Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Berner Fachhochschule
Technik und Informatik

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Implementing a Protocol for Maintenance and Construction of Network Topologies in TOWN

TIK-Report xxx

Ulrich Fiedler, fiedler@tik.ee.ethz.ch

Pietro Cerutti, pietro.cerutti@bfh.ch

Reto Kohlas, reto.kohlas@bfh.ch

August 9, 2007

Implementing a Protocol for Maintenance and Construction of Network Topologies in TOWN

Pietro Cerutti
pietro.cerutti@bfh.ch

Ulrich Fiedler
ulrich.fiedler@bfh.ch

Reto Kohlas
reto.kohlas@bfh.ch

August 9, 2007

Abstract

The goal of the TOWN project is to lay the technical foundations for ASCOM to build routers with multiple IEEE 802.16-2004 interfaces that can form wireless multi-hop networks for disaster recovery in a self-organized way.

The main scope of this report is the design of a protocol that implements a previously proposed topology construction and channel allocation algorithm in a TOWN network. The protocol is inspired by OSPF. Each router tracks the state of all links in the network with its own copy of a network database. Additionally, each routers tracks scan results of all potential links in the network. Changes in link states and new scan results are opportunistically flooded through the network. A conflict resolution protocol resolves possible inequalities between neighboring routers. A modified Dijkstra algorithm running locally ensures that router joins lead to network topologies that are optimized for capacity and load balancing. This algorithm also includes a greedy channel allocation to minimize interference of wireless routers in range.

For a proof of concept, we head towards implementing the protocol in a test environment which we call demonstrator. The demonstrator is a well equipped PC which hosts up to three dozens of virtual routers running the protocol. We review issues concerning the implementation of this demonstrator such as how to drive this emulation and how to emulate the wireless network. Finally, we conclude with a summary, time and implementation plan.

Contents

1 Introduction

Self-organizing wireless metropolitan area networks enabling telephony applications are of particular interest to rescue forces in disaster recovery. Such networks can be formed by routers employing multiple commercially available IEEE 802.16-2004 (WiMAX) interfaces.

TOWN (Telephony over Wireless Metropolitan Area Networks) is the name of a joint project between TIK/ETHZ and ASCOM AR&T. Its main goal is to lay the technological foundations to develop wireless routers which can form wireless multi-hop networks in a self-organized way once dispersed on the site of a disaster.

1.1 Project background

The project has started in April 2006 and is scheduled to end in February 2008. BFH has joined the project as a new partner in October 2006 with Ulrich Fiedler being transferred from TIK/ETH to BFH. During these 14 months we have identified and evaluated an algorithm for self-organized topology construction and channel allocation in a wireless multi-hop network with routers that employ two IEEE 802.16-2004 interfaces and that have at least one gateway to the wired network. Now we head towards implementing this algorithm on top of a demonstrator system and of top of ASCOM hardware.

1.2 The topology construction algorithm

The topology construction algorithm, proposed in [1], constructs a network topology and assigns channels to the cells made up by base station interfaces of the routers. The algorithm maximizes the network capacity between each router and the master gateway, balances the load on the gateways, and minimizes the interference on wireless links. It assumes the following configuration on router interfaces (see Fig. 1 for an illustration): In gateways, both interfaces are set to base station (BS) mode to open up two cells to enhance capacity. In routers, the first interface is set to subscriber station (SS) mode to connect the router to the network by joining an existing cell. The second interface is set to base station (BS) mode to open up a cell and to offer connectivity to additional routers. When used to transport traffic, this BS interface is assigned a specific channel. Otherwise this interface runs on the default management channel to make the potential cell visible to join candidate routers. This configuration inherently leads to a tree topology with parent/child relationships.

The overall concept of the self-organizing topology construction/channel allocation algorithm is as follows: The network is build constructing a spanning tree starting at the master gateway. To decide which links to actually

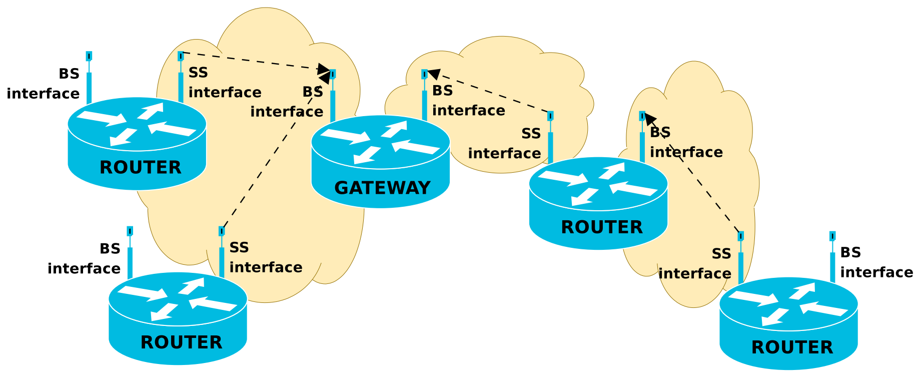

Figure 1: The topology of a TOWN network is constructed starting at a gateway that is connected to the wired network. This master gateway configures both wireless interfaces to base station mode, offering connectivity to routers joining the network. The other routers configure one of their interfaces to subscriber station mode to connect to their assigned parent router, and offer further connectivity via the other interface, which is configured to base station mode. Routers measure downlink receiver SNRs of potential links before connecting. The topology construction algorithm is employed to determine the network topology.

use to build the tree, the algorithm first gathers SNR measurements to estimate link qualities. Based on these link qualities, the algorithm decides which router is next to join the spanning tree. Then routers in the tree exchange topology and SNR information before inviting the next router to join. The channels of the router's interface in BS mode are allocated greedily with the least noisiest channel first. For details refer to the WEIRD'07 paper [1].

1.3 Scope of this report and of the demonstrator

In this document, we describe (i) the design of a *protocol* for running the algorithm on a set of routers and in a distributed manner (ii) how to implement the protocol on *virtual routers* that run on a single PC. This system of virtual machines which emulate the protocol is called the *demonstrator*. The virtual machines are configured in such a way that they are as close as possible to the target hardware.

The scope of the demonstrator implementation is a proof-of-concept for the proposed algorithm for self-organized topology construction and channel allocation. To enable systematic testing, the demonstrator runs a unit that drives the emulation and emulates the wireless network.

