

# Session 4: Wiselib Routing Algorithms

## Algorithm Development

Tobias Baumgartner

Braunschweig Institute of Technology

October 13th, 2009

# Outline

- 1 Routing Algorithms
- 2 Routing Table
- 3 Special Issue: Hidden Functionality
- 4 Special Issue: Os Facet Internals
- 5 Outlook
- 6 Excercises

# Outline

- 1 Routing Algorithms
- 2 Routing Table
- 3 Special Issue: Hidden Functionality
- 4 Special Issue: Os Facet Internals
- 5 Outlook
- 6 Excercises

# Routing Concept

```

1| concept Routing
2| {
3|     typedef ... OsModel;
4|     typedef ... RoutingTable;
5|     typedef ... Radio;
6|     typedef ... Os;           ///< Can just be OsModel::Os
7|     typedef ... self_type;
8|     typedef ... node_id_t;
9|     typedef ... size_t;
10|    typedef ... data_t;
11|
12|    void enable( void );
13|    void disable( void );
14|
15|    void send( node_id_t receiver , size_t len , data_t *data );
16|
17|    template<class T, void (T::*TMethod)(node_id_t , size_t , data_t*)>
18|        int reg_rcv_callback( T *obj_pnt );
19|    void unreg_rcv_callback( int idx );
20|
21|    void set_os( Os* os );
22|    Os* os();
23| };

```

## Routing Concept (2 of 3)

- enable
  - Turn on radio
  - Ready to send and receive messages
- disable
  - Turn off radio
  - Can not receive messages (at least not forwarded to registered receivers)
  - Do not send messages
- send
  - Send data to given node
  - Not reliable (it can not be ensured that the data reaches the destination)

## Routing Concept (3 of 3)

- `set_os`
  - Set the `Os` pointer that can be passed to the `Os Facets`
- `reg_rcv_callback`
  - Register passed member function as callback when messages are received
  - Return unique id
- `unreg_rcv_callback`
  - Unregister the member function that is identified by the unique id

# Implementations

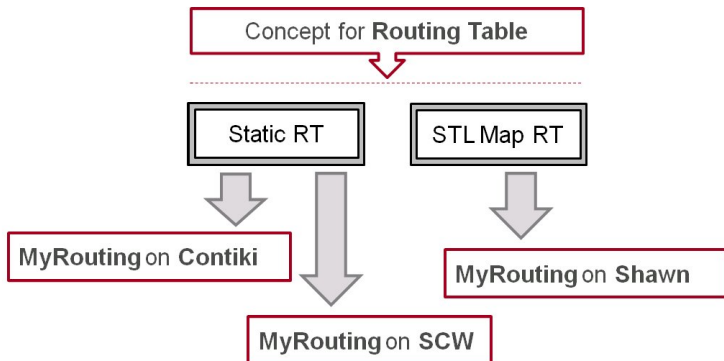
- Currently, there are several implementations available
  - DSDV
  - DSR
  - Tree
- Recently added by RACTI
  - AODV
  - Tora

# Outline

- 1 Routing Algorithms
- 2 Routing Table**
- 3 Special Issue: Hidden Functionality
- 4 Special Issue: Os Facet Internals
- 5 Outlook
- 6 Excercises

# Routing Table

- Concept for **Routing Table**
  - Provide routing entry for given node
  - Type of routing entry given as template parameter
  - STL container



## Example

- Routing table as template parameter

```

1 template<typename OsModel_P ,
2           typename RoutingTable_P ,
3           ... >
4 class DsdvRouting
5 {
6     typedef RoutingTable_P RoutingTable;
7     RoutingTable routing_table_;
8     ...

