igmp_send_leave(uint32_t group_addr) {
    igmp_packet_t pkt;

    pkt.type = IGMP_LEAVE_GROUP;
    pkt.max_resp_time = 0;
    pkt.group_addr = htonl(group_addr);
    pkt.checksum = 0;
    pkt.checksum = ip_checksum((uint8_t*)&pkt, sizeof(pkt));

    // Envoi vers tous les routeurs
    ip_send_packet(IGMP_ALL_ROUTERS, IPPROTO_IGMP,
                   (uint8_t*)&pkt, sizeof(pkt), 1);
}

// Réception d'un paquet IGMP
void igmp_receive(const uint8_t *data, uint16_t len) {
    if (len < sizeof(igmp_packet_t)) return;

    igmp_packet_t *pkt = (igmp_packet_t*)data;

    switch (pkt->type) {
        case IGMP_QUERY: {
            uint32_t group = ntohl(pkt->group_addr);

            // General Query (group = 0.0.0.0)
            if (group == 0) {
                // Répondre pour tous nos groupes
                mcast_group_t *grp = mcast_mgr.groups;
                while (grp) {
                    // Délai aléatoire avant réponse
                    uint32_t delay = (rand() % pkt->max_resp_time) * 100;
```

```
                schedule_report(grp->group_addr, delay);
                grp = grp->next;
            }
        }
        // Group-Specific Query
        else if (mcast_is_member(group)) {
            uint32_t delay = (rand() % pkt->max_resp_time) * 100;
            schedule_report(group, delay);
        }
        break;
    }

    case IGMP_V2_REPORT:
        // Autre hôte a répondu, on peut supprimer notre timer
        // (Report Suppression)
        cancel_report(ntohl(pkt->group_addr));
        break;
  }
}
```

4.4 Envoi/Réception Multicast

```c
// Envoi d'un paquet multicast
int mcast_send(uint32_t dst_group, uint16_t dst_port,
               const uint8_t *data, uint16_t len) {
  if (!is_multicast(dst_group)) {
    return -1;
  }

  // Construction paquet UDP
  udp_packet_t *udp_pkt = build_udp_packet(
    get_local_ip(),
    dst_group,
    get_ephemeral_port(),
    dst_port,
    data,
    len
  );

  // Envoi avec TTL multicast (défaut: 1 pour local)
  ip_send_packet(dst_group, IPPROTO_UDP,
```

```
            (uint8_t*)udp_pkt,
            sizeof(udp_header_t) + len,
            1);  // TTL

    free(udp_pkt);
    return 0;
}

// Réception d'un paquet multicast
void mcast_receive_handler(const uint8_t *frame, uint16_t len) {
    // Extraction de l'en-tête Ethernet
    eth_header_t *eth = (eth_header_t*)frame;

    // Vérification MAC destination multicast
    if ((eth->dst_mac[0] & 0x01) == 0) {
        return;  // Pas multicast
    }

    // Extraction IP
    ip_header_t *ip = (ip_header_t*)(frame + sizeof(eth_header_t));
    uint32_t dst_ip = ntohl(ip->dst_addr);

    // Vérification appartenance au groupe
    if (!mcast_is_member(dst_ip)) {
        return;  // On n'écoute pas ce groupe
    }

    // Traitement selon protocole
    switch (ip->protocol) {
        case IPPROTO_IGMP:
            igmp_receive(frame + sizeof(eth_header_t) +
                         (ip->version_ihl & 0x0F) * 4,
                         ntohs(ip->total_length) -
                         (ip->version_ihl & 0x0F) * 4);
            break;

        case IPPROTO_UDP:
            udp_process_multicast(ip);
            break;
    }
}
```

5. Optimisations pour l'Embarqué

5.1 Gestion Mémoire