Out of scope of the implementation are a full product development, excessive state machine testing and tuning of the configuration (e.g., timers).

The implementation proposed in this report assumes that the topology construction protocol accounts solely for downlink receiver SNRs of beacons measured at the time the candidate router sends an invite request to the temporary parent router (i.e., we assume that the IEEE 802.16-2004 equipment can only provide downlink receiver SNRs during operation.). Uplink receiver SNRs, which require background scan during network operation, are assumed not to be available due to stringent outage requirements of traditional telephony operations. Changes of downlink receiver SNR during operation, which would also require background scan, are also assumed not to be available.

1.4 Outline

The remaining part of this report is structured as follows. In Section 2 we start by describing the overall functioning of the protocol. Section 3 is devoted to the presentation of the major data structures, in particular the topology database. In Section 4 we explain how message exchange by a flooding mechanism works and in Section 5 we describe how inconsistencies in databases lying on different routers are resolved in our protocol. Then we continue by detailing how the process of routers joining and leaving the network are handled (Sections 6 and 7). In Section 8 we specify the format and the handling of the different messages that are part of our protoco. Section 9 discusses demonstrator-related issues such as how to drive the emulation. We conclude by a summary in Section 10.

2 Design overview

Our protocol for a TOWN network operation and network topology construction adopts many design aspects of OSPF [3], which is commonly used as a routing protocol in wired networks. We propose to divide our overall protocol into the following four disjoint subprotocols: a *database conflict resolution* and a *hello* protocol (which are both run on a regular basis during normal operation), as well as a *router join* and a *router leave* protocol (which change the topology of the network).

2.1 Normal operation

In our protocol, each router keeps both a copy of the so-called *network topology database* (called *topoDB* for short) and *scantable database* (also called *scanTabDB*). The topoDB and the scanTabDB are both lists of network links. The topoDB keeps track of the full topology information of the network. It includes information on existing associations to base stations (links) as well the corresponding channel configurations. The scantabDB contains information on potential associations, that is potential links, measured SNRs between a candidate router and routers in range).

Changes in the network topology, which are due to router joins or leaves, are flooded through the network by means of messages called *link state advertisements (LSA).* They are flooded opportunistically, i.e., LSAs are not acknowledged. As a consequence, it is possible that the topoDB or the scanTabDB of two routers differ. The purpose of the database conflict resolution protocol is to make the topoDB and the ScanTabDB of a parent router and its child routers equal: the protocol resolves inconsistencies in topoDBs *and* scanTabDB (see Section 5). To avoid mutual exclusion issues, the topoDB and scanTabDB are solely accessed and updated by a dedicated process. The router initiating the hello protocol can verify whether its neighboring routers are alive.

2.2 Changes in network topology

Initially, the TOWN network consists of one router only, which is connected to the wired network. This master gateway is identified by configuration. Routers and other gateways *join* the network one-by-one, first gateways, then routers. Join decisions are based on SNR measurements by all join candidate routers stored in the scanTabDB and the network topology stored in the topoDB. The join process (see also Section 6) is initiated by the join candidate router and goes as follows (Fig. 2): The *join candidate router* scans and creates a list, the scanTabDB, which contains all routers in range including the downlink receiver SNRs on the links to these routers. Then the join candidate router performs a temporary association to the router with the most favorable SNR to flood the join request including scan table throughout the network. The router managing the cell of this temporary association is the *temporary parent router* of the join candidate in the spanning tree. The temporary parent router then waits until no new join requests or link state advertisements arrive during a predefined latency before it employs its own topoDB and the received scan tables from all candidate routers to feed the modified Dijkstra topology construction algorithm. This algorithm subsequently determines (i) whether this candidate router is the next to join the network and (ii) which is the permanent association/permanent parent router for this router. If the candidate router is the next to join the network, the temporary parent informs the join candidates where to associate. Subsequently, after a consistency check of the topoDB, the join candidate router associates with its permanent parent. The parent can then notify the TOWN network of the new link by flooding a link state advertisement of type ADD.

Sometimes a router *leaves* the TOWN network, either *gracefully* (i.e., intentionally) or *ungracefully.* If a router wants to leave the network gracefully, it sends a message of type of BYE to both its parent router and its

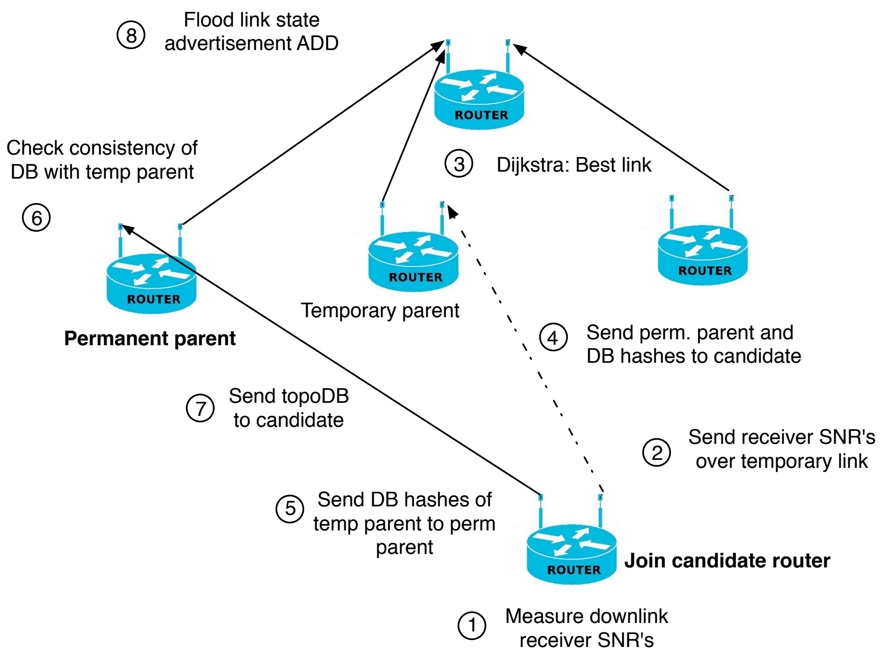
8 Flood link state advertisement ADD
ROUTER
Check consistency of DB with temp parent
3 Dijkstra: Best link
6
ROUTER
ROUTER
ROUTER
Temporary parent
Permanent parent
4 Send perm. parent and DB hashes to candidate
7 Send topoDB to candidate
2 Send receiver SNR's over temporary link
5 Send DB hashes of temp parent to perm parent
ROUTER
Join candidate router
1 Measure downlink receiver SNR's

Figure 2: Overview of a router joining the network.

child routers. Ungraceful leaves are detected by TOWN routers by running the hello protocol.