```

- Pass static routing table

```

1 typedef StaticArrayRoutingTable<Os, Os::Radio , 8,
2     DsdvRoutingTableValue<Os, Os::Radio> > DsdvRoutingTable;
3 typedef DsdvRouting<Os, DsdvRoutingTable> dsdv_routing_t;

```

- Pass `std::map` as routing table

```

1 typedef StlMapRoutingTable<Os, Os::Radio ,
2     DsdvRoutingTableValue<Os, Os::Radio> > DsdvRoutingTable;
3 typedef DsdvRouting<Os, DsdvRoutingTable> dsdv_routing_t;

```

# Outline

- 1 Routing Algorithms
- 2 Routing Table
- 3 Special Issue: Hidden Functionality**
- 4 Special Issue: Os Facet Internals
- 5 Outlook
- 6 Excercises

# Template Parameters in Algorithms

- Radio as template parameter

```

1 | template <... ,
2 |         typename Radio_P ,
3 |         ... >
4 | class MyAlgorithm
5 | {
6 |     typedef Radio_P Radio ;
7 |     ...

```

- *Blind* Usage

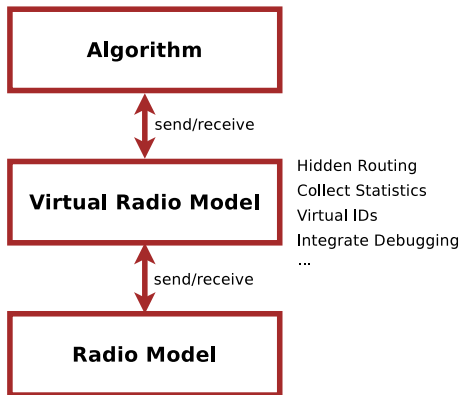
```

1 | template <... >
2 | void MyAlgorithm <... >::foo ()
3 | {
4 |     Radio::send( destination , len , data ) ;
5 | }

```

- Nothing known about what is hidden behind the Radio
  - Ordinary OS Facet such as ContikiRadio, iSenseRadio
  - Routing algorithm
  - Overlay Network
  - ...

# Example Usage



## Example: Uart Radio

- Radio that can also read from/write to the UART

```

1 | template<typename OsModel, typename Radio, typename Uart, ...>
2 | MyUartRadio

```

- Send to radio and UART

```

1 | static inline void
2 | send( Os *os, node_id_t to, size_t len, data_t *buf ) {
3 |     Uart::write( os, len, buf );
4 |     Radio::send( os, to, len, buf ); }

```

- Notify registered receivers on both incoming UART and radio packets

```

1 | static void rcv_uart_packet( size_t len, data_t* data ) {
2 |     notify_receivers( ... );
3 | }
4 |
5 | static void receive_message( node_id_t id, size_t len,
6 |                             data_t *buf )
7 |     notify_receivers( ... );
8 | }

```

## Example: Hidden Routing for Debug

- Debug interface that uses routing internally

```

1 | template<typename OsModel, typename Routing, typename Uart, ...>
2 | class RtDebug

```

- Route to gate when debug

```

1 | static void debug( Os *os, const char *msg, ... )
2 | {
3 |     routing_.send( ... );
4 | }

```

- On received routing messages, print to Uart

```

1 | static enable( Os *os ) {
2 |     routing_.reg_rcv_callback<RtDebug, &RtDebug::rcv>(os);
3 | }
4 |
5 | static rcv( node_id_t id, size_t len, data_t *buf ) {
6 |     Uart::write( os_, len, buf );
7 | }

```

# Outline

- 1 Routing Algorithms
- 2 Routing Table
- 3 Special Issue: Hidden Functionality
- 4 Special Issue: Os Facet Internals**
- 5 Outlook
- 6 Exercises

# Basics

- Os Facets are the connection to the underlying OS
- Most Preferably: Just one-liner that can be directly inlined
  - > No overhead, neither run-time nor memory
- **But:** Not possible in any case!
- Especially when registering callbacks in C-based OSs
  - > Member function pointers not compatible with C function pointers
- Sometimes replication of OS functionality required