```c
// Pool statique pour éviter malloc/free
#define MAX_MCAST_GROUPS 8

static mcast_group_t group_pool[MAX_MCAST_GROUPS];
static uint8_t group_pool_bitmap = 0;

mcast_group_t* alloc_group(void) {
    for (int i = 0; i < MAX_MCAST_GROUPS; i++) {
        if (!(group_pool_bitmap & (1 << i))) {
            group_pool_bitmap |= (1 << i);
            memset(&group_pool[i], 0, sizeof(mcast_group_t));
            return &group_pool[i];
        }
    }
    return NULL;
}

void free_group(mcast_group_t *grp) {
    int idx = grp - group_pool;
    if (idx >= 0 && idx < MAX_MCAST_GROUPS) {
        group_pool_bitmap &= ~(1 << idx);
    }
}
```

5.2 Filtrage Matériel

Sur beaucoup de contrôleurs Ethernet embarqués, configurez les filtres MAC multicast pour réduire les interruptions:

```c
// Exemple pour STM32 ETH
void eth_add_multicast_filter(const uint8_t *mac) {
    uint32_t crc = ethernet_crc32(mac, 6);
    uint32_t hash = (crc >> 26) & 0x3F;  // 6 bits de hash

    if (hash < 32) {
        ETH->MACHTLR |= (1 << hash);
```

```
    } else {
        ETH->MACHTHR |= (1 << (hash - 32));
    }
}
```

5.3 Mode Sans IGMP

Pour réseaux locaux contrôlés, vous pouvez désactiver IGMP:

```c
void mcast_init_no_igmp(void) {
    mcast_mgr.igmp_enabled = 0;
    // Les routeurs ne sauront pas, mais ça fonctionne en LAN
}
```

6. Cas d'Usage Typiques

6.1 Découverte de Services (mDNS-like)

```c
#define SERVICE_DISCOVERY_GROUP 0xE0000251  // 224.0.2.81
#define SERVICE_PORT 5353

void announce_service(const char *name) {
    mcast_join_group(SERVICE_DISCOVERY_GROUP);

    char msg[128];
    snprintf(msg, sizeof(msg), "SERVICE:%s:IP:%08X",
             name, get_local_ip());

    mcast_send(SERVICE_DISCOVERY_GROUP, SERVICE_PORT,
               (uint8_t*)msg, strlen(msg));
}
```

6.2 Synchronisation Temporelle

```c
#define TIME_SYNC_GROUP 0xE0000123  // 224.0.1.35
#define TIME_SYNC_PORT 9999
```

```
void time_master_broadcast(uint32_t timestamp) {
    uint32_t ts_net = htonl(timestamp);
    mcast_send(TIME_SYNC_GROUP, TIME_SYNC_PORT,
               (uint8_t*)&ts_net, sizeof(ts_net));
}

void time_slave_listen(void) {
    mcast_join_group(TIME_SYNC_GROUP);
    // Réception dans le handler UDP
}
```

6.3 Streaming Audio/Vidéo

```c
#define AUDIO_STREAM_GROUP 0xEF000001  // 239.0.0.1
#define AUDIO_STREAM_PORT 8000

void stream_audio_packet(const uint8_t *samples, uint16_t len) {
    // Ajout d'un header RTP simplifié
    rtp_header_t rtp = {
        .version = 2,
        .payload_type = 96,  // Dynamic
        .sequence = get_next_seq(),
        .timestamp = get_audio_timestamp(),
        .ssrc = get_source_id()
    };

    uint8_t packet[sizeof(rtp) + len];
    memcpy(packet, &rtp, sizeof(rtp));
    memcpy(packet + sizeof(rtp), samples, len);

    mcast_send(AUDIO_STREAM_GROUP, AUDIO_STREAM_PORT,
               packet, sizeof(packet));
}
```

7. Débogage et Tests

7.1 Outils Système