Each router is responsible to manage the cell(s) opened up by its base station(s): a link state advertisement is always initiated by the parent router of the link. To identify the temporal sequence of topology changes in the cell, i.e. ADDs or DELs of links, an LSA contains a *sequence number*, which is specific to the cell. At each time a router sends an LSA for a link to a child router, the router increments its counter representing the current sequence number.

3 Topology databases

Each router keeps a topoDB and a scanTabDB to feed the modified Dijkstra algorithm. Both topoDB and scanTabDB are lists of network links. A link consists of the following fields:

$$ID_{parent}, ID_{child}, seqNr, type, age, linkQual, channel$$

ID_{parent} and ID_{child} are each the MAC addresses of the network interface which is set to BS mode of the parent and child router, respectively.[1] $SeqNr$

[1]The ID of the master gateway router corresponds to the lexically smaller MAC-Addr of the two network interfaces.

is the sequence number as issued by the router that manages the cell with its base station interface. This numbering enables to identify temporal sequences of topology changes on a cell level. *Age* is the local age of the entry. It enables to remove old entries. The field *type* either has value ADD, DEL, or POT. If the latest entry of a link in the database is of type ADD, then the link is part of the network; if the latest entry of a link is of type DEL, then it has been removed. A link of type POT represents a potential link. *LinkQual* is the link quality as derived from the downlink receiver SNR measurement. *Channel* indicates the used channel of an ADD link.

4 Message flooding

Link state advertisements are *flooded unreliably* through the network, we call this type of flooding *opportunistic flooding*. We presume that opportunistic flooding combined with frequent consistency checks and a lean protocol to remove inconsistencies are more natural with IEEE 802.16-2004 equipment than reliable flooding. The message to be flooded is (a) cell broadcasted to all child routers, and (b) forwarded to the parent router. All children routers forward the message to their child routers. The parent router employs a *cell broadcast* to forward the message to all other children in the cell and forwards the message to its parent router (until the master gateway is reached). Replaying the same message is suppressed by tracking sequence numbers. Since forwarding and cell broadcasting are unreliable, topoDBs or scanTabDBs of may not be up to date.

5 The database conflict resolution protocol

To detect inequalities of DBs lying on two neighboring routers, we propose to run a database conflict resolution protocol on a cell level and on a regular basis. The protocol is run separately both for the topoDB and the scanTabDB. The protocol goes as follows (see Fig. 3): each parent router managing a cell broadcasts a *hash value* of the corresponding DB to its child routers by means of a `DBHash-Response` message. To dissolve hash collisions, we propose to include a random number in the hash. The hash value of the DB permits to check the equality of two DBs: the child applies the hash function to its DB and checks whether the received hash values corresponds to the computed hash values. If the hash values are unequal, the child router solicits the corresponding DB from its parent (sends a `DB-Request` to the parent). As response, the parent broadcasts the DB to its child routers (broadcasts `DB-Response`). Every time a child router receives a DB, it resolves conflicts between this DB and its local copy and sends back those links that are missing in the received DB back. Fig. 4 shows the DB conflict resolution protocol from the perspective of a child

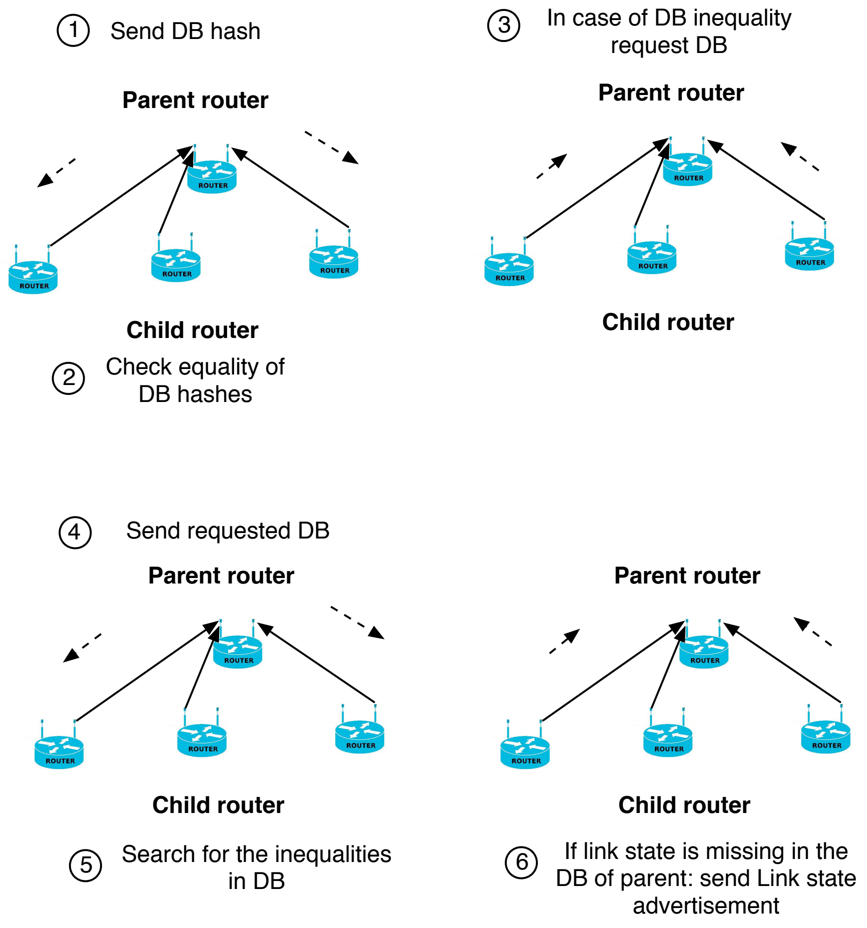
1 Send DB hash
Parent router
Child router
2 Check equality of DB hashes
3 In case of DB inequality request DB
Parent router
Child router
4 Send requested DB
Parent router
Child router
5 Search for the inequalities in DB
Parent router
Child router
6 If link state is missing in the DB of parent: send Link state advertisement
ROUTER

Figure 3: The database conflict resolution protocol.

router. Note that we do not depict the DB conflict resolution from the view of the parent, it is simple: the parent has just to reply to `DB-Request` and to process `LSA` messages. In the state transition diagrams we use a notation as given in Appendix C.

