IoT Security – Autumn 2024

Lab 2: RPL UDP Communications and Capturing Traffic with Wireshark

Manh Bui
School of Electrical and Data Engineering
Email: DucManh.Bui@uts.edu.au

About me

- Name: Manh Bui
- You can call me **Manh**
- Research interests: Cybersecurity, Blockchain and Machine Learning
- Email/Teams: DucManh.Bui@uts.edu.au

Low-power and Lossy Networks (LLNs)

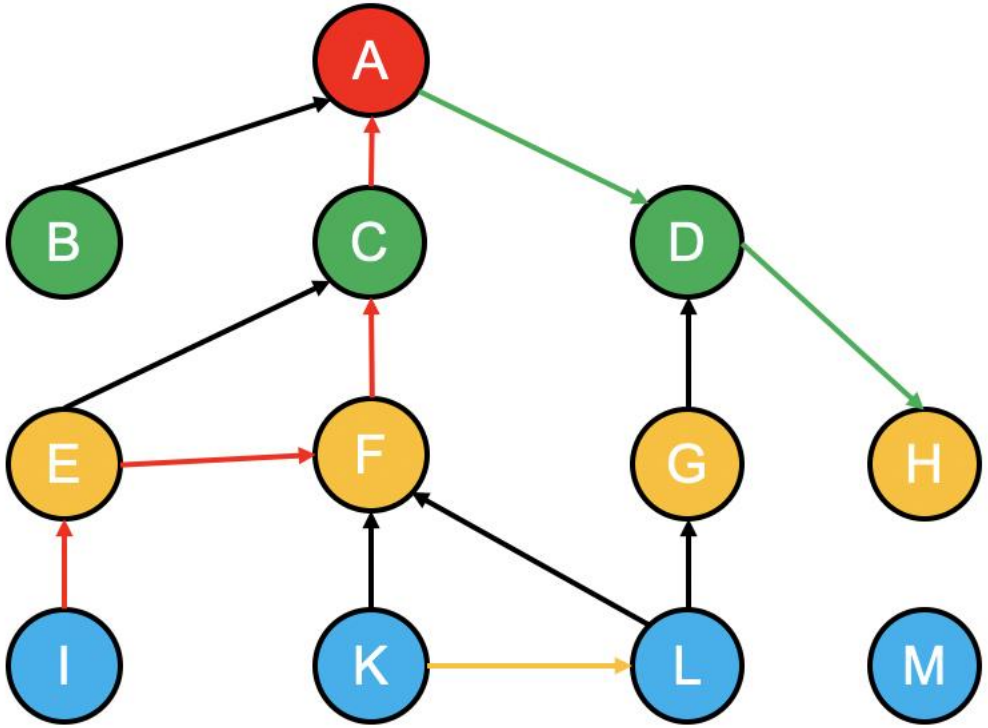
- Made up of many **embedded devices** with **limited power, memory,** and **processing resources.**
- Characterized by unstable links with **high loss rates, low data rates** and **low packet delivery rates**
- **LLNs** provide a **low-cost and low-power** approach to connect **sensors, devices, and machines** in an IoT network.
- The traffic patterns could be **Point-to-Point** (P2P) or **Point-to-Multipoint** (P2MP) or **Multipoint-to-Point** (MP2P).

RPL - IPv6 Routing Protocol for LLNs

- A routing protocol for **wireless networks** with **low power** consumption and generally susceptible to **packet loss**.
- A proactive protocol based on **distance vectors** and operates on IEEE 802.15.4.
- Supports P2P, **MP2P**, P2MP communications.
- Target **collection-based networks**, where nodes **periodically** send measurements to a collection point.
- Designed to be highly adaptive to network conditions and to provide alternate routes whenever default routes are inaccessible.
- Contains thousands of nodes

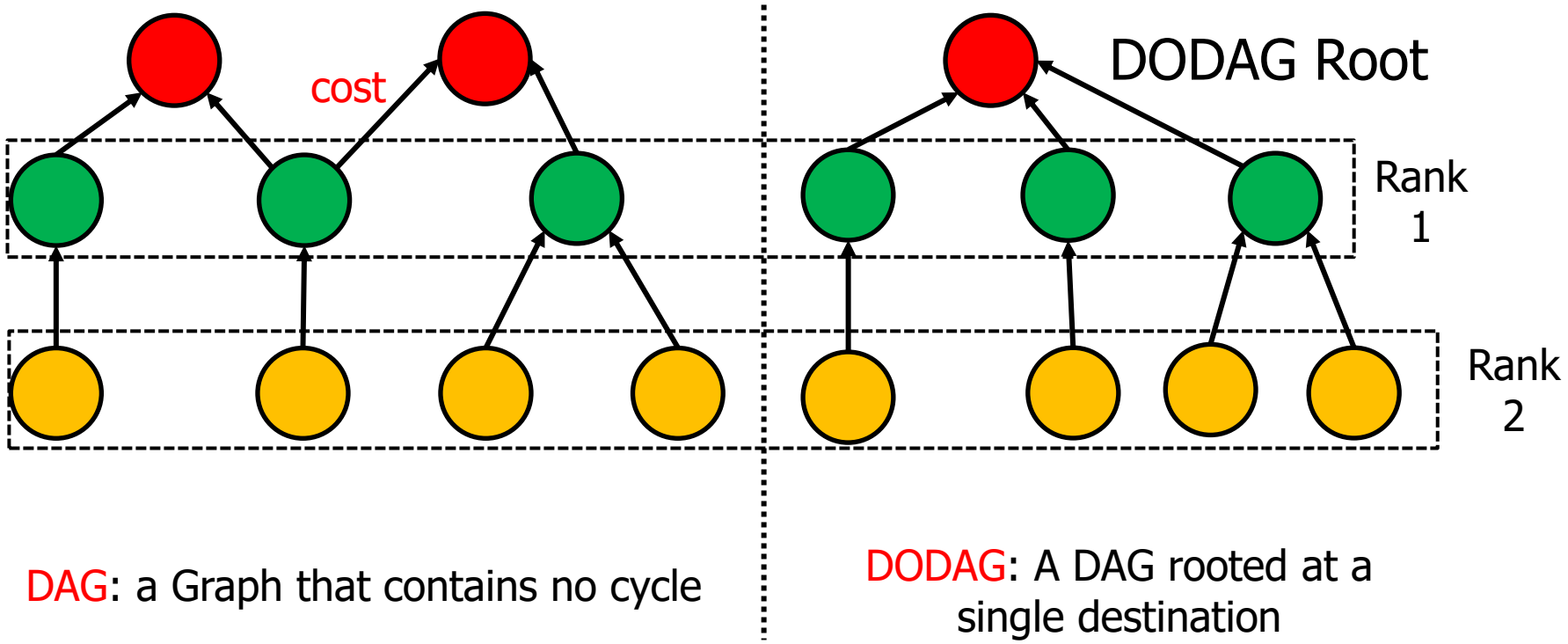
RPL - IPv6 Routing Protocol for LLNs

- Three types of traffic: MP2P, P2MP, and P2P



RPL - IPv6 Routing Protocol for LLNs

RPL organizes a topology as a Directed Acyclic Graph (DAG)



