Design and implementation of a DECT stack for Linux

Patrick McHardy <kaber@trash.net>

Linux Kongress 2010, Nürnberg

http://dect.osmocom.org/

About

This presentation will present a DECT stack for Linux, containing all components starting from baseband drivers up to an asterisk channel driver.

Started around January 2009 after fixing a few bugs in the dedected.org software and realizing how far away from a full implementation it was.

Development was primarily motivated by the desire to fool around with this quite old technology and see how buggy other implementations really are :)

About

Initially based on the reverse engineered RX-only Com-on-Air PCMCIA driver from the dedected.org project, but basically fully rewritten since then.

Mainly written by myself, except for the DSAA (DECT Standard Authentication Algorithm) and DSC (DECT Standard Cipher), which have both been reverse engineered and implemented by the dedected.org project, who also helped me a lot in getting TX support into the firmware.

What is DECT?

- DECT stands for “Digital Enhanced Cordless Telecommunications”, a short-range wireless communications standard used primarily for cordless telephony, but also for Data Terminals, Wireless payment terminals, Baby monitors, Door openers, Traffic Light control, Industrial control, ...

- Specified by ETSI in 1987, further enhanced since then.

- Quite similar to other wireless telecommunication standards like GSM or TETRA.

DECT standards

The DECT standards are split into two parts: DECT Common Interface (CI) contains the base standard, DECT application profiles define requirements for specific applications like telephony, packet radio service, etc. and build on the Common Interface. Even the base standard alone is huge and contains over 1200 pages.

- Common Interface: ETSI EN 300 175-1 – EN 300 175-8

DECT standards
Common Interface

- ETSI EN 300 175-1: Part 1: Overview
- ETSI EN 300 175-2: Part 2: Physical layer (PHL)
- ETSI EN 300 175-3: Part 3: Medium Access Control (MAC) layer
- ETSI EN 300 175-4: Part 4: Data Link Control (DLC) layer
- ETSI EN 300 175-5: Part 5: Network (NWK) layer
- ETSI EN 300 175-6: Part 6: Identities and addressing
- ETSI EN 300 175-7: Part 7: Security features
- ETSI EN 300 175-8: Part 8: Speech and audio coding and transmission

DECT standards
Profiles

This list contains a small excerpt of the most common profiles, there are many more:

- Public Access Profile (PAP): ETSI EN 300 175-9: obsoleted by GAP

- Generic Access Profile (GAP): ETSI EN 300 444

DECT standards
Profiles

- NG DECT Part 1: Wideband speech: ETSI TS 102 527-1

- NG DECT Part 2: transparent IP packet data: ETSI TS 102 527-2

- NG DECT Part 3: Extended wideband speech services: ETSI TS 102 527-3

- NG DECT Part 4: Light Data Services, Software Update Over The Air (SUOTA), content downloading and HTTP based applications: ETSI TS 102 527-4

DECT Common Interface

The DECT Common Interface contains the following parts:

- Part 1: Overview: overview of the system and protocol architecture, definition of terms

- Part 2: Physical layer (PHL): Specifies radio parameters (modulation, frequency, timing, power values), TDMA frame structure, packet formats, synchronization, primitives to higher layers.

DECT Common Interface

• Part 3: Medium Access Control (MAC) layer: specifies MAC services (broadcast message control (BMC), connectionless message control (CMC), traffic-bearer control (TBC), multi-bearer control (MBC)), definition of logical channels, various control messages, multiplexing and mapping to physical channels.

DECT Common Interface

• Part 4: Data Link Control (DLC) layer: specifies logical data links, C-plane (control plane) services, U-plane (user plane) services, MAC connection management

• Part 5: Network (NWK) layer: specifies functions for Link Control, Call Control, Mobility Management, Supplementary Services.

DECT Common Interface

• Part 6: Identities and addressing: specifies equipment related identities and their relationships.

• Part 7: Security Features: specifies authentication and ciphering processes, key types and key management, effect of the security features on the lower layers

• Part 8: Speech and audio coding and transmission: specified requirements for real-time speech and other audio services

Basic terminology

- FP (Fixed Part): a DECT base station
- PP (Portable Part): a DECT telephone, terminal, ...

Physical layer
Radio spectrum

- DECT operates in the 1880 MHz -1980 MHz band (Europe, Asia, Australia, South America) and 1920 MHz – 1930 MHz band (U.S.). Other bands are defined.

- Carrier width 1.728 MHz, 10 standard carriers, further carriers (up to a total of 64) defined per band.