A conflict occurs if a link is in one DB but not in the other DB, or if the link is in both DBs but with different sequence numbers. The conflict resolution between two links is performed solely based on sequence numbers, i.e. for each cell, the link entry with the higher sequence number wins. If the parent DB misses some entries (i.e., the link information is correct on the child router but not up to date on the parent router), the child sends an `LSA`-message to the parent. To simplify this conflict resolution protocol we propose to add the size of the DB.

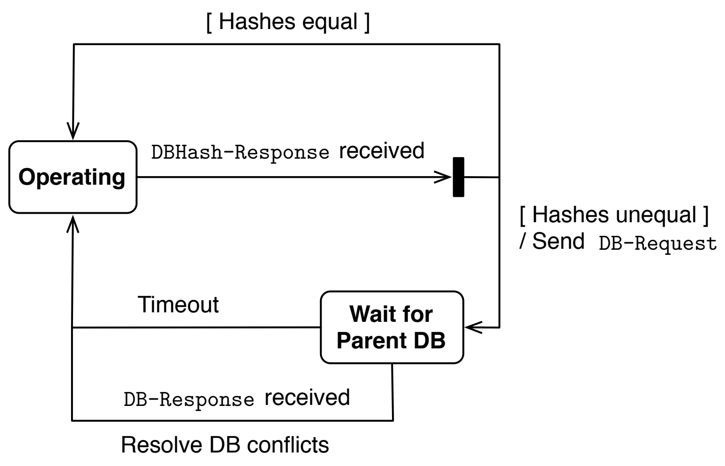

Figure 4: The state transition diagram for the conflict resolution protocol, from the perspective of a child router.

6 The join protocol

The join process of a candidate router has three major perspectives: the perspective of the join candidate router, the temporary parent, and the permanent parent. Each of the perspectives is described in one of the three subsequent subsections.

6.1 The join candidate router's perspective

The join candidate router first scans all channels and checks whether it senses activity (see Fig. 5). It measures the downlink receiver SNR of all routers in range. Then the join candidate router associates temporarily to the router with the most favorable SNR. Using this association the join candidate router "injects" its scan measurement results into the network. For each measured potential link, the join candidate router sends a link advertisement message of type POT. Moreover, the join candidate router sends a message of type `Invite-Request` (request for invitation). This message contains the number of routers in range of the join candidate router.

The join candidate router then waits for an `Invite-Response` from the temporary parent. The `Invite-Response` message includes which join candidate router is next to join the network and to which router this router has to associate. If the temporary parent and the permanent parent are the same, the join candidate router sends a `DB-Request` for both the topoDB and the scanTabDB.

If the temporary parent and the permanent parent are not the same, the join candidate router learns from its temporary parent the actual hashes

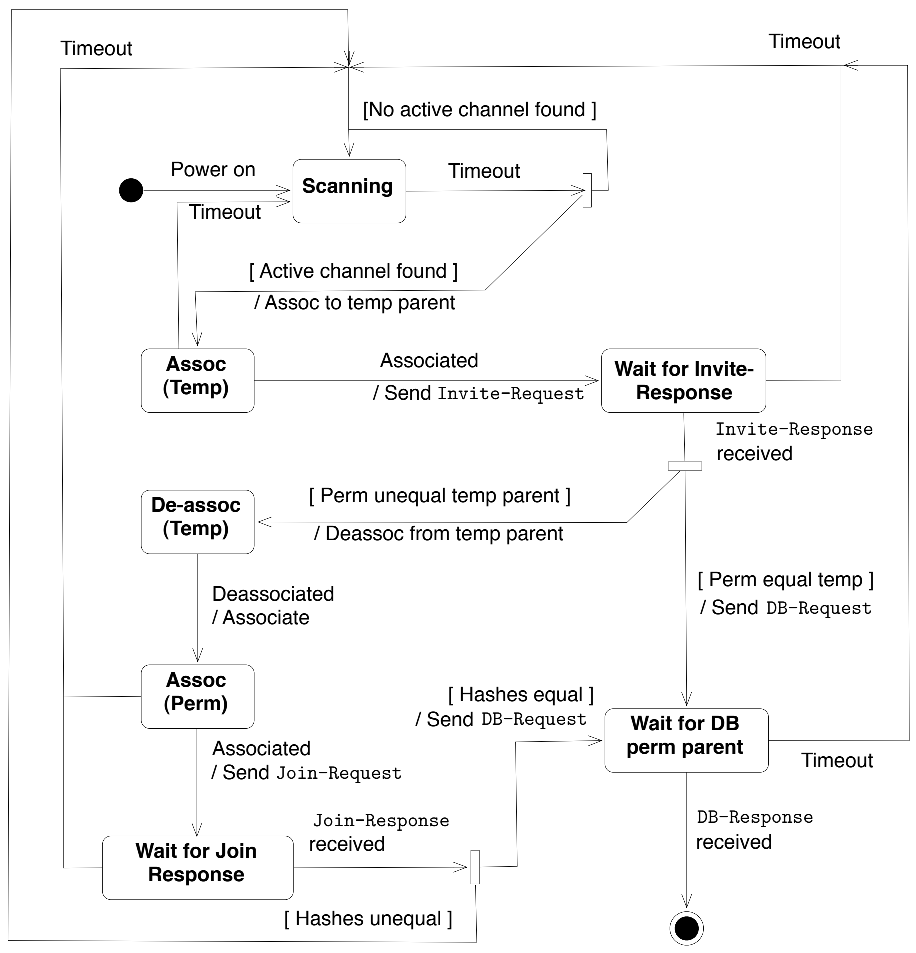

Figure 5: The state diagram for a router joining the network, from the perspective of the candidate router joining the network.

of the topoDB and the scanTabDB. Then the join candidate router sends a `Join-Request` message to the permanent parent. The child waits for `Join-Response`. If the permanent parent tells the join candidate router that the parent's DBs are equal to the DBs of the temporary parent, then the join candidate router solicits a `DB-Request` message. Finally, the join candidate router sets its second network interface to BS mode.