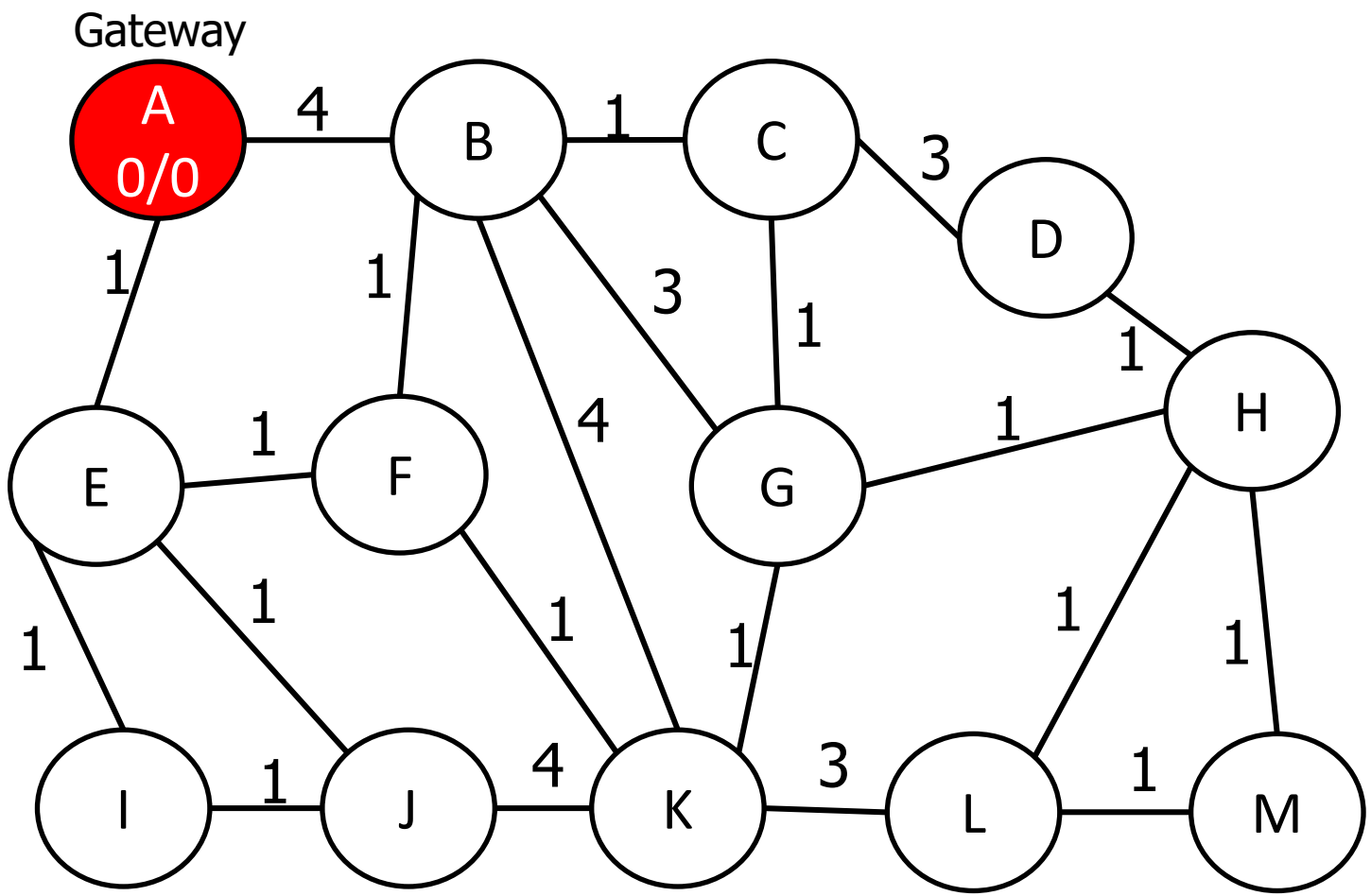
DAG: a Graph that contains no cycle

DODAG: A DAG rooted at a single destination

Each link has a cost (distance, latency, number of transmissions)