## Example: Debug

```
1 template<typename OsModel_P>
2 class ContikiDebug
3 {
4 public :
5     typedef OsModel_P OsModel;
6     typedef typename OsModel::Os Os;
7     // _____
8     static void debug( Os *os, const char *msg, ... )
9     {
10         va_list fmtargs;
11         char buffer[0xff];
12         va_start( fmtargs, msg );
13         vsnprintf( buffer, sizeof(buffer) - 1, msg, fmtargs );
14         va_end( fmtargs );
15         printf( "%s", buffer );
16     }
17 };
```

# Radio Concept

```

1 | concept Radio
2 | {
3 |     typedef ... OsModel;           typedef ... Os;
4 |     typedef ... node_id_t;         typedef ... data_t;
5 |     typedef ... size_t;           typedef ... message_id_t;
6 |
7 |     enum SpecialNodeIds
8 |     { BROADCAST_ADDRESS = ..., NULL_NODE_ID = ... };
9 |     enum Restrictions
10 |    { MAX_MESSAGE_LENGTH = ... };
11 |
12 |     static int
13 |     send( Os *os, node_id_t id, size_t len, const data_t *data );
14 |
15 |     static int enable( Os *os );
16 |     static int disable( Os *os );
17 |     static node_id_t id( Os *os );
18 |
19 |     template<class T, void (T::*TMethod)(node_id_t, size_t, data_t*)>
20 |     static int reg_rcv_callback( Os *os, T *obj_pnt );
21 |
22 |     static void unreg_rcv_callback( Os *os, int idx );
23 | };

```

## Example: ContikiRadioModel

```
1  static void send( Os *os, node_id_t id, size_t len, data_t *data )
2  {
3      uint8_t buf[RIMEBUF_SIZE];
4
5      struct sender_info_t *si = (struct sender_info_t*)buf;
6      si->src = *rimeaddr_node_addr.u16;
7      si->dest = id;
8      memcpy( buf + sizeof(sender_info_t), data, len );
9
10     rimebuf_clear();
11     rimebuf_copyfrom( buf, len + sizeof(sender_info_t) );
12     abc_send( &contiki_radio_conn );
13 }
```

```
1  static node_id_t id( Os *os )
2  {
3      return node_id;
4  }
```

## Example: iSenseRadioModel

```
1  static void  
2  send( Os *os , node_id_t id , size_t len , block_data_t *data )  
3  {  
4      os->radio().send( id , len , data , 0 , 0 );  
5  }  
6  
7  static void enable( Os *os )  
8  {  
9      os->radio().enable();  
10 }  
11  
12 static void disable( Os *os )  
13 {  
14     os->radio().disable();  
15 }  
16  
17 static node_id_t id( Os *os )  
18 {  
19     return os->id();  
20 }
```

# Outline

- ① Routing Algorithms
- ② Routing Table
- ③ Special Issue: Hidden Functionality
- ④ Special Issue: Os Facet Internals
- ⑤ Outlook**
- ⑥ Excercises

# Current and Future Work

- Algorithms
  - Clustering algorithms/Topology Control
  - Localization
  - Time Synchronization
  - ...
- Os Facets
  - Clock
  - More radio concepts (RSSI and LQI, reliable transfer, ...)
  - ...
- Internal Datastructures
  - Position
  - Concept for Time
  - ...

# Outline

- 1 Routing Algorithms
- 2 Routing Table
- 3 Special Issue: Hidden Functionality
- 4 Special Issue: Os Facet Internals
- 5 Outlook
- 6 Exercices**

## Exercise for this Session

- Combine your FloodingAlgorithm with TreeRouting
  - Use your FloodingAlgorithm to construct a tree in a network that can be used to route messages back to the gateway (which initiates the flooding).
  - A template is provided; but use also the existing TreeRouting (in `wiselib.stable/algorithms/routing/tree_routing.h`) for copy & paste
- Note that you are able to register multiple receive functions, e.g.

```

1 | flooding_.template reg_rcv_callback <
2 |     self_type , &self_type::receive_flooding >(this );
3 | Radio::template reg_rcv_callback <
4 |     self_type , &self_type::receive_radio >( os(), this );

```

That can be used, for example, for tree construction (`receive_flooding`) and routing back to the gate (`receive_radio`)