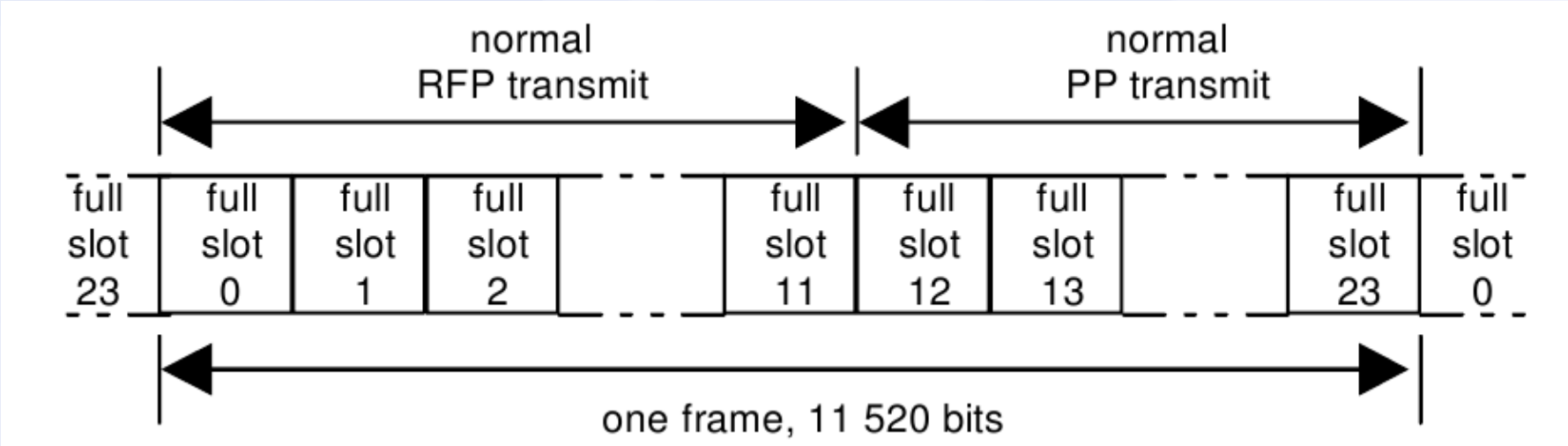
Physical layer
Modulation

DECT supports various modulation schemes:

- 2-level modulation mandatory, optionally up to 64-level modulation for higher data rates.
- 1.152 Mbit/s per carrier with 2-level modulation
- 6.912 Mbit/s per carrier with 64-level modulation

Physical layer
TDMA

DECT uses a TDMA frame structure for access in time: frame of 11520 symbols (bits), split into 24 slots of 480 symbols each, 100 frames per second



Physical layer
TDMA slots

TDMA slots may be used only partially or adjacent slots may be combined. Defined slot formats are:

- Full slot (480 symbols)
- Half slot (240 symbols)
- Double slot (960 symbols)
- Variable capacity slot j (100+j or 104+j symbols) with 0 <= j <= 856

Physical layer
Physical packet formats

For each slot format, a corresponding physical packet format exists:

- short physical packet P00 (96 bits)
- basic physical packet P32 (420 or 424 bits)
- high capacity physical packet P80 (900 or 904 bits)
- low capacity physical packet P00j (100+j or 104+j bits) with 0 <= j <= 856

Physical layer
Data fields

Physical packets contain multiple data fields with different purposes:

- S-field (synchronization field): 16 bits preamble, 16 bits synchronization word. Contained in all packet formats, used for clock and packet synchronization.

- D-field (data field): contains higher layer data, its size is dependent on the packet format.

Physical layer
Data fields

- Z-field: repeats the last 4 bits of the D-field (X-CRC), used to detect unsynchronized interference sliding into the end of the physical packet. The P00-Packet contains no Z-Field, in other packet formats its use is optional and defined by DECT profiles.

Physical layer
Physical channels

Physical channels provide a connectionless simplex service for data transmission and are created by transmitting physical packets on a particular RF-channel and a particular slot position in successive frames. TDD is used to create double simplex or duplex channels.

- Short physical channel R00: Packet P00
- Basic physical channel R32: Packet P32
- High capacity physical channel R80: Packet P80
- Variable rate physical channel R00j: Packet P00j

Physical layer
Timing

Besides providing means to transmit and receive packets, the physical layer is also responsible for providing timing to the higher layers. The FP provides timing to all PPs though the S-Field synchronization pulse and the TDMA frame structure. The MAC layer extends the TDMA slot and frame timing by superimposing a multiframe structure on the TDMA frame structure.

FPs containing multiple transceivers must have their transceivers synchronized with an accuracy of 15 +- 2us.

Medium Access Control layer

Due to the special requirements of a cellular wireless technology, like handling interference, performing various kinds of handover, privacy, combined with the strict timing requirements of a circuit-switched technology and the large number of diverse applications, the DECT MAC layer is the most complicated layer in the DECT stack.