6.2 The temporary parent router's perspective

After receiving a `Invite-Request`-message, the temporary parent router floods scan measurement results injected by the join candidate router: for each potential link received, the temporary parent floods an `LSA`-message of type POT throughout the network. Moreover, the temporary parent router

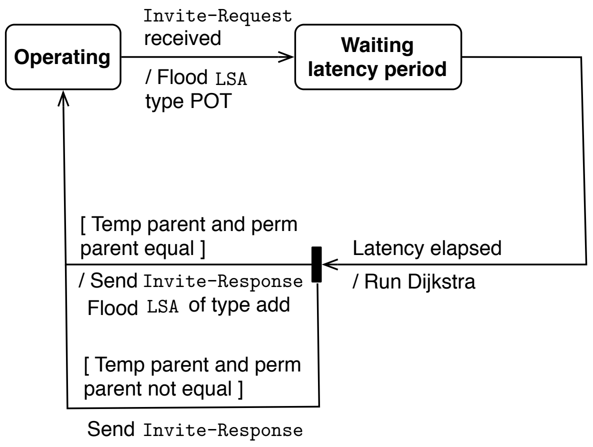

Figure 6: The state diagram for a router joining the network, from the perspective of the temporary parent router.

runs the modified Dijkstra algorithm after waiting for a latency period ("give flooding a chance"). The result of the run is forwarded to the join candidate by the `Invite-Response` message: contains association information plus the hash values of the topoDB and the hash value of the scanTabDB.

6.3 The permanent parent router's perspective

After receiving a `Join-Request` message, the permanent parent router compares the two hash values to check the consistency of topoDB and scanTabDB at the temporary and at the permanent router and accepts or rejects the association request. The result is send by means of an `Invite-Response` message. Upon accept, the permanent router floods an `LSA`-message of type ADD through the TOWN network. We note that the latency period before running the modified Dijkstra algorithm need to be extended upon retry ("wait until it gets calm on the net").

7 The leave protocol

We have to differentiate between two different ways of routers leaving the network: *graceful leave* of routers (i.e., it is in the intention of the network operator that the router leaves the network) and routers leaving the network *ungracefully* due to link failures or router crashes.

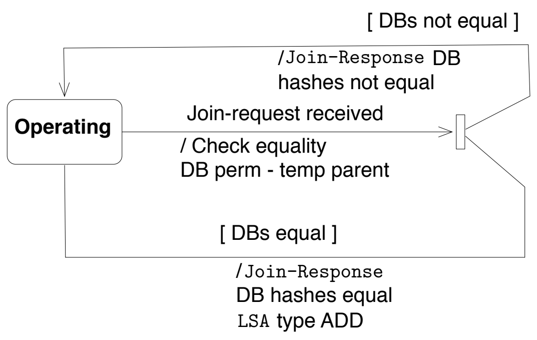
[DBs not equal]
/Join-Response DB hashes not equal
Operating
Join-request received
/ Check equality DB perm - temp parent
[DBs equal]
/Join-Response DB hashes equal LSA type ADD

Figure 7: The state diagram for a router joining the network, from the perspective of the permanent parent router.

7.1 Graceful leaves of routers

In investigated scenarios, less than 2% of the routers have more than two children routers with a configuration as desired by ASCOM [**?**]. Thus, no special handling of subtrees topologies that remain after graceful leave of a single router is required. Sending an `LSA` of type DEL to the parent, disassociating from the link to the parent, cell-broadcasting a `Bye-Message` to all children, flushing the topoDB and scanTabDB, and starting a new join process suffices (see Fig 8).

7.2 Link failures, router crashes, and the hello protocol

A router is not able to distinguish whether a parent left the network due to a link failure or due to a router crash. Therefore we handle these two cases equally. Each router temporarily checks whether there is connectivity to its parents and to all children under its management. To detect link failures, we use a simple hello protocol: A missing acknowledgment to a `Hello-Message` indicates that the link is dead with high probability. Note that if a router does not receive a response from its neighbor it repeats the message to ascertain the the link is indeed dead.

If the connectivity to the parent is lost, the router cell-broadcasts a `Bye-Message` to its children, flushes its topoDB and scanTabDB, and starts a new join process. Note that `Bye-Messages` are not acknowledged. If the connectivity to a child is lost, the router sends a `Del-Message` to its parent and cell-broadcasts the `Del-message` to the remaining child routers.

8 Format and handling of messages

In this section we list for all message types their *scope* (i.e., at which occasions a router exactly sends a message), their format (i.e., fields of message),

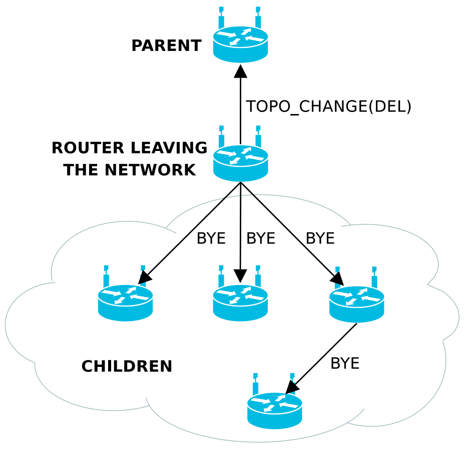

Figure 8: The graceful leave process. The router leaving the network sends an `LSA` of type DEL to its parent, closes the link to the parent, and then floods a `BYE-Message` to its children.

and the corresponding message handling (i.e., how a client receiving the message has to handle it). Note that common to all messages are the following message fields: SENDER_ID and RECEIVER_ID, as well as a field for the message type and a sequence number for the message.

8.1 `Invite-Request`

Scope Used by a join candidate router to inform a temporary parent router that the join candidate router wants to join the network.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const INVITE-REQUEST	The type of this message is Invite-Request
SCAN_SEQ	Sequence number for this message
NOF_BSS	No. of routers in range of join candidate router

Handling of receiver Run topology construction and channel allocation algorithm to find best link. Send the result in an `Invite-Response`.