Routing based upon one or more DODAGs (i.e., DAG) to optimize objectives

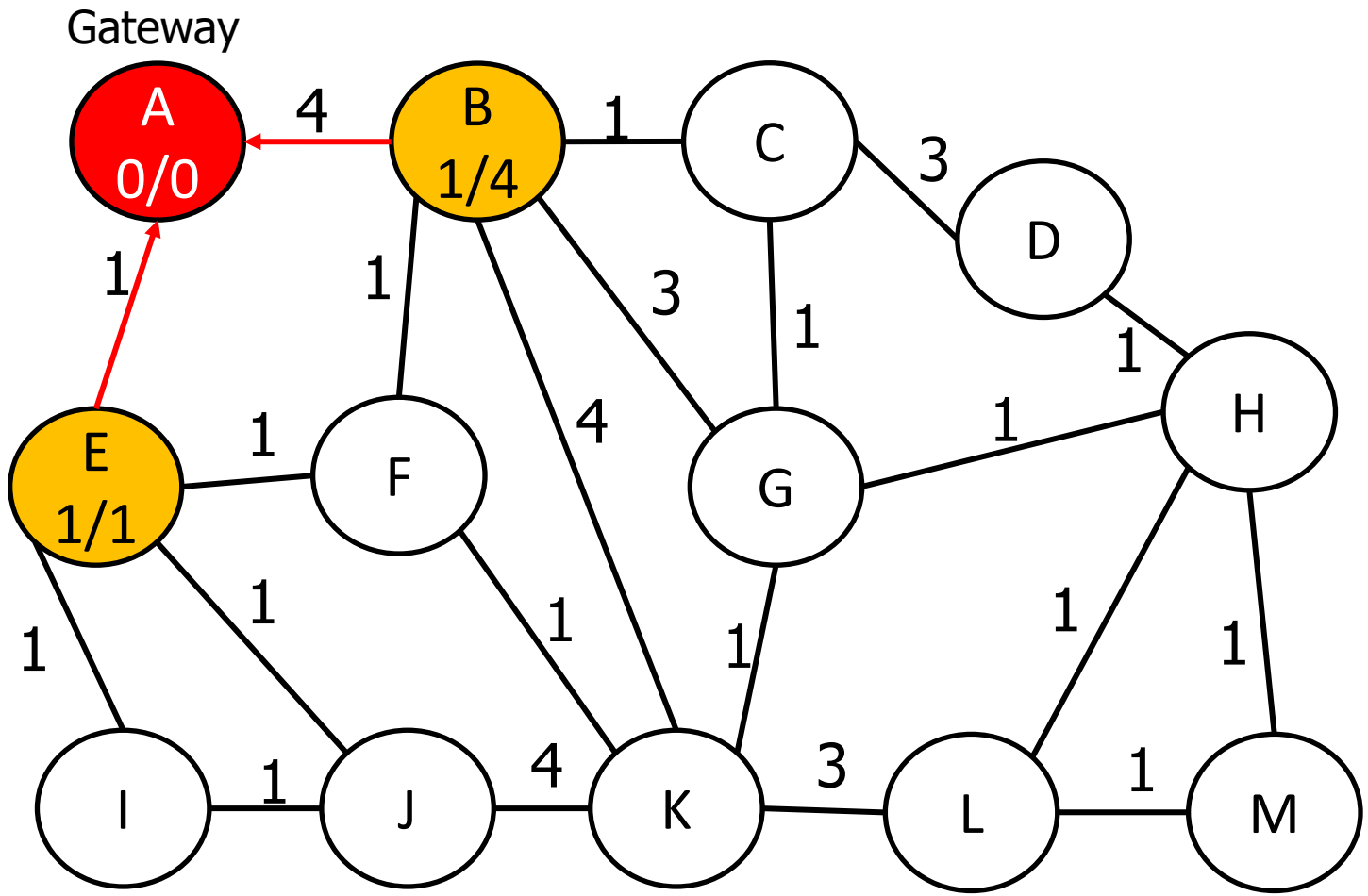
RPL - IPv6 Routing Protocol for LLNs



MP2P: sensor nodes send data to a gateway/root

Rank = hop count

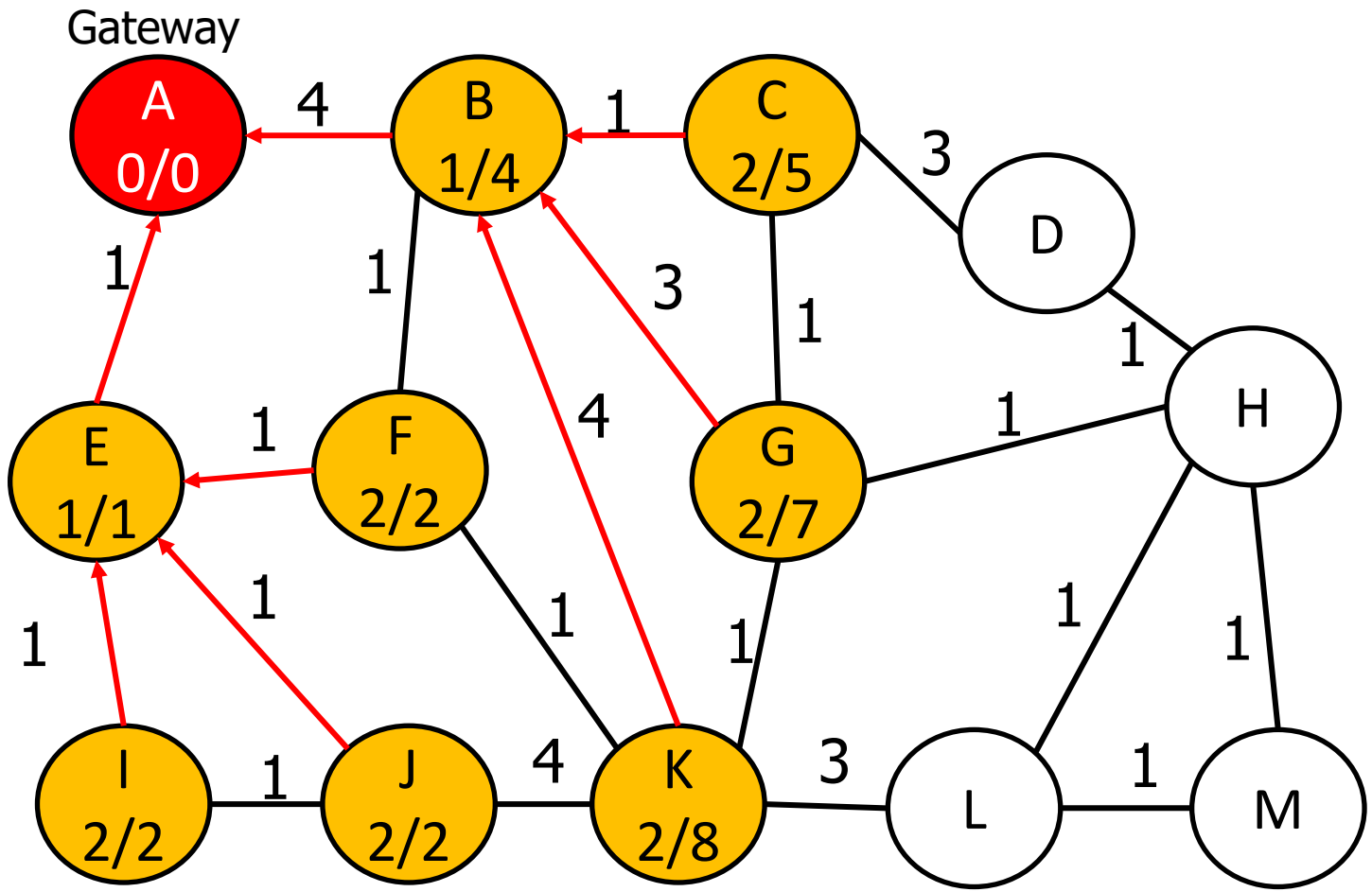
RPL - IPv6 Routing Protocol for LLNs



A sends DODAG Information Object (DIO) to B and E.

A becomes their preferred parent.