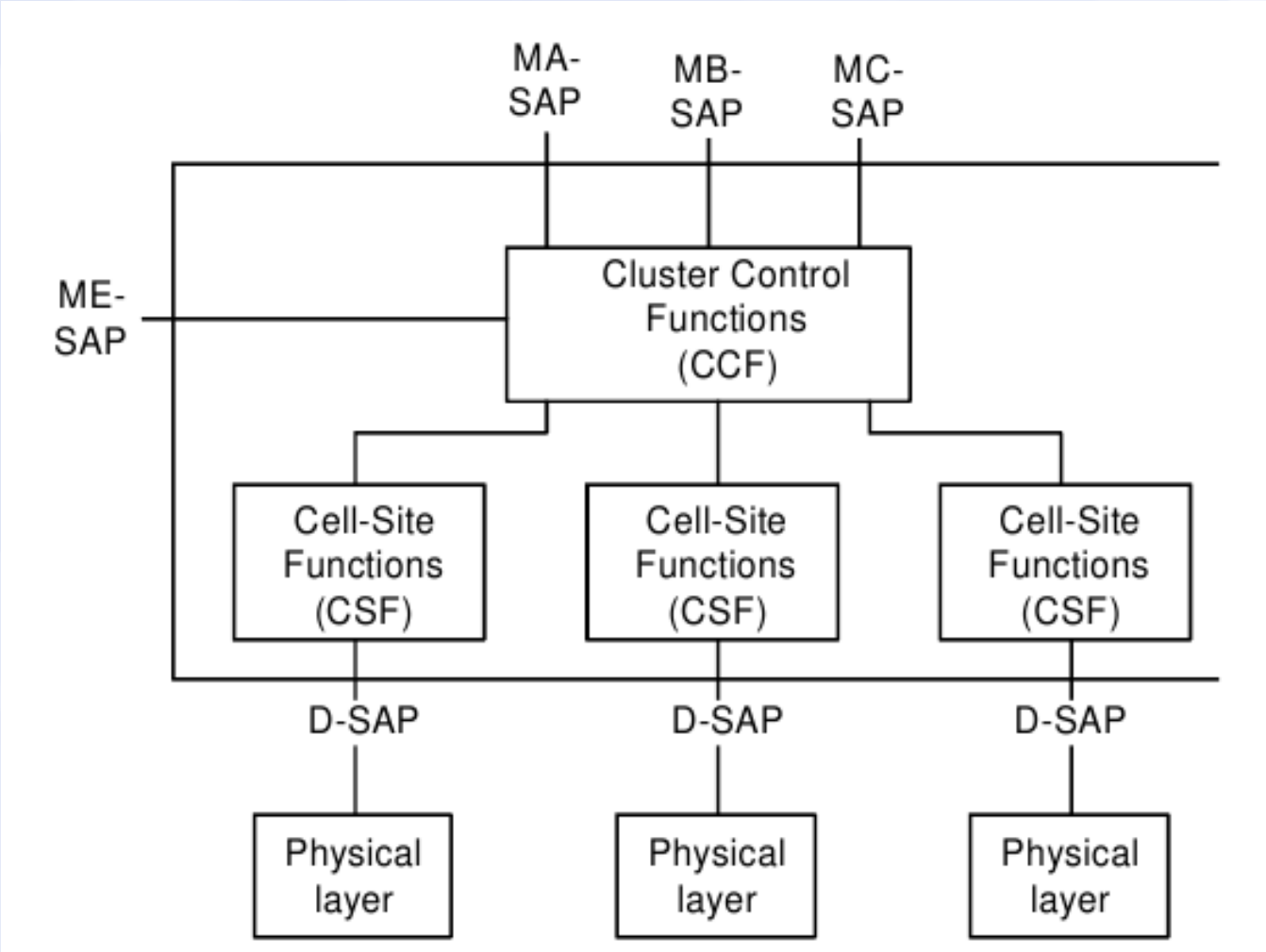
Medium Access Control layer
Reference model

Internally the MAC layer is split into two parts:

- Cell Site Functions (CSF) include all functions that are concerned with only one cell. Multiple instances of CSF may exist in multi-cell system. These instances would typically be located in different physical locations.

- Cluster Control Functions (CCF) functions are used to control more than one cell. A single instance of CCF exists in a DECT cluster. Communication between CSF and CCF is not specified.

Medium Access Control layer
Reference model



Medium Access Control layer
Services

The MAC layer provides three major groups of services to the higher layers:

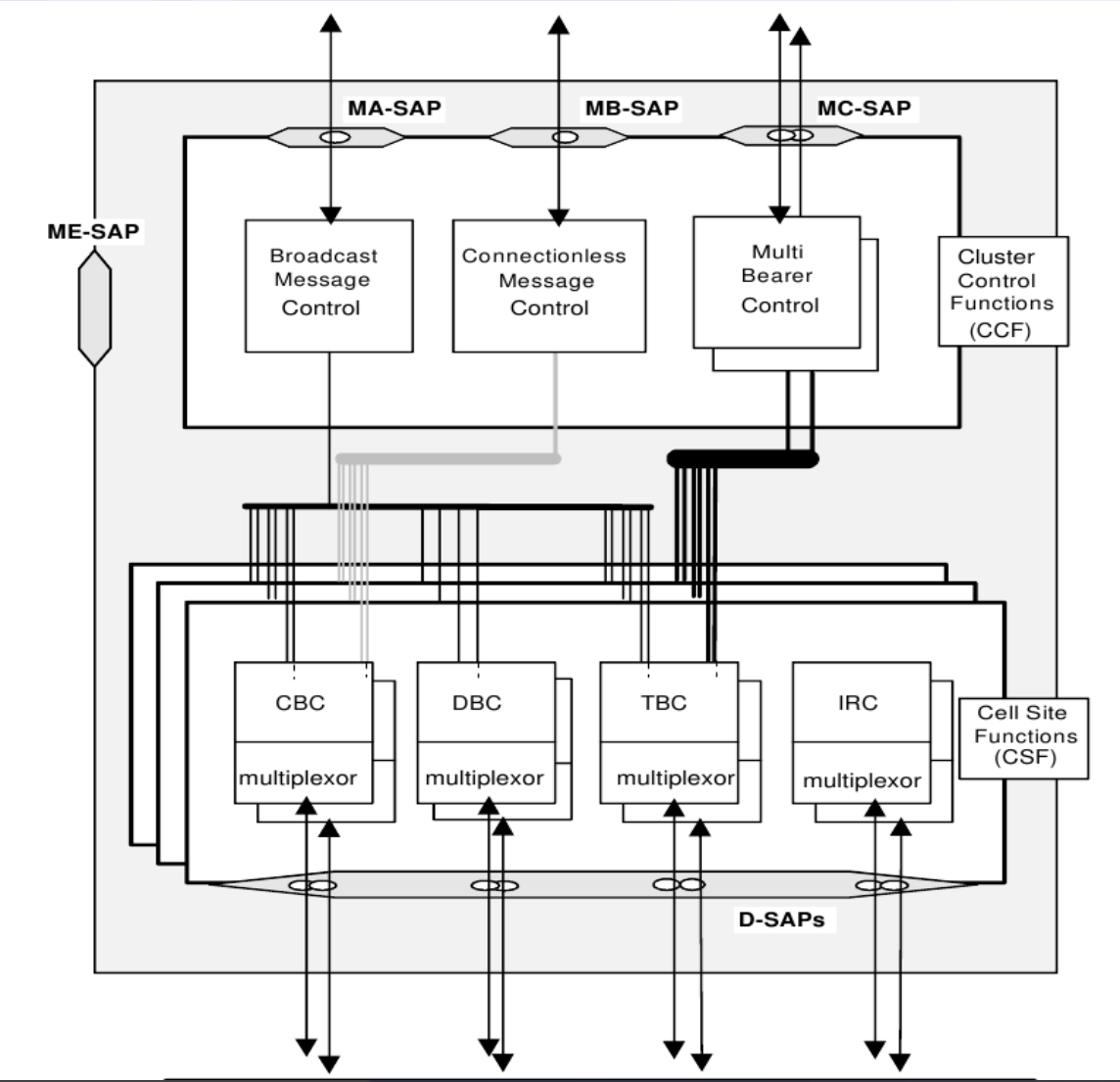
- Broadcast message control (BMC): broadcast service for point-to-multipoint communication in the direction FP->PP

