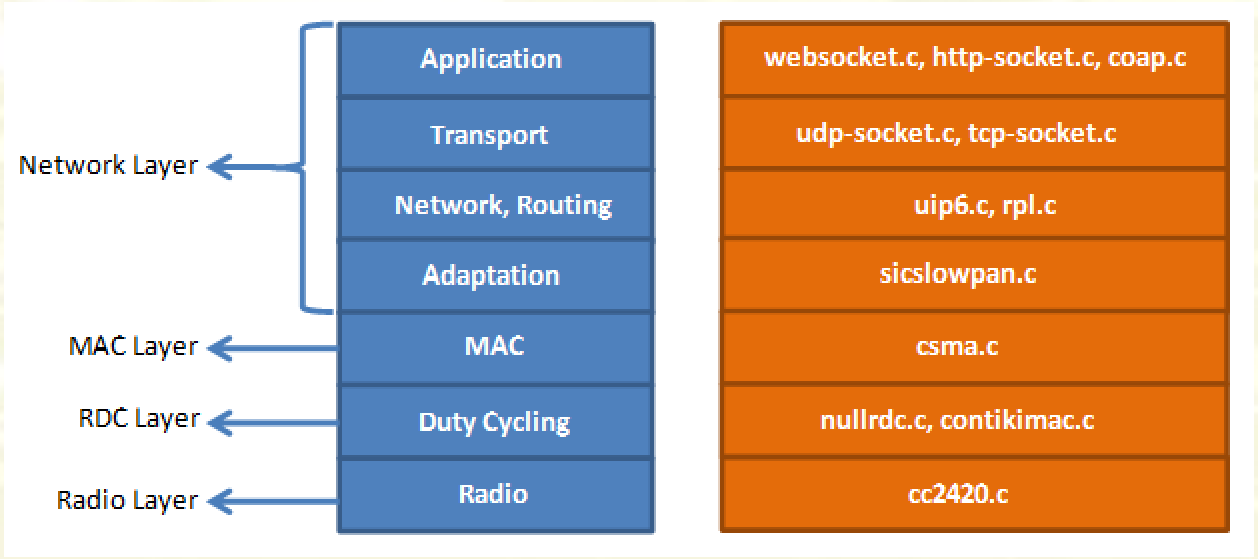
Basic Networking - IPv6

Carlo Vallati
Assistant Professor@ University of Pisa
c.vallati@iet.unipi.it

Networking

- Contiki implements a fully compliant TCP/IP stack:
  - IPv6, 6LoWPAN, RPL, TCP/UDP, CoAP/HTTP
  - MAC layers: CSMA, NullMAC
  - Radio Duty-Cycling (RDC) layers: ContikiMA, NullRDC



Contiki IPv6 assumptions

- Each node has a *single interface*
- Each interface can have up to UIP_NETIF_MAX_ADDRESSES unicast IPv6 addresses including its link-local address

Contiki IPv6 limtations

- uIP: world's smallest IP stack, implemented in constrained devices
- http://en.wikipedia.org/wiki/UIP_(micro_IP)

- Limited buffering capabilities
- Packet buffer shared through all the stack
- Some devices might have space for only one packet

Enable IPv6

http://contiki.sourceforge.net/docs/2.6/

To enable uIP add inside the Makefile

```
WITH_UIP6=1
UIP_CONF_IPV6=1
CFLAGS+= -DUIP_CONF_IPV6=1 -DWITH_UIP6=1
```

```
// Change the channel
#undef CC2420_CONF_CHANNEL
#define CC2420_CONF_CHANNEL 20
```

To set the channel

Include in the program:

```
#include "net/ip/uip.h"
#include "net/ipv6/uip-ds6.h"
#include "net/ip/uip-debug.h”
```

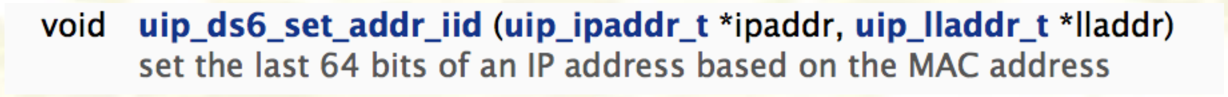
IPv6

- Manipulate IP addresses

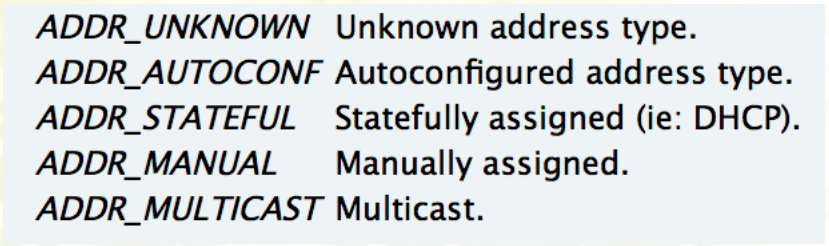
```
uip_ipaddr_t ipaddr;
uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
```

- Configure an interface

```
uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
```



void **uip_ds6_set_addr_iid** (**uip_ipaddr_t** *ipaddr, **uip_lladdr_t** *lladdr)
set the last 64 bits of an IP address based on the MAC address

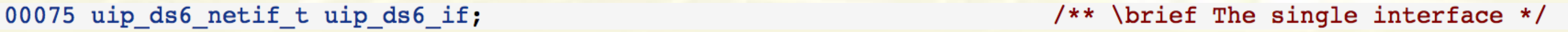


ADDR_UNKNOWN	Unknown address type.
ADDR_AUTOCONF	Autoconfigured address type.
ADDR_STATEFUL	Statefully assigned (ie: DHCP).
ADDR_MANUAL	Manually assigned.
ADDR_MULTICAST	Multicast.