8.2 `Invite-Response`

Scope Used by the temporary parent to inform the candidate router which join candidate router should join next, which is the permanent parent, as well as the hash values of topoDB and scanTabDB.

Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const INVITE-RESPONSE	The type of this message is JOIN-RESPONSE
INVITE_SEQ	Sequence number for this message
NEW-ROUTER	Which router can connect to the network
PERM-PARENT	The optimal parent router
CHANNEL	Which channel to use
HASH-topoDB	Hash value of topoDB of sender
HASH-scanTabDB	Hash value of scanTabDB of sender

Handling of receiver If the router receiving `Invite-Response` is the next to join, check whether temporary parent router and the permanent parent are the same. If yes, send `DB-Request` for topoDB and scanTabDB to temporary parent. If no, establish an association with the permanent parent and send a `Join-Request` to permanent parent.

8.3 `Join-Request`

Scope As soon as the candidate router gets an `Invite-Response` from the temporary parent, it sends a `Join-Request` to the permanent parent (if the permanent parent is not the same as the temporary parent). The message contains the hash values of the topoDB and the scanTabDb of the temporary parent.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const JOIN-REQUEST	The type of this message is JOIN-RESPONSE
JOIN_SEQ	Sequence number for this message
HASH-topoDB	Hash value of topoDB of sender
HASH-scanTabDB	Hash value of scanTabDB of sender

Handling of receiver Parent router sends a `Join-Response`.

8.4 `Join-Response`

Scope Inform the join candidate router whether topoDB and scanTabDB of permanent parent and temporary parent are equal.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const JOIN-RESPONSE	The type of this message is JOIN-RESPONSE
JOIN_SEQ	Sequence number for this message
ASSOCIATE	Child can permanently associate or not.

Handling of receiver If child router should not join, restart the join process. Otherwise solicit topoDB and scanTabDB by two messages of type `DB-Request`.

8.5 `LSA`

Scope A link state advertisement. Inform target router that router has joined the network (ADD), or that a router has left the network (DEL), or that potential link has been detected (POT). The message is sent at the following stages of the protocol: (a) When a link has permanently been established, (b) when a router has left the network (gracefully or not), (c) when a potential link has been measured.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const LSA	Type of message is link state advertisement (LSA)
LSA_SEQ	Sequence number for this message
TYPE	ADD, DEL, or POT.
LINK-PARENT	The parent of the link.
LINK-CHILD	The child of the link.
LINK-QUAL	The quality of the link.
CHANNEL	The channel used in an added link.

Handling of receiver If message of type ADD, add link to topoDB. If message of type DEL, take a look in topoDB which links have to be set to DEL (it is possible that an entire subtree has left the network). If message of type POT, add link to scanTabDB.

8.6 `DBHash-Request`

Scope Request hash value (integrity value) of either topoDB or scanTabDB from receiver router. Not used in the current version of protocol. Used to top design off.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const TOPODBHASH-REQ	Type of message is TOPODBHASH-REQ
TYPE	Either of type topoDB or scanTabDB.

Handling of receiver Send `DBHash-Response`.

8.7 `DBHash-Response`

Scope Send hash of either topoDB or scanTabDB of sender router to receiver router. Is used as part of the database conflict resolution protocol.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const TOPODBHASH-RSP	Type of message is TOPODBHASH-RSP
TYPE	Either of type topoDB or scanTabDB.
HASH	Hash value of database.
RND	Random number to dissolve hash collisions.
NOF_LINKS	Number of links in the database.

Handling of receiver Check whether received hash value equals hash value of own DB. If not, reply by a `DB-Request`.

8.8 `DB-Request`

Scope Used (a) by a new router to get DBs of permanent parent router and (b) by a child router that detects an inequality in one of its DBs.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const TOPO_DB_REQ	Type of message is Topology Database Request
TOPO_DB_REQ_SEQ	Sequence number for this message
TYPE	Either of type topoDB and scanTabDB.

Handling of receiver Send `DB-Response`.

8.9 `DB-Response`

Scope Send after reception of `DB-Request`.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const TOPO_DB	The type of this message is Topology Database
TOPO_DB_SEQ	Sequence number for this message
NOF_LINKS	Number of links in DB
TYPE	Either of type topoDB and scanTabDB.
PARENT_MAC	The parent router in the relationship
CHILD_MAC	The child router in the relationship
LQM	The link quality between the two routers

In each Topology Database message there are as many 3-tuples {PARENT_MAC, CHILD_MAC, SNR} as specified by the NOF_RELATIONS field.
Handling of receiver Look for inequalities to own DB. If there is an inequality, send `LSA` back.

8.10 `Hello-Request`

Scope A `Hello-Message` is employed on a periodical basis to ensure that a link between two neighboring routers is up (i.e. transmission over that link is possible).
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const HELLO_Req	The type of this message is hello request
Hello_SEQ	Sequence number for this message

Handling of receiver Send `Hello-Response`.

8.11 `Hello-Response`

Scope `Hello-Response` is employed to indicate that the requested router is alive.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const HELLO_RSP	The type of this message is hello response
Hello_SEQ	Sequence number of hello request message

Handling of receiver If `Hello-Response` is missing, router has to flood `LSA` of type DEL.

8.12 `Bye-Message`

Scope. `Bye-Message` is used to inform a router that it should leave the network gracefully.
Format

Field	Description
SENDER_ID	Identifies the sender of this message
RECEIVER_ID	Identifies the receiver of this message
const BYE	The type of this message is bye
BYE_SEQ	Sequence number for this message

Handling of receiver. If `BYE-Message` does not come from parent: flood `LSA` of type DEL to the parent and close the link to the parent. In any case: cell-broadcast a `BYE-Message` to all your children, flush the topoDB and scanTabDB, and start a new join process.

9 Demonstrator-related issues

We plan to implement the protocol in a test environment called the demonstrator, prior to the porting to ASCOM hardware (i.e., to the ASCOM routers). The purpose of building this demonstrator is to provide a proof-of-concept for the protocol. In this section, we describe the main components of the protocol implementation on the demonstrator. A conceptual overview of the implementation is given in Fig. 9.