- Connectionless message control (CMC): provides bi-directional point-to-point or point-to-multipoint services to the higher layers

Medium Access Control layer
Services

• Multi-Bearer control (MBC): provides connection oriented point-to-point service to the higher layers, using one or more traffic bearers. When using multiple bearers, data may be distributed for increased bandwidth, or duplicated for increased reliability.

Medium Access Control layer



Medium Access Control layer
Logical channels

These services are used to carry a number of logical data channels, which are multiplexed into physical packets (various other types and subtypes exist):

- C channels (C_S and C_F): higher layer C-plane channel, transmitted only on traffic bearers, use error correction based on Automatic Repeat Requests (ARQ)

- I channels (I_N and I_P): higher layer U-plane channel, transmitted only on traffic bearers

Medium Access Control layer
Logical channels

- B_s channel: slow broadcast channel in direction FP->PP, transmitted on all traffic, connectionless and dummy bearers

Medium Access Control layer
Logical channels

The MAC layer also contains a number of internal control channels:

• Q channel: system information channel: low-rate channel transmitted on all bearers to provide PPs with system information, some of which is needed to lock to an FP.

• N channel: Identities channel: transmitted on all bearers in both directions, provides PPs with the FP identity and access rights information for locking to an FP, used for a MAC layer handshake in the direction PP->FP on traffic bearers

Medium Access Control layer
Logical channels

• M channel: MAC control channel: transports point-to-point MAC layer information, used for bearer setup and handover, encryption activation, service attribute and bandwidth negotiation, retrieving channel lists

• P channel: MAC paging channel: carries either higher layer paging messages or MAC internal pages to supply MAC layer information to PPs, like bearer channel switches, supported carriers, blind transceiver slots, etc. Higher layer information is transmitted on all bearer types, MAC internal information in some cases only on specific bearers.

Medium Access Control layer
Cell Site Functions

The Cell Site Functions (CSF) contain:

• Connectionless Bearer Control (CBC): functions that control one connectionless bearer. Multiple instances of CBC may exist.

Medium Access Control layer
Cell Site Functions

• Dummy Bearer Control (DBC): functions that control one dummy bearer. The dummy bearer is responsible for transmitting the continuous broadcast service, which is used to transmit system and timing information necessary for identifying and locking to a cell (Q- and N-channel). In the absence of traffic bearers one or two instances of DBC may exist, when traffic bearers are active they perform the continuous broadcast service. A DBC controls one simplex bearer.

Medium Access Control layer
Cell Site Functions

• Traffic Bearer Control (TBC): functions that control one traffic bearer. Traffic bearers provide continuous point-to-point transmissions for communicating with other systems. A TBC controls one duplex or double simplex bearer. Traffic Bearer Control itself is controlled by a Multi-Bearer Control instance in the Cluster Control Functions. Multiple instances of TBC may exist.

Medium Access Control layer
Cell Site Functions

• Idle Receiver Control (IRC): functions that control the receiver when idle. IRC is responsible for scanning for bearer setup attempts and maintaining channel lists for channel selection. One instance of IRC per receiver exists, controlling all idle slots.

• Multiplexing and demultiplexing of logical channels to data fields of physical packets: a complex set of rules determines when and where to place data in physical packets.

Medium Access Control layer
Cluster Control Functions

The Cluster Control Functions (CCF) functions contain:

- Broadcast Message Control (BMC): functions that control and distribute the cluster's broadcast information from/to all CBCs, TBCs and DBCs. One instance of BMC exists per CCF.

- Connectionless Message Control (CMC): functions that control and distribute the cluster's connectionless services to to one or more CBCs. At most one instance of CMC exists per CCF.

Medium Access Control layer
Cluster Control Functions

• Multi-Bearer Control (MBC): functions that control the multiplexing and management of data associated with a MAC connection. A MBC manages one or more (in case of handover or multi-bearer connections) TBCs. One instance of MBC exists per MAC connection.