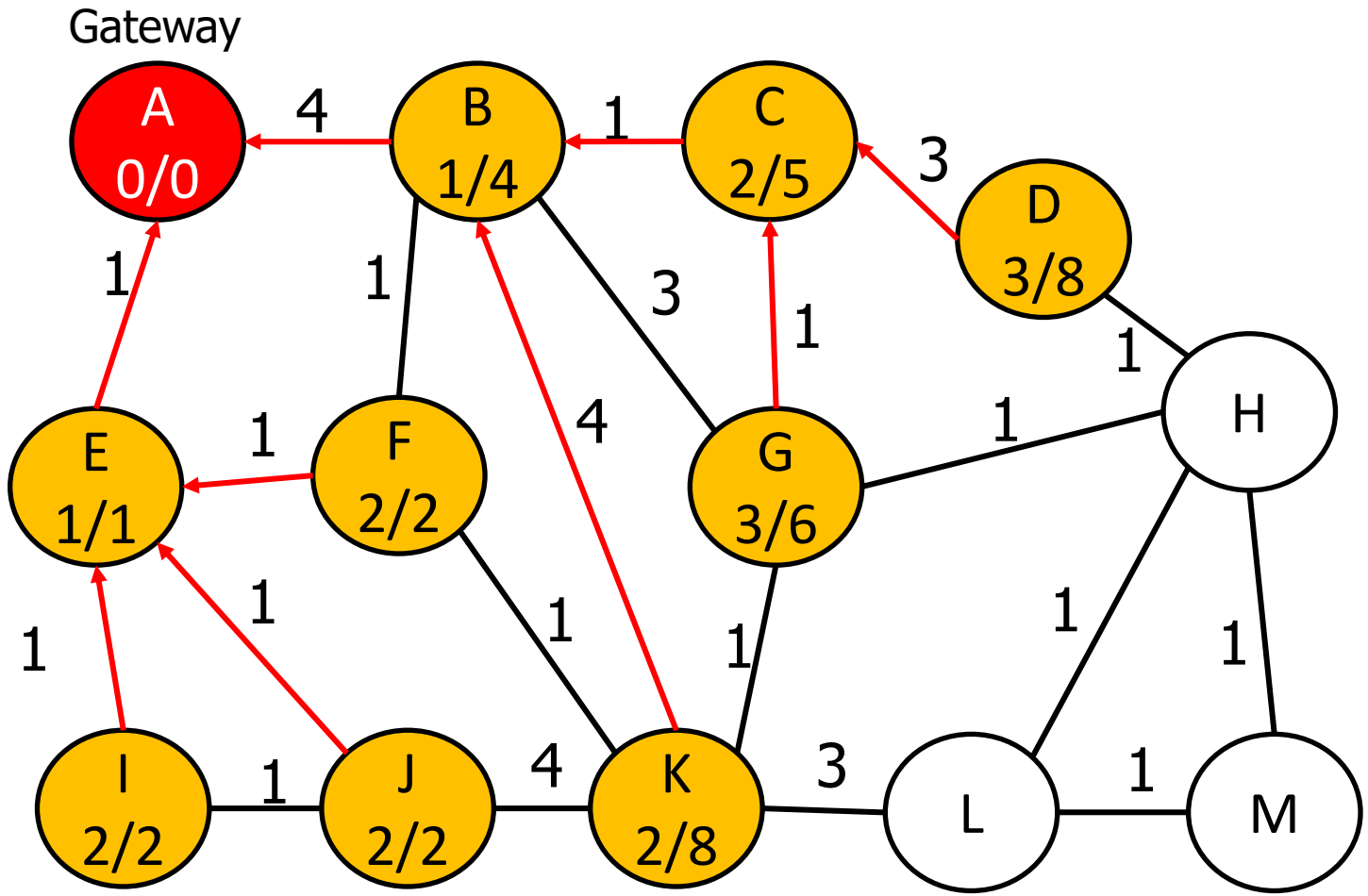
RPL - IPv6 Routing Protocol for LLNs



E sends DIO to A, F, I and J.
A ignores (rank).

E becomes the parent of F, I, and J.

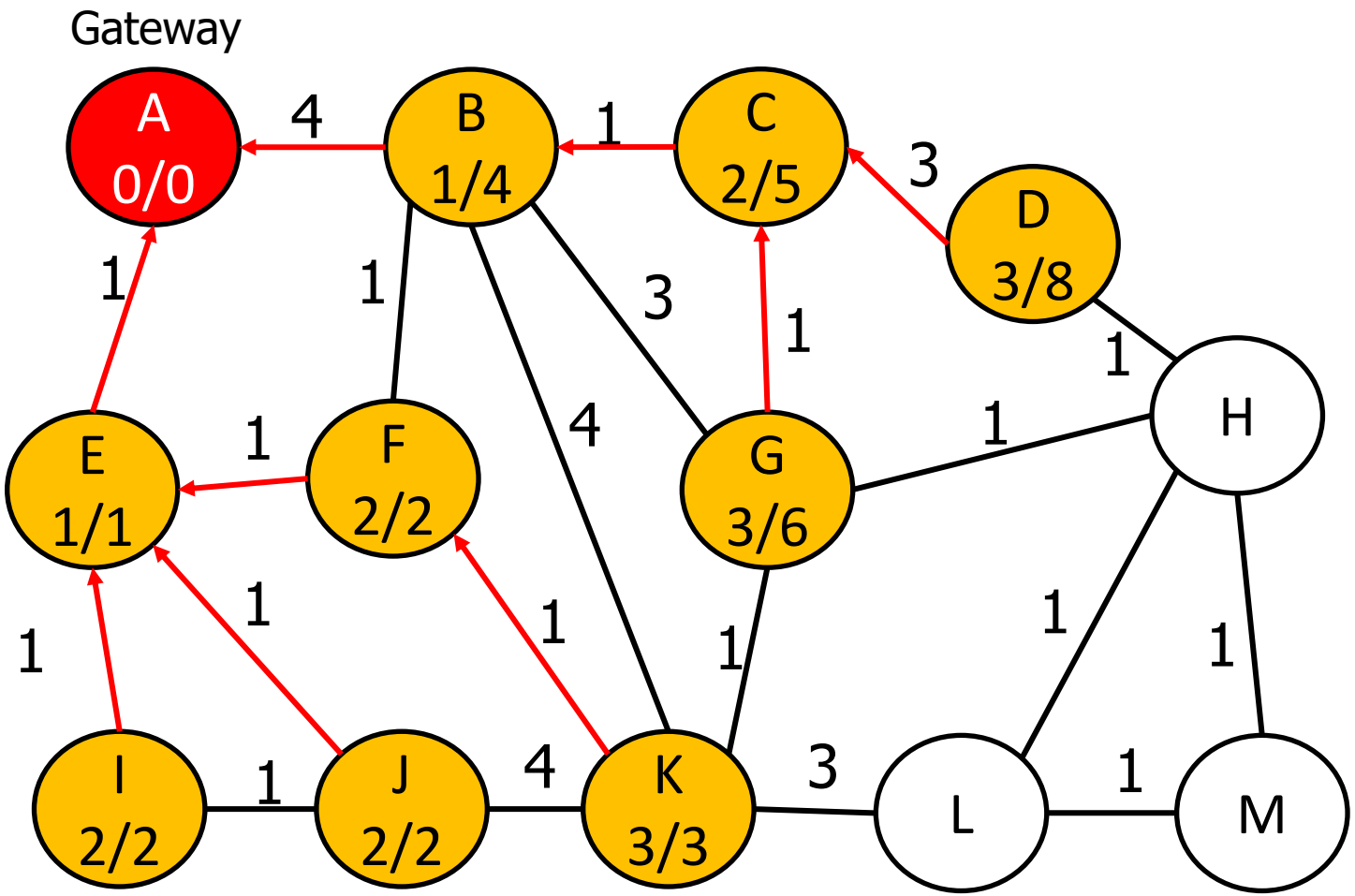
RPL - IPv6 Routing Protocol for LLNs



C sends DIO to B,D and G.
B ignores (rank).

C becomes the parent of D and G.