The demonstrator is hosted by a single PC and by the virtualization software VMWare. In this VMWare-environment, each router of a TOWN network is represented by a corresponding virtual machine, on which the same implementation of the protocol runs. The implementation consists of one part that has to be ported from the demonstrator to the ASCOM hardware, as well as one part that needs not to be ported (as explained later).

The part of the implementation to be ported is indicated in green color in Fig 9. Its core is the implementation of the "logical steps" of the protocol.

This component of the implementation is the same for the virtual machines and the ASCOM routers, and it runs in their respective management units.

The protocol implementation makes use of two WiMax network devices and a set of communication primitives such as broadcasting of messages on a cell level. To ease portability of the protocol implementation, we define a common interface for the network device and the communication primitives. The network device interface implements mainly the following functions:

- Functions for turning a device on and off,
- and functions for setting an interface to base station or subscriber station mode.

The concrete header functions in a C-like notation can be found in Appendix D. The communication interface consists of routines for sending (unicast, broadcast) and receiving messages, see Appendix E.

One of our intention when designing our demonstrator is to set up a test environment which is as close as possible to a real TOWN network consisting of ASCOM routers. To enable reproducible testing, we introduce an emulation driver that handles events external and internal to the TOWN network. An external event is a router being switched on or off. Internal events are triggered by external events by messages being received. We thus need to emulate message forwarding over the TOWN network.

The emulation driver unit is implemented by a process that runs on a separate, dedicated virtual machine in the demonstrator. To ease debugging, both system-external and system-internal events are specified by corresponding configuration files.

The file configuring the system-internal events has specific entries for messages that are sent, lost, or replicated. In the demonstrator, when one router sends a message to another, the message is actually send to the emulation driver unit to be delayed, lost, duplicated. The emulation driver unit handles and forwards the messages to the recipient as configured.

The file configuring the system-external events has entries for starting and stopping routers (i.e., the virtual machines). To ease debugging, we define specific entries for the configuration file allowing to print or load a topoDB. The emulation driver unit is responsible for sending the system-external event message to the routers as configured. These event messages are sent via a dedicated communication channel, to increase the portability of the implementation. The virtual machine receiving the external event message is responsible for taking the appropriate action.

10 Summary

In this report, we have discussed a protocol for TOWN network operation and the network topology construction. The protocol is based on a previ-

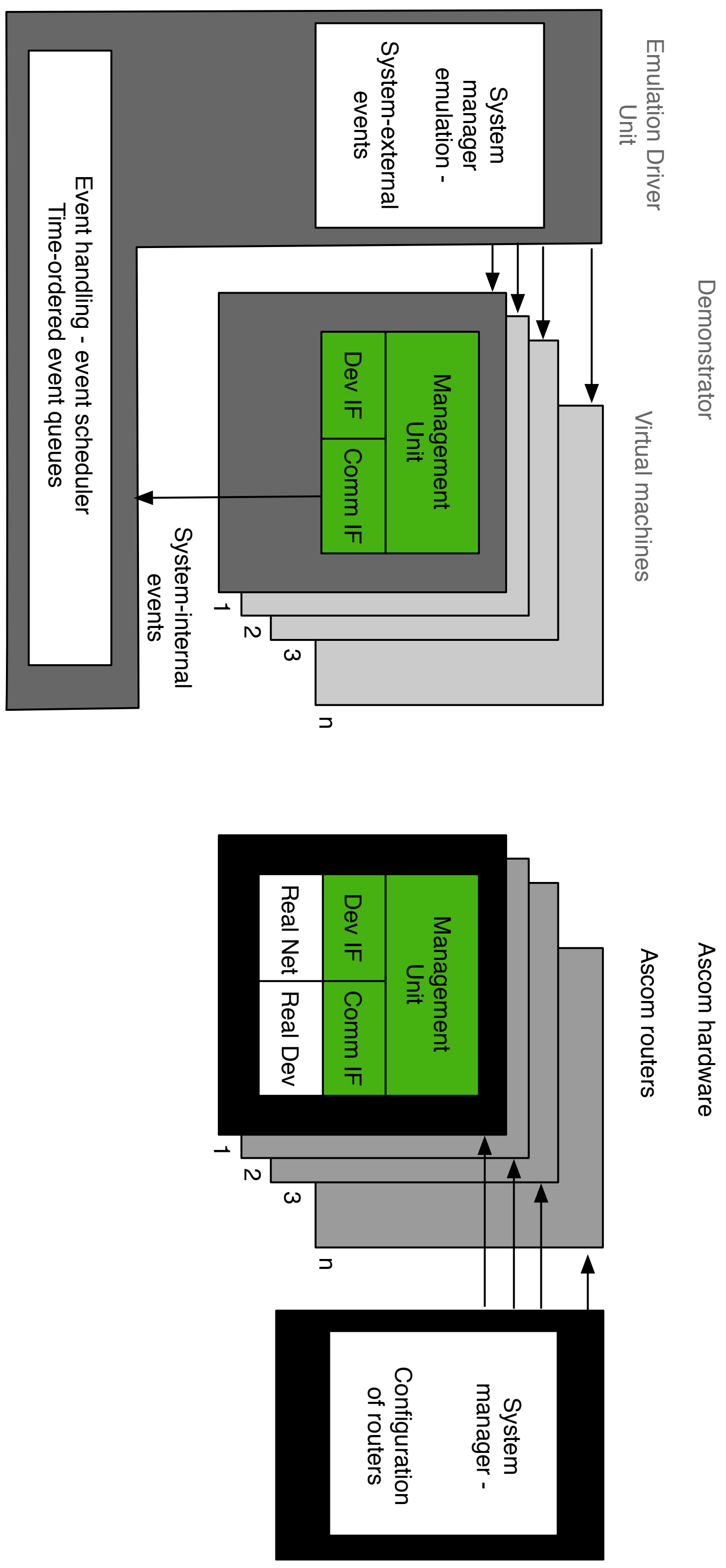

Figure 9: Comparison of the architecture of the TOWN demonstrator and of the ASCOM hardware.

ously proposed network topology construction and channel allocation algorithm. The design of the protocol starts from the idea that each router keeps both a list of all links in the current TOWN network (topoDB) and a list of potential links (scanTabDB). The overall protocol is split into four disjoint and simple protocols: a database conflict resolution protocol, a hello protocol, a router join and a router leave protocol. We have identified a dozen different message types that are sent within the protocol and have discussed how to handle them. Now we head towards implementing the protocol on the demonstrator in a way that makes porting the implementation to ASCOM hardware simple. One focus of future work is testing the software on the demonstrator to provide a proof-of-concept for the protocol. Finally, the project partners will port the software to ASCOM hardware.