Data Link Control layer

The Data Link Control layer offers two independent types of service:

• DLC C-plane: Control Plane, provides a reliable point-to-point or point-to-multipoint service for transporting error-protected Network layer signalling information. Additionally a broadcast service is provided.

• DLC U-plane: User Plane, provides an end-to-end transport for user information. Several independent circuit-mode and packet-mode services exist.

Data Link Control layer

Additionally the DLC layer is responsible for opening and closing MAC connections in response to service demands and routing DLC frames to and from the available MAC connections.

Data Link Control layer
C-plane

C-plane data link services are provided by two entities:

• LAPC entity: the higher LAPC entity provides a protocol for logical link establishment, frame transmission with optional flow control and retransmissions. The LAPC protocol is derived from the ISDN LAPD protocol, but differs in some aspects. Each LAPC entity is paired with a lower Lc entity.

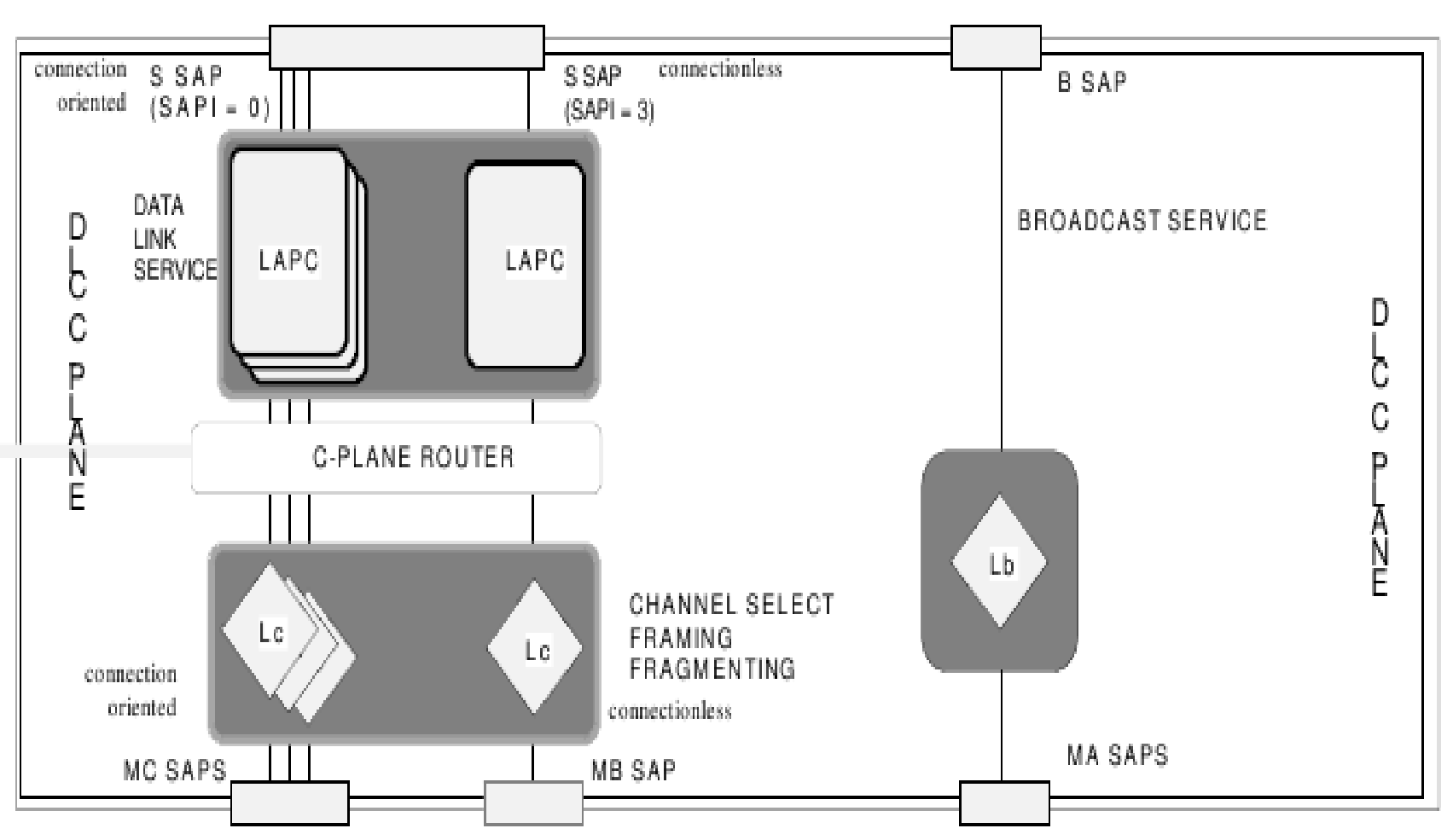
Data Link Control layer
C-plane

• Lc entity: the lower Lc entity is responsible for selecting the logical MAC channel for frame transmission (C_S or C_F), buffering and fragmenting complete LAPC frames from/to the MAC layer as well as performing checksum generation and validation. One Lc entity per MAC connection exists.

Data Link Control layer
C-plane

The DLC broadcast service is provided by the Lb entity. The Lb entity is responsible for buffering and forwarding higher layer messages from/to the MAC layer, distributing and transmitting messages over different clusters and coallition and filtering of received messages from different clusters. The main application of the DLC broadcast service is to transmit and receive paging messages.