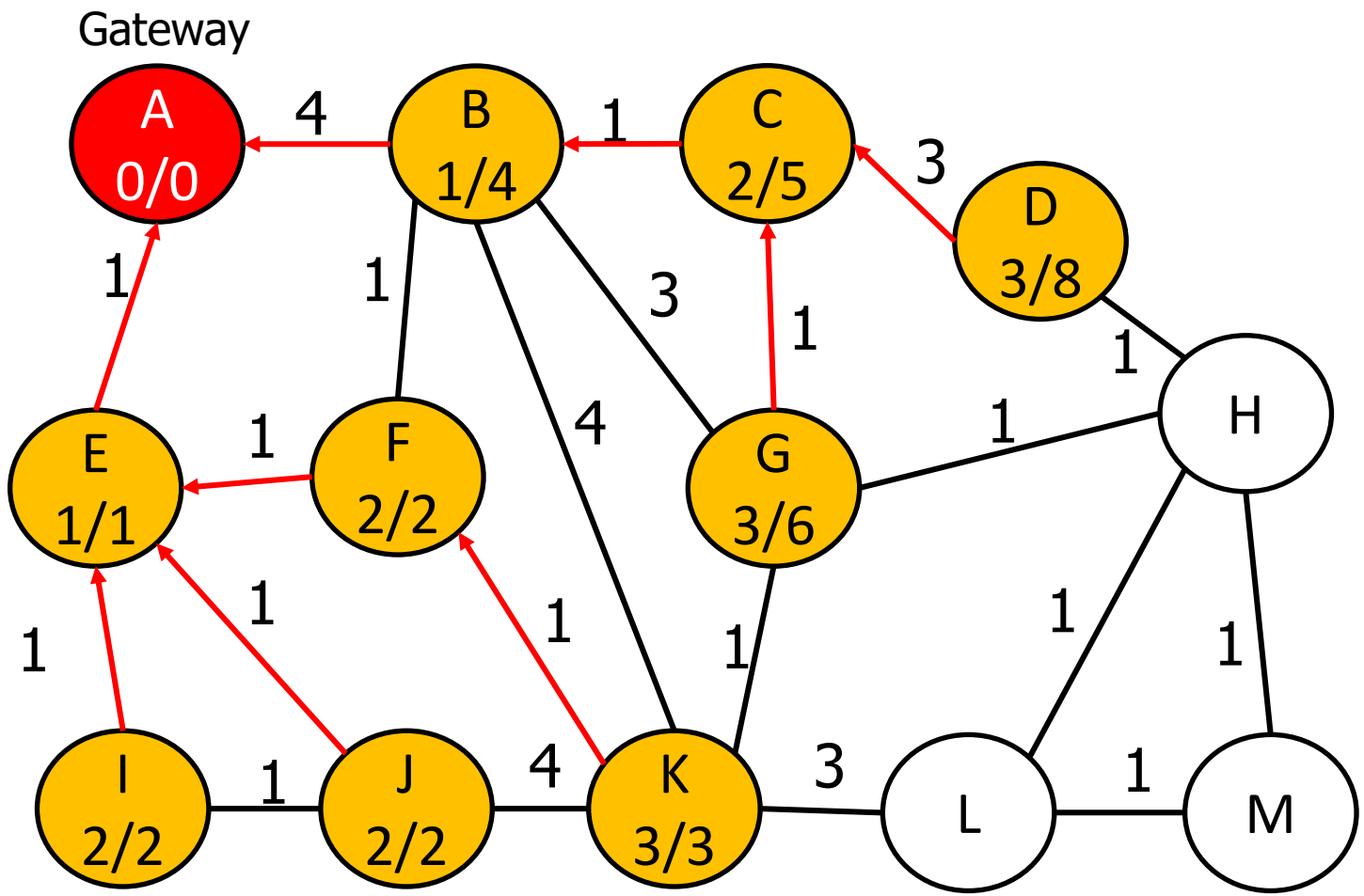
RPL - IPv6 Routing Protocol for LLNs



F sends DIO to B,E and K.

B and E ignore (rank), F becomes parent of K.

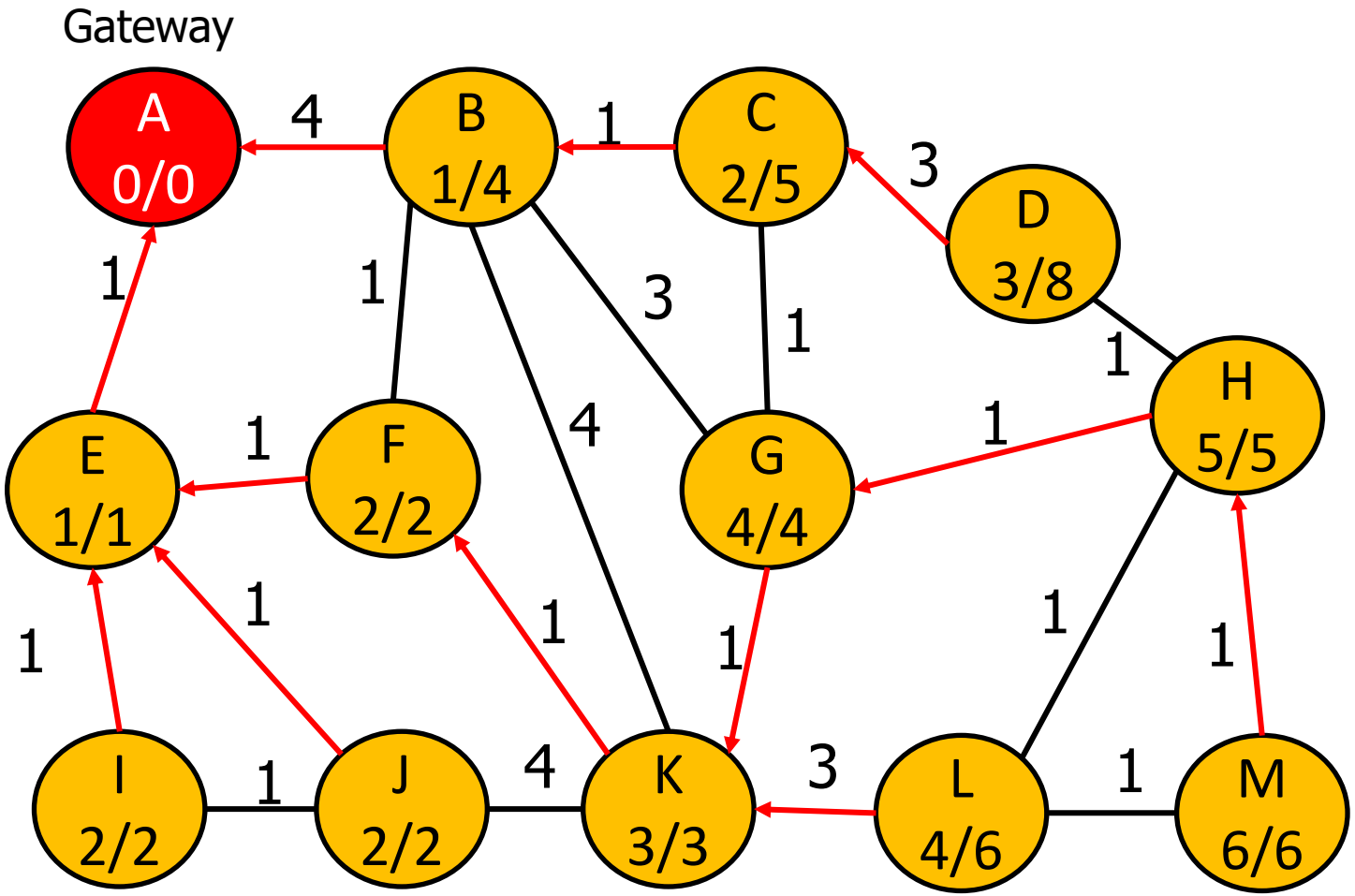
RPL - IPv6 Routing Protocol for LLNs



I sends DIO to E and J.