```bash
# Linux: Joindre un groupe multicast
```

```
ip maddr add 224.0.1.2 dev eth0

# Écouter traffic multicast
tcpdump -i eth0 'net 224.0.0.0/4'

# Vérifier les groupes joints
netstat -g
```

7.2 Tests avec Socket BSD

```c
// Programme de test émetteur
int create_mcast_sender(void) {
    int sock = socket(AF_INET, SOCK_DGRAM, 0);

    struct sockaddr_in addr = {
        .sin_family = AF_INET,
        .sin_port = htons(9999),
        .sin_addr.s_addr = inet_addr("224.0.1.2")
    };

    // TTL multicast
    uint8_t ttl = 1;
    setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));

    char msg[] = "Hello multicast!";
    sendto(sock, msg, sizeof(msg), 0,
           (struct sockaddr*)&addr, sizeof(addr));

    return sock;
}

// Programme de test récepteur
int create_mcast_receiver(void) {
    int sock = socket(AF_INET, SOCK_DGRAM, 0);

    // Bind sur le port
    struct sockaddr_in addr = {
        .sin_family = AF_INET,
        .sin_port = htons(9999),
        .sin_addr.s_addr = INADDR_ANY
    };
```

```
    bind(sock, (struct sockaddr*)&addr, sizeof(addr));

    // Rejoindre le groupe
    struct ip_mreq mreq = {
        .imr_multiaddr.s_addr = inet_addr("224.0.1.2"),
        .imr_interface.s_addr = INADDR_ANY
    };
    setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
               &mreq, sizeof(mreq));

    return sock;
}
```

8. Considérations Avancées

8.1 TTL et Portée

Le TTL (Time To Live) définit la portée des paquets multicast:
- TTL = 1: Sous-réseau local uniquement
- TTL = 32: Organisation (site)
- TTL = 64: Région
- TTL = 255: Global (théorique)

8.2 Source-Specific Multicast (SSM)

IGMPv3 permet de spécifier la source:

```c
// Joindre (S,G) au lieu de (*,G)
typedef struct {
    uint32_t source_addr;
    uint32_t group_addr;
} ssm_membership_t;

// Filtrage: accepter seulement si source = X
int mcast_receive_ssm(uint32_t src_ip, uint32_t dst_group) {
    ssm_membership_t *ssm = find_ssm_entry(dst_group);
    if (ssm && ssm->source_addr != src_ip) {
        return 0;  // Rejeté
    }
    return 1;
}
```

```

### 8.3 Fiabilité (PGM/NORM)

Pour applications critiques, des protocoles de fiabilité existent:
- PGM (Pragmatic General Multicast): NAK-based
- NORM (NACK-Oriented Reliable Multicast): FEC + NAKs

Implémentation simplifiée avec séquencement:

```c
typedef struct {
    uint32_t sequence;
    uint16_t total_fragments;
    uint16_t fragment_id;
    uint8_t data[];
} reliable_mcast_t;

// Émetteur: numérotation séquentielle
// Récepteur: détection de trous et demande retransmission
```

9. Checklist Implémentation

- [ ] Conversion IP multicast → MAC multicast
- [ ] Gestion table des groupes (join/leave)
- [ ] Filtrage matériel des adresses MAC multicast
- [ ] Support IGMP v2 minimum (Query, Report, Leave)
- [ ] Timers pour Report suppression
- [ ] Gestion TTL des paquets sortants
- [ ] Vérification checksum IGMP
- [ ] Pool mémoire statique (si embarqué contraint)
- [ ] Tests avec tcpdump et iperf
- [ ] Documentation des groupes utilisés par l'application

10. Références

- RFC 1112: Host Extensions for IP Multicasting
- RFC 2236: Internet Group Management Protocol, Version 2
- RFC 3376: Internet Group Management Protocol, Version 3
- RFC 4541: Considerations for IGMP and MLD Snooping Switches
- IEEE 802.1D: MAC Bridges (multicast filtering)
```

Cette documentation vous donne les fondations pour implémenter une stack multicast légère en C. Adaptez selon vos contraintes matérielles et besoins applicatifs.