Data Link Control layer
C-plane



Data Link Control layer
U-plane

Each U-Plane service is divided into two entities: an upper LUx entity and a lower FBx entity. The upper LUx entity contains all service dependent functions, the lower FBx entity is responsible for buffering and framing the U-Plane frames to/from the MAC layer.

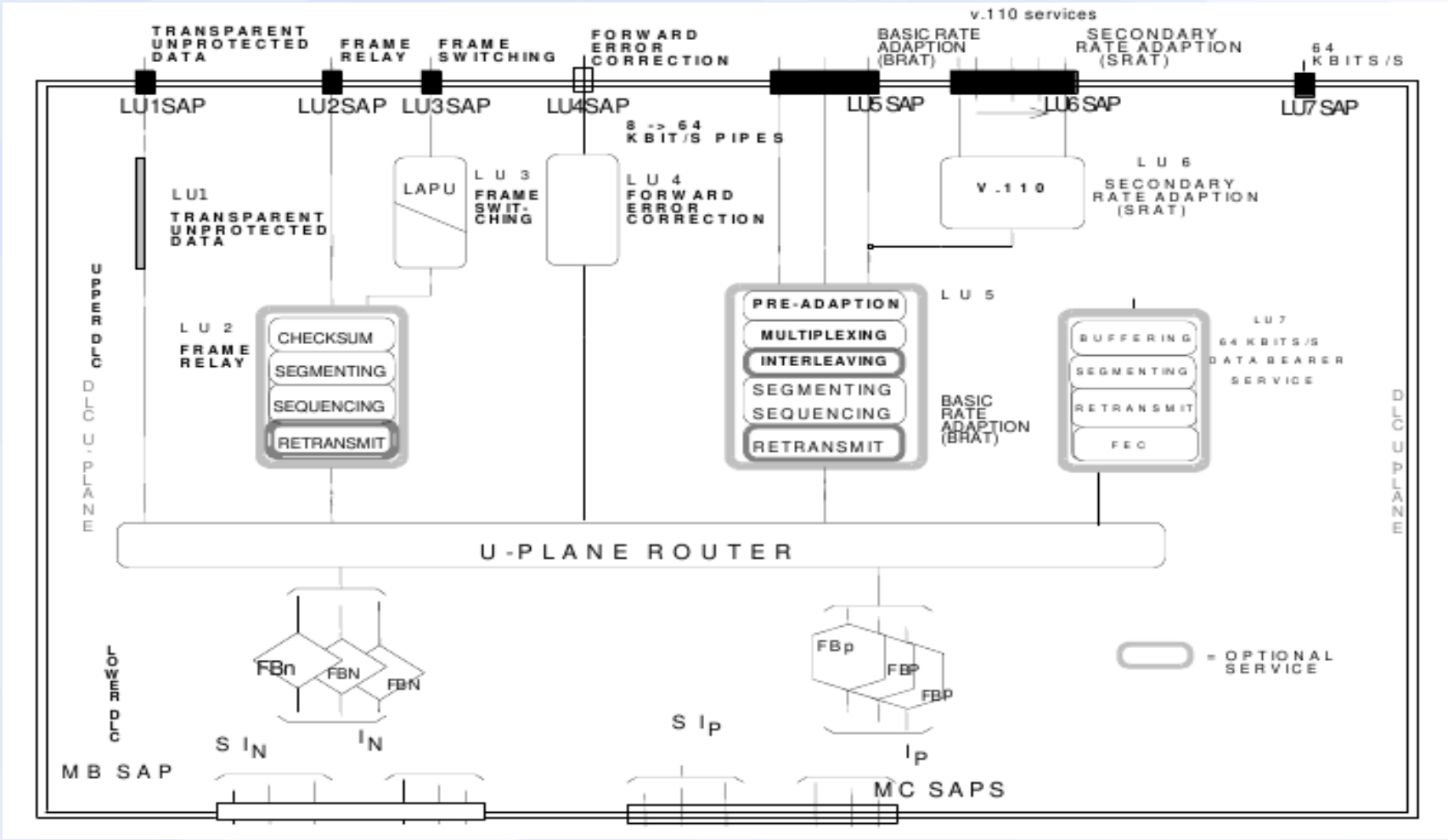
Data Link Control layer
U-plane

The commonly used LUx entities for voice services are:

- LU1 (TRUP – TRansparent UnProtected service): used for narrow band ADPCM G.726 32 kbit/s voice service

- LU12 (UNF – UNprotected Framed service): used for wideband G.722 64 kbit/s voice service

Data Link Control layer
U-plane



Network layer

The Network layer contains the following functions:

• LCE – Link Control entity: establishment, operation and release of C-plane links.

• CC – Call Control entity: establishment, operation and release of circuit switched calls

• CISS – Call Independent Supplementary Services entity: support of call independent supplementary services

Network layer

- COMS – Call Oriented Message Service entity: support of connection-oriented messages

- CLMS – ConnectionLess Message Service entity: support of connectionless messages

- MM – Mobility Management entity: management of identities, authentication, ciphering, location updates, key allocation

Network layer

Most Network layer services utilize a TLV encoded message format termed S-Format messages. The two exceptions are the paging service and the CLMS-Fixed service, which use a fixed frame structure called B-Format messages.

Network layer
S-Format messages

S-Format messages begin with a fixed header containing a transaction identifier to distinguish multiple parallel transactions between the same higher layer entities, a protocol discriminator to identify the higher layer entity and a message type.

This is followed by zero or more Information Elements. Most Information Elements use a TLV encoding, a few fixed length elements exist as well. Each message type defines a set of allowed and mandatory Information Elements for both directions.