E ignores (rank),
J ignores (cost)

RPL - IPv6 Routing Protocol for LLNs



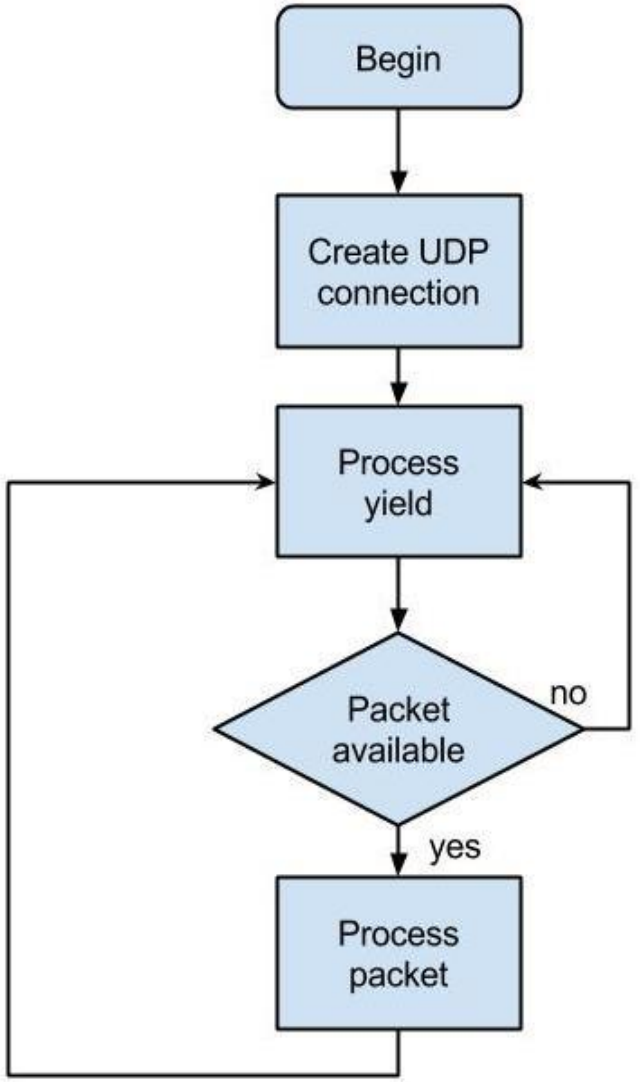
Continue the process until the entire DODAG is constructed

RPL UDP Communications

- UDP is implemented on top of RPL to **carry control messages** between **nodes** in the network
- UDP provides a **lightweight** and **efficient** mechanism for **transmitting packets** in RPL
- LLNs is comprised of a **UDP server**, which accepts available packets, and **several UDP clients**, which send packets **periodically** to the server through single-hop or multi-hops

UDP Server

- Initializes RPL DAG
- Sets up UDP connection;
- Waits for packets from client, receives and print them on `stdout`.
- `contiki/examples/ipv6/rpl-udp/udp-server`



Initialize RPL DAG

```
/* Mode 3 - derived from link local (MAC) address */
  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
#endif

  uip_ds6_addr_add(&ipaddr, 0, ADDR_MANUAL);
  root_if = uip_ds6_addr_lookup(&ipaddr);
  if(root_if != NULL) {
    rpl_dag_t *dag;
    dag = rpl_set_root(RPL_DEFAULT_INSTANCE,(ui                    dr);
    uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0,
    rpl_set_prefix(dag, &ipaddr, 64);
    PRINTF("created a new RPL dag\n");
  } else {
    PRINTF("failed to create a new RPL
  }
```

Set an IPv6 address for the server

Check whether the IP address was set successfully or not

Set the IP address of the server as the root of initial DAG

Set up a UDP Connection

```
server_conn = udp_new(NULL, UIP_HTONS(UDP_CLIENT_PORT), NULL);
if(server_conn == NULL) {
  PRINTF("No UDP connection available, exiting the process!\n");
  PROCESS_EXIT();
}
udp_bind(server_conn, UIP_HTONS(UDP_SERVER_PORT));

PRINTF("Created a server connection with remote address ");
PRINT6ADDR(&server_conn->ripaddr);
PRINTF(" local/remote port %u/%u\n", UIP_HTONS(server_conn->lport),
       UIP_HTONS(server_conn->rport));
```

Bring the connection to the server's local port

Create new UDP connection to client's port

Receive and Process incoming packet

```
while(1) {
  PROCESS_YIELD();
  if(ev == tcpip_event) {
    tcpip_handler();
  } else if (ev == sensors_ev[illegible] && data == &button_sensor) {
    PRINTF("Initiaing global repa[illegible]\n");
    rpl_repair_root(RPL_DEFAULT_INSTA[illegible]CE);
  }
}

PROCESS_END();
```

If there is a packet available

If the button is pressed, reset RPL DAG

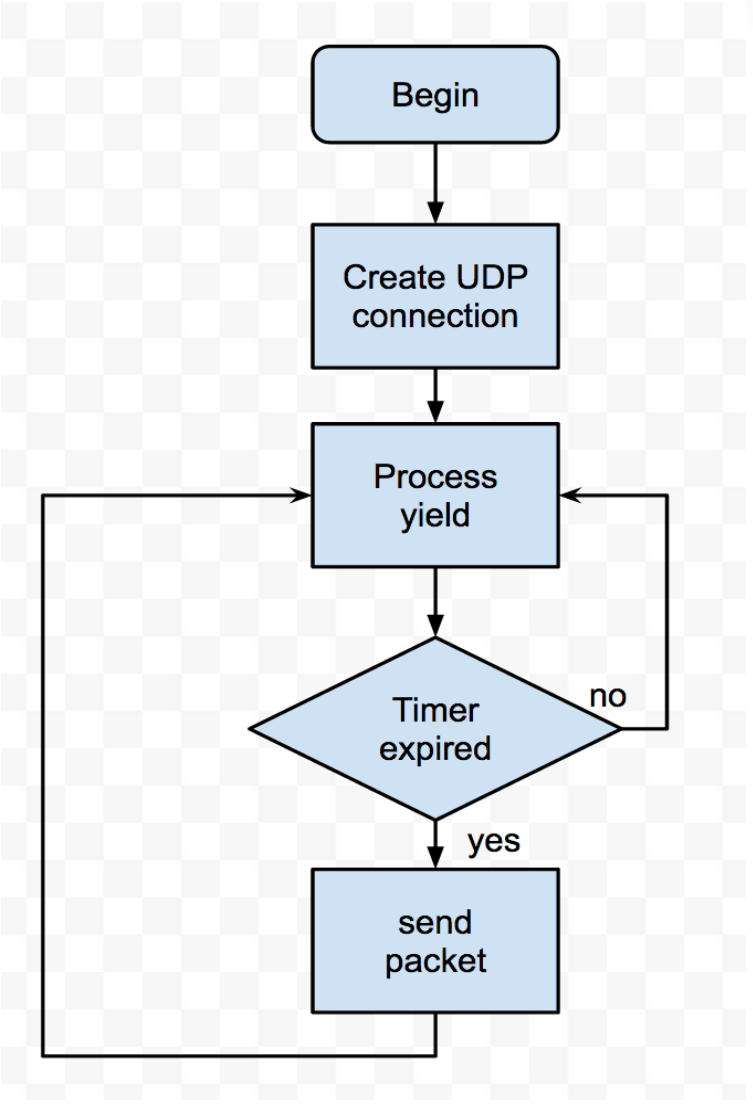
```
tcpip_handler(void)
{
  char *appdata;

  if(uip_newdata()) {
    appdata = (char *)uip_appdata;
    appdata[uip_datalen()] = 0;
    PRINTF("DATA recv '%s' from ", appdata);
    PRINTF("%d",
           UIP_IP_BUF->srcipaddr.u8[sizeof(UIP_IP_BUF
    PRINTF("\n");
```