IPv6

- Get all the IPv6 of a node

```
int i;
uint8_t state;
printf("IPv6 addresses: ");
for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
  state = uip_ds6_if.addr_list[i].state;
  if(uip_ds6_if.addr_list[i].isused) {
          uip_debug_ipaddr_print(
             &uip_ds6_if.addr_list[i].ipaddr);
          printf("\n");
  }
}
```



```
uip_ds6_netif_t uip_ds6_if;                                          /** \brief The single interface */
```

Do it!!

- Write a program that set an IPv6 address and retrieve all the IP addresses assigned to the node.

- Solution: get-address.c

Set the mote ID for Z1

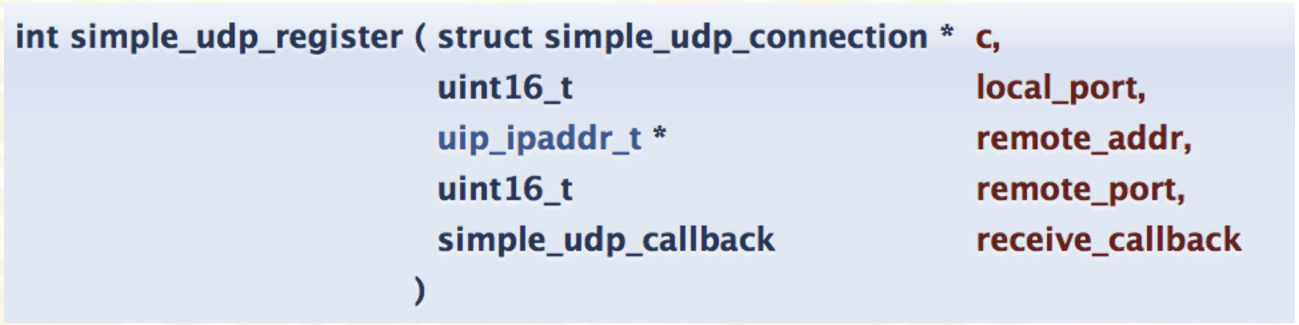
To set the node id:

make burn-nodeid.upload nodeid=158 nodemac=158

Z1 uses the Mote ID used to auto assign an IP address

Simple UDP - Initialization

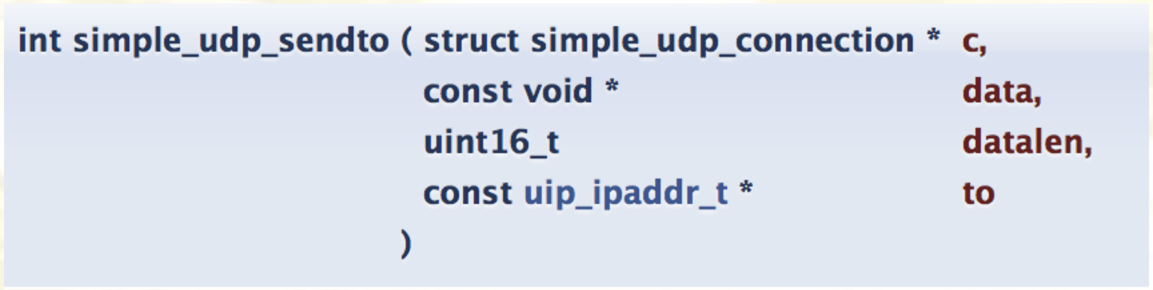
```
#include "simple-udp.h”

static struct simple_udp_connection
broadcast_connection;

simple_udp_register(&broadcast_connection, UDP_PORT,
                     NULL, UDP_PORT,
                     receiver);
```



```
int simple_udp_register ( struct simple_udp_connection *  c,
                          uint16_t                        local_port,
                          uip_ipaddr_t *                  remote_addr,
                          uint16_t                        remote_port,
                          simple_udp_callback             receive_callback
                        )
```

Simple UDP - Send

```
simple_udp_sendto(&broadcast_connection,
     "Test", 4, &addr);
```



```
int simple_udp_sendto ( struct simple_udp_connection *  c,
                        const void *                    data,
                        uint16_t                        datalen,
                        const uip_ipaddr_t *            to
                      )
```

Simple UDP - Receive

```
static void
receiver(struct simple_udp_connection *c,
        const uip_ipaddr_t *sender_addr,
        uint16_t sender_port,
        const uip_ipaddr_t *receiver_addr,
        uint16_t receiver_port,
        const uint8_t *data,
        uint16_t datalen)
{
    …
}
```

Neighbor Discovery

In order to allow peer to peer communication among two hosts you need to enable Neighbor Discovery. Add in project-conf.h file:

```
// Enable NA
#undef UIP_CONF_ND6_SEND_NA
#define UIP_CONF_ND6_SEND_NA            1
```

Do it!!

- Write a program that send periodically broadcast IPv6 packets
- If a packet is received, print something!

```
uip_create_linklocal_allnodes_mcast(&addr);
```

- Solution: broadcast-example.c

Do it!!

- Create two copies of the previous program:
  - One that only process received packets
  - One that periodically sends unicast packet to the other node with a message that contains a counter that is incremented every time

- Solution: unicast-sender.c / receiver.c

Multi-hop communication

- So far only **single hop** communication -> nodes must be in communication range

- What if we need multi-hop communication??
- Take a look at broadcast-routing.c the implementation of a simple 2-hop routing algorithm!