Network layer
S-Format messages

S-Format message structure:

		Octet:
Transaction Identifier	Protocol Discriminator	1
Extended Transaction Value		1a
Message Type		2
Information elements		3
		N

Network layer
Link Control Entity

The Link Control entity is responsible for the following tasks:

- Link establishment and maintenance in response to higher layer service demands

- Routing of new higher layer protocol endpoints to existing data links

- Submitting (FP) and receiving (PP) paging messages

Network layer
Link Control Entity

- Uplink and downlink routing of higher layer messages based on transaction identifiers and protocol discriminators

Network layer
Call Control

The Call Control entity is responsible for establishment, maintenance and release of circuit switched calls. It optionally performs service negotiation during call setup and service modification for established calls. Each call is associated with one or more U-plane service instances.

Network layer
Mobility Management

The Mobility Management entity provides services necessary for secure provision of DECT services. Among these are:

• Access rights procedures: used to “pair” the PP with the FP. Usually combined with key allocation.

• Key allocation procedure: over-the-air key allocation, initially converts a 4-digit authentication code into a 128 bit authentication key, later on can be used to replace the authentication key.

Network layer
Mobility Management

- Authentication: authentication of the FP, the PP or the user

- Ciphering: switching of the ciphering state of a MAC connection

- Location registration: informs the FP about the location of the PP, can be used to route direct setup attempts from the FP to the correct cell

Network layer
Mobility Management

- Identity assigment: assignment of a temporary identity used for paging or a network assigned identity

- Identification: request of specific identification parameters from the PP

- Parameter retrieval: exchange of information, f.i. external handover information

Implementation
Overview

The implementation is split between user-space and the kernel at the DLC layer. The Physical layer, the MAC layer and the DLC layer, which all have strict timing requirements, are contained in the kernel. The DLC layer offers a socket API to user-space

Additionally a netlink family for management operations, a raw socket family for receiving and transmitting raw frames and a TIPC based network protocol for communication between the CCF and CSF exist in the kernel.

Implementation
Overview

In user-space a libnl based library (“libnl-dect”) offers access to the netlink management protocol and a separate library (“libdect”) implements the NWK layer, as well as some auxiliary functions. An example IWU is implemented as asterisk channel driver.

Implementation
Physical layer

The physical layer functionality is provided by a common transceiver layer as well as a driver for the SC14421 and SC14424 DECT baseband processors (Com-on-Air PCI/PCMCIA). Additionally a virtual transceiver driver for testing purposes exists.

Drivers register a structure describing their capabilities and containing a set of callback functions with the common transceiver layer. The main task of a driver is to receive and transmit frames and perform signal strength measurements.

Implementation
Physical layer

```
struct dect_transceiver_ops {
        void            (*disable)(const struct dect_transceiver *trx);
        void            (*enable)(const struct dect_transceiver *trx);

        void            (*confirm)(const struct dect_transceiver *trx);
        void            (*unlock)(const struct dect_transceiver *trx);
        void            (*lock)(const struct dect_transceiver *trx, u8 slot);

        void            (*set_mode)(const struct dect_transceiver *trx,
                                    const struct dect_channel_desc *chd,
                                    enum dect_slot_states mode);
        void            (*set_carrier)(const struct dect_transceiver *trx,
                                       u8 slot, u8 carrier);
        void            (*tx)(const struct dect_transceiver *trx,
                              struct sk_buff *skb);

        u64             (*set_band)(const struct dect_transceiver *trx,
                                    const struct dect_band *band);
        void            (*destructor)(struct dect_transceiver *trx);
        const char      *name;

        u32             slotmask;
        u8              eventrate;
        u8              latency;
};
```

Implementation
Physical layer

Drivers process multiple consecutive slots at once and queue a structure describing all events which occurred during these slots to the common transceiver layer.

```
struct dect_transceiver_event {
        struct dect_transceiver     *trx;
        atomic_t                    busy;
        struct list_head            list;
        struct sk_buff_head         rx_queue;
        u8                          rssi[DECT_HALF_FRAME_SIZE / 2];
        u8                          rssi_mask;
        u8                          slotpos;
};
```

Due to MAC layer timing requirements, at most 6 slots may be processed in one batch.

Implementation
Physical layer

The main task of the common transceiver layer is to process the event batches queued by the drivers in softirq context and pass them to the MAC layer and provide two virtual clock sources for RX and TX.

The TX clock leads the RX clock by a static amount of slots since the baseband processor must be programmed for upcoming slots before its internal clock actually reaches those slots. The RX clock, which is advanced based on events queued by the transceiver, is by definition in the past.

Implementation
Physical layer

Multiple transceivers can be joined in “transceiver groups”, the event batches queued for each group are sorted chronologically before they are replayed to the MAC layer.

Clock synchronization among the transceivers is achieved over the air by locking secondary transceivers to the signal of the primary one. A simple mechanism to detect and ignore transceivers which have lost synchronization exists.

Implementation
Physical layer

Additionally the common transceiver layer performs some book-keeping functions to keep track of the state of a transceiver (locked/unlocked) and individual slots (available/in use) and offers a few simple helper functions for packet transmission, transceiver maintenance as well as slot and carrier calculations.

Implementation
Physical layer

Transceivers export some state and statistics through the netlink API, which can be viewed in userspace using an example program from libnl.

In this example the transceiver is running as tertiary transceiver attached to a FP (visible by the <sync> indication) and is handling one ciphered MAC connection. Slots 12-22 perform the FP's tertiary scan for setup attempts.

dect-transceiver-list –name trx2

Implementation
Physical layer

```
DECT Transceiver trx2@cell0:
        Type: sc1442x
        RF-band: 00000
        Events: busy: 0 late: 970

        slot 0: <tx,cipher> carrier: 2 (1893.888 MHz)
            RX: bytes 296 packets 37 a-crc-errors 8 x-crc-errors 0 z-crc-errors 0
            TX: bytes 4464 packets 93
        slot 2: <idle> carrier: 0 (1897.344 MHz)
            RX: bytes 0 packets 0 a-crc-errors 0 x-crc-errors 0 z-crc-errors 0
            TX: bytes 0 packets 0
        slot 4: <rx,sync> carrier: 9 (1881.792 MHz +0.896 kHz) signal level: -38.88dBm
            RX: bytes 5752 packets 719 a-crc-errors 63 x-crc-errors 0 z-crc-errors 0
            TX: bytes 0 packets 0
        ...
        slot 10: <idle> carrier: 6 (1886.976 MHz)
            RX: bytes 0 packets 0 a-crc-errors 0 x-crc-errors 0 z-crc-errors 0
            TX: bytes 288 packets 6
        slot 12: <rx,cipher> carrier: 2 (1893.888 MHz +0.216 kHz) signal level: -46.65dBm
            RX: bytes 6308 packets 142 a-crc-errors 37 x-crc-errors 35 z-crc-errors 36
            TX: bytes 0 packets 0
        slot 14: <scanning> carrier: 0 (1897.344 MHz)
            RX: bytes 3528 packets 72 a-crc-errors 33 x-crc-errors 31 z-crc-errors 28
            TX: bytes 0 packets 0
        ...
        slot 22: <scanning> carrier: 0 (1897.344 MHz)
            RX: bytes 1960 packets 40 a-crc-errors 40 x-crc-errors 38 z-crc-errors 37
            TX: bytes 0 packets 0
```

Implementation
Physical layer

The driver for the SC14421/14424 baseband processors supports both FP and PP mode, ciphering, scrambling and checksum calculations in hardware, as well as RSSI and phase offset measurements. Support for radio chips is modularized, depending on the radio in use, different RF-bands are supported either partially or completely. The firmware is available as source code and can be compiled during the build process using the ASL macro assembler.

The PCI version additionally features blinking antennas :)

Implementation
Physical layer

The virtual transceiver driver internally creates “transceiver groups”. Transceivers in the same group are able to communicate with each other. Groups and transceivers can be added or removed through a sysfs interface.

Each transceiver has an assigned position and power level, which can be changed through the sysfs interface and which is used to calculate the signal strength at the receiver. This is intended for handover testing, the implementation is not complete yet however.

Implementation
MAC layer

From the lower side, the MAC layer's cell site's function are driven by the transceiver layer, from which it received frames, signal strength measurements and timing information. As specified by the standard, the CSF implements Traffic Bearers, Dummy Bearer Control, Idle Receiver Control etc.

One part not specified by the standard are “monitor bearers”, which are basically passive (RX-only) traffic bearers for monitoring two-way communication between a PP and a FP.

Implementation
MAC layer

Communication between the CSF and the CCF is implemented through “handles”, which contain a set of function pointers, which can either be invoked directly, or through a TIPC based network protocol.

The TIPC protocol encodes all parameters in network packets, the receiving side decodes them again and invokes the primitive.

Implementation
MAC layer

```
struct dect_csf_ops {
        int     (*set_mode)(const struct dect_cell_handle *,
                            enum dect_cluster_modes);
        int     (*scan)(const struct dect_cell_handle *,
                        const struct dect_llme_req *lreq,
                        const struct dect_ari *, const struct dect_ari *);
        int     (*preload)(const struct dect_cell_handle *,
                           const struct dect_ari *, u8,
                           const struct dect_si *);
        int     (*enable)(const struct dect_cell_handle *);

        void    (*page_req)(const struct dect_cell_handle *, struct sk_buff *);

        int     (*tbc_establish_req)(const struct dect_cell_handle *,
                                     const struct dect_tbc_id *,
                                     const struct dect_channel_desc *,
                                     enum dect_mac_service_types, bool);
        int     (*tbc_establish_res)(const struct dect_cell_handle *,
                                     const struct dect_tbc_id *);
        void    (*tbc_dis_req)(const struct dect_cell_handle *,
                               const struct dect_tbc_id *,
        ,,,
```

Implementation
MAC layer

```
struct dect_ccf_ops {
        int     (*bind)(struct dect_cluster_handle *,
                        struct dect_cell_handle *);
        void    (*unbind)(struct dect_cluster_handle *,
                          struct dect_cell_handle *);

        void    (*time_ind)(struct dect_cluster_handle *,
                            enum dect_timer_bases, u32, u8, u8);

        void    (*scan_report)(const struct dect_cluster_handle *,
                               const struct dect_scan_result *);
        void    (*mac_info_ind)(const struct dect_cluster_handle *,
                                const struct dect_idi *,
                                const struct dect_si *);

        int     (*tbc_establish_ind)(const struct dect_cluster_handle *,
                                     const struct dect_cell_handle *,
                                     const struct dect_tbc_id *,
                                     enum dect_mac_service_types, bool);
        int     (*tbc_establish_cfm)(const struct dect_cluster_handle *