Receive and print data

UDP Client

- Sets up UPD connection;
- Sends packet to UDP server periodically.
- contiki/examples/ipv6/rpl-udp/udp-client



Set up a UDP connection

```
/* new connection with remote host */
client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
if(client_conn == NULL) {
  PRINTF("No UDP connection available, ex ing the process!\n");
  PROCESS_EXIT();
}
udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_P T));

PRINTF("C ated a connection with the server
PRINT6ADDR &client_conn->ripaddr);
PRINTF(" cal/remote port %u/%u\n",
      UIP_ TONS(client_conn->lport), UIP_HTONS(c ent_conn->rport));
```

Bring the connection to the client's local port

Create new UDP connection server's port

Send packets

```
etimer_set(&periodic, SEND_INTERVAL);
while(1) {
  PROCESS_YIELD();
  if(ev == tcpip_event) {
    tcpip_handler();
  }
  if(etimer_expired(&periodic)) {
    etimer_reset(&periodic);
    ctimer_set(&backoff_timer, SEND_TIME, send_packet, NULL);
```

If there is a packet available, the UDP client will extract data (same as tcpip_handler() of the server)

Send a packet every SEND_INTERVAL

Send a packet to the UDP server through client_conn

```
  static int seq_id;
  char buf[MAX_PAYLOAD_LEN];

  seq_id++;
  PRINTF("DATA send to %d 'Hello %d'\n",
         server_ipaddr.u8[sizeof(server_ipaddr.u8) - 1], seq_id);
  sprintf(buf, "Hello %d from the client", seq_id);
  uip_udp_packet_sendto(client_conn, buf, strlen(buf),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
}
/*---------------------------------------------------------------------------*/
```

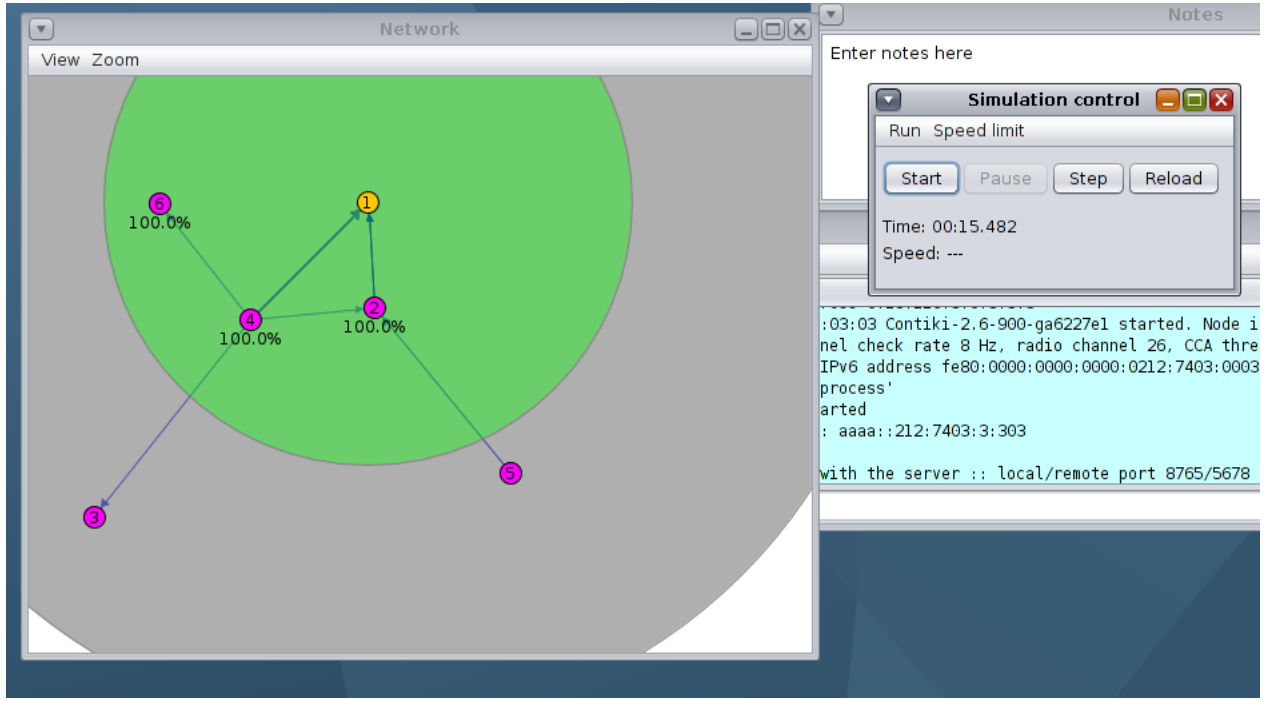
Cooja Simulation

- Create one server using the firmware at:

  contiki/examples/ipv6/rpl-udp/udp-server.c

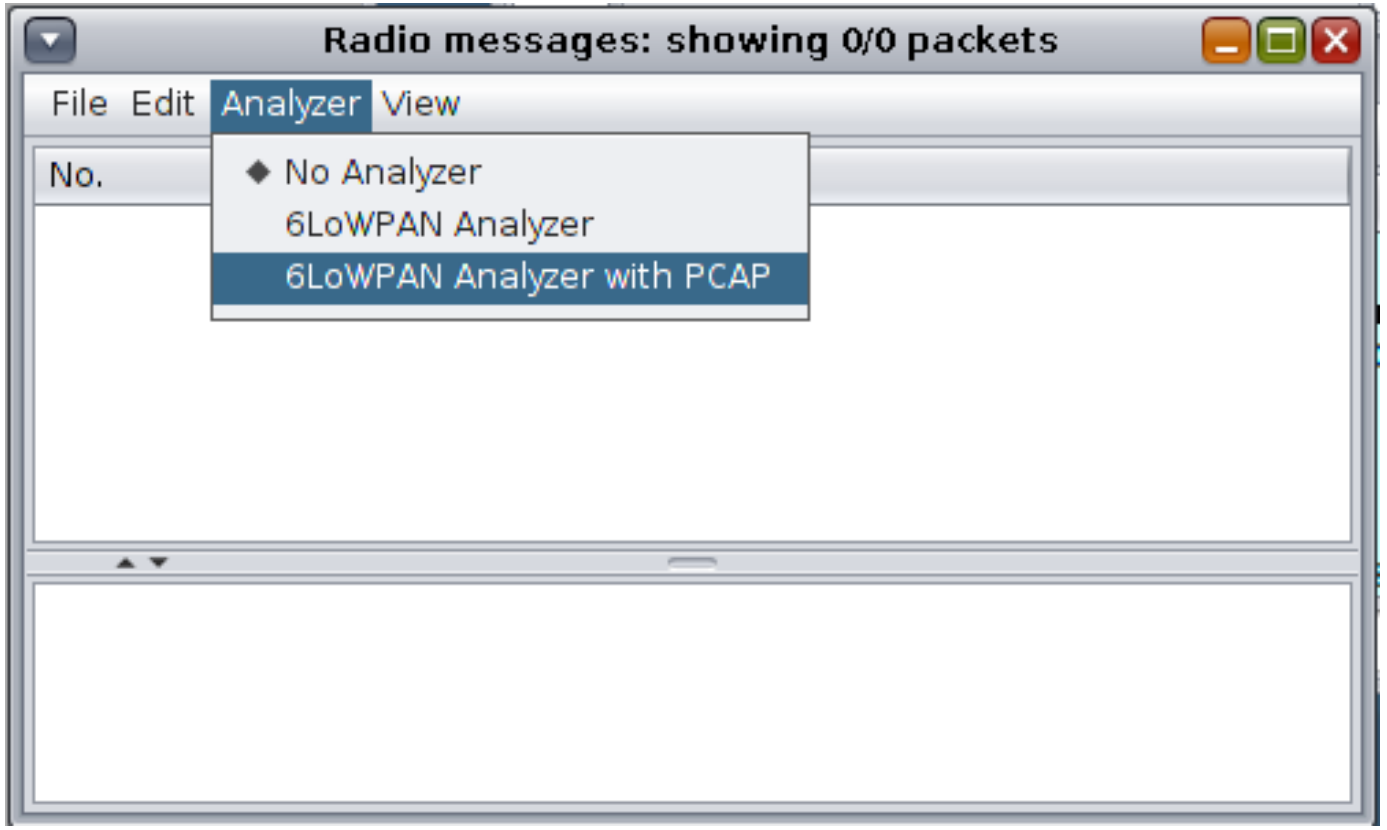
- Create 5 clients using the firmware at:

  contiki/examples/ipv6/rpl-udp/udp-client.c



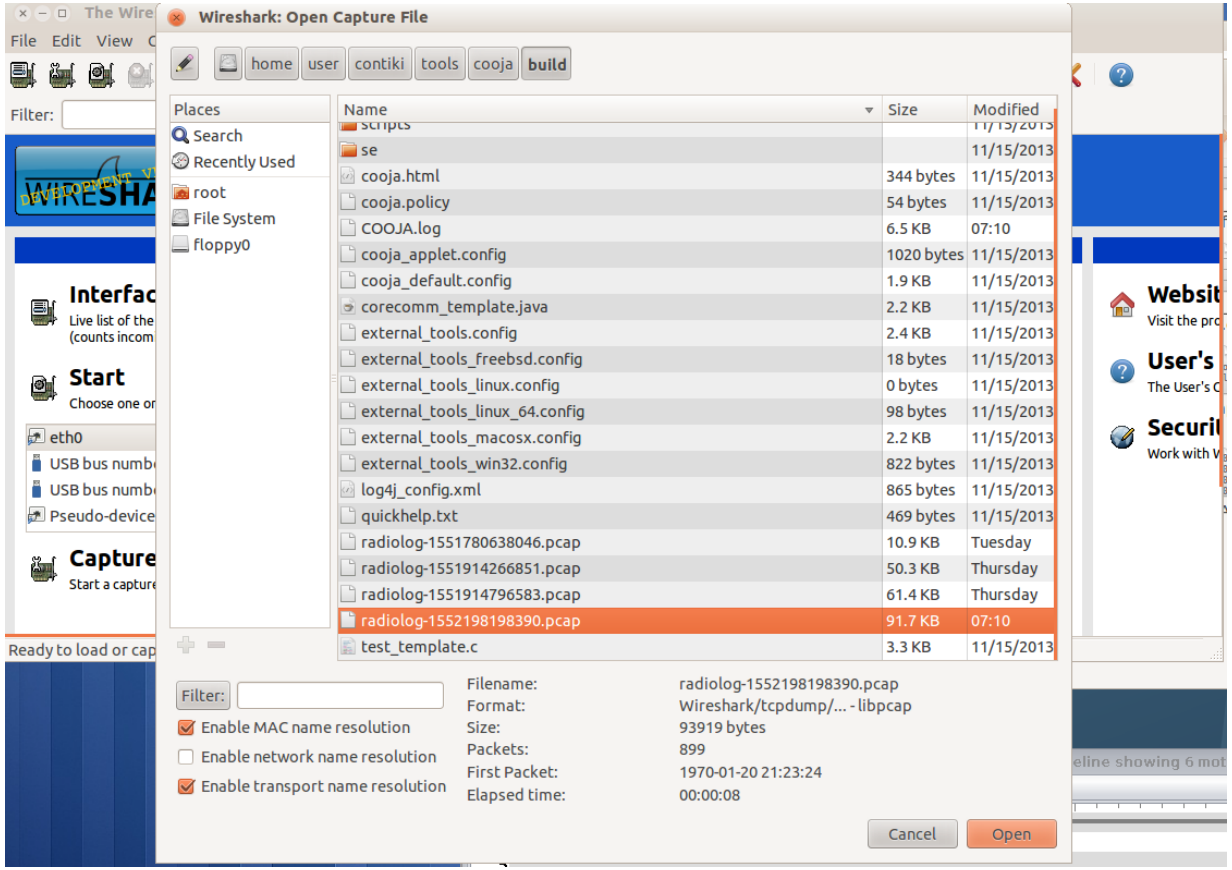
Capture traffic with Wireshark

To capture traffic, we need to choose 6LoWPAN Analyzer with PCAP: Tools -> Radio messages -> Analyzer -> 6LoWPAN Analyzer with PCAP



Capture traffic with Wireshark

- The PCAP file will be stored at: contiki/tools/cooja/build with the form: radiolog-xxxxxxx.pcap
- Use Wireshark to open this file and see traffic flows.



Exercises

1. Go to core/net/rpl and navigate through the C files, look for DEBUG defines and change its value to DEBUG_PRINT, this will print out to screen useful information allowing to better understand the RPL mechanics.
2. New application with multiple servers.
3. Change the client code to send different messages.

References

1. Get Started with Contiki: http://www.contiki-os.org/start.html
2. Contiki Tutorial: http://anrg.usc.edu/contiki/index.php/Contiki_tutorials
3. A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," IEEE international conference on local computer networks, Tampa, FL, USA, Nov. 2004.
4. A. L. Colina, A. Vives, A. Bagula, M. Zennaro, and E. Pietrosemoli, "IoT in 5 days" [Online]. Available: http://www.iet.unipi.it/c.vallati/files/IoTinfivedays-v1.1.pdf


Thank
You
Vinaka
köszönöm
Kiitos
Maake
Asante
Shukria
Dhanyavadagalu
감사합니다
Dankscheen
спасибо
Kam Sah Hammida
شكرا
Manana
Dankon
Dank Je
Mauruuru
Biyan
Diolch i Chi
Terima Kasih
Matondo
Taiku
Blagodaram
Ngiyabonga
Dziekuje
Chokrane
Tack
Grazie
Juspaxar
Arigato
Mochchakkeram
Gracias
Khap
நன்றி
Bedankt
Dakujem
धन्यवाद
cảm ơn bạn
Paldies
Tingki
Ua Tsaug Rau Koj
Grazas
Gratias Tibi
Obrigado
Děkuji
Nirringrazzjak
Di Ou Mèsi
Kia Ora
Kop Khun
ありがとう
Suksama
Hvala
Rahmat
Matur Nuwun
谢谢
хвала
Welalin
Djiere Dieuf
Misaotra
Danke
Merci
Go Raibh Maith Agat
Tuke
Eskerrik Asko
Salamat
ขอบคุณ
Najis