References

[1] U. Fiedler, E. Glatz, Self-organized Topology Construction and Channel Allocation for Multi-IEEE 802.16-2004 Radio Routers in Disaster Recovery, in Proceedings of the 1st workshop on WiMAX, wireless, and mobility, Coimbra, Portugal, 2007.

[2] The QualNet Wireless Network Simulation, www.scalable-networks.com

[3] J. Moy, OSPF Version 2, Internet RFC 2328, 1998

[4] Ch. Huitema, Routing in the Internet, 2nd edition, Prentice Hall, ISBN 0-13-022647-5, 2007.

[5] J. Corbet, A. Rubini, G. Kroah-Hartman, Linux Device Drivers, 3rd Edition, O'Reilly, ISBN 0-596-00590-3, 2005.

[6] F. Steck, Effects of Churn on TOWN Network Topologies, Semester Thesis, Bern University of Applied Sciences, 2007.

A General project perspective

Mid August	M0: Design complete (report)
End October	M1: DTC runs on demonstrator
Mid December	M2: DTC runs on ASCOM HW
Mid February	Pietro Cerutti leaves project
End February	M3: End of TOWN project

Table 1: General project perspective

Mx (x=0,1,2,3) are the milestones to be kept to keep pace with the TOWN project proposal.

B Implementation plan

Communication environment	abstracted network driver primitives (cell broadcast, ...)	2 weeks
Network abstraction	table driven message transfer	1.5 weeks
Emulation driver modul	script and master config file	1 week
Regular operation I	message flooding	0.5 weeks
Regular operation II	build topoDB, scanTabDB read/writeDBs to/from files	0.5 weeks
Regular operation III	DB conflict resolution protocol	1 week
Regular operation IV	run Dijkstra on DBs	0.5 weeks
Test regular operation		0.5 weeks
Join I	implement use of network abstraction primitives for join	0.5 weeks
Join II	implement use of flooding, use of Dijkstra after wait, communicate Dijkstra result to candidate router, flood topo change	1 week
Test Join	construct topologies	1 week
Leaves and crashes	Hello protocol handling of Bye msgs	1 week
Test leaves and crashes		1 week

Table 2: Implementation plan

Each substep has to be tested and validated separately.

C Notation for state machine diagrams

The notation for the state machine diagrams in this report follow the UML 2.0 standard. Each transition is labeled with a string matching the pattern "[guard] trigger / activity".

- A *guard* describes a condition which must be satisfied for the transition to occur.
- A *trigger* describes the event causing the transition.
- An *activity* describes an action to be taken while transiting from one state to another.

D Network driver abstraction

The network driver abstraction consists of those function for preparing the operation of the (emulated) network device. More concretely, the abstraction has functions for turning the device on and off, for scanning for available networks, for connecting it to (disconnecting from) another wireless access point, as well as for setting the network device to BS mode. The corresponding C header file has the following functions defined:

```
struct wrp_dev_handle;
```

Identifies device we want to perform operations on.

```
struct wrp_dev_handle wrp_dev_open(const char * devname);
```

Open the device which we wish to configure. The device name is `eth0`, `eth1` etc. The handle returned must be passed to all other calls listed below.

```
void wrp_dev_close (const struct wrp_dev_handle *handle);
```

Close the device and free any resources allocated by the wrapper. After this call `handle` is no longer valid. This function cannot fail.

```
struct wrp_dev_scan_result *
wrp_dev_scan(const struct wrp_dev_handle * handle);
```

Trigger a scan and return a list of devices which we found.

```
int wrp_dev_join(const struct wrp_dev_handle * handle, struct ether_addr
AP_addr, const char essid[MAX_ESSID_SIZE+1]);
```

Join to a given AP as identified by its MAC address `ether_addr` and `ESSID`. If necessary, this function will change the operational mode of the device into a SS so that it can perform the join.

```
void wrp_dev_leave(const struct wrp_dev_handle * handle);
```

Leave the current AP. This function cannot fail.

```
int wrp_dev_bs(const struct wrp_dev_handle * handle, const double
freq const char essid[MAX_ESSID_SIZE+1]);
```

Operate as a base station using the `ESSID` and the channel as specified by `freq`.

texttt`void wrp_dev_down(const struct wrp_dev_handle * handle);`

Put the device down. It will no longer send or receive packets.

`void wrp_dev_up(const struct wrp_dev_handle * handle);`

Put the device up. It will now be able to send and receive packets, as well as perform scans.

E Communication interface

The communication interface consists of functions allowing to send messages (either by unicast or broadcast). The C header file of the communication interface is the following:

`struct wrp_comm_handle;`

This handle is passed to identify which device we want to send to or receive from.

`struct wrp_comm_handle * wrp_comm_open(const char * devname);`

Open a device (`eth0, eth1,` etc.) so that we can send and receive messages.

`void wrp_comm_close (struct wrp_comm_handle * const handle);`

Close a device and free any resources allocated by the wrapper. After this call the handle is no longer valid. This function cannot fail.

```
void wrp_comm_local(const struct wrp_comm_handle * handle, struct
ether_addr *local);
```

Return the local MAC address of the interface.

```
int wrp_comm_send(const struct wrp_comm_handle * handle, const struct
ether_addr dst, const uint8_t * data, const size_t size);
```

Send a message out to the device a specified by `dst`, which can either be a unicast MAC address or the MAC broadcast address. The data to be sent is pointed by `data`, the data is `size` bytes long.

```
int wrp_comm_recv(const struct wrp_comm_handle * const handle, struct
ether_addr *src, uint8_t * data, size_t size);
```

Locking receive on device `handle`. The pointer `src` is a MAC address. The argument `data` is a pointer to a buffer where the data received from `src` is to be stored. The buffer should be able to store up to `size` bytes. If the received message is longer than `size`, the message will be truncated.

```
int wrp_comm_fd(const struct wrp_comm_handle * const handle);
```

Return a file descriptor for a given `handle`, which can be used to determine if there is data to be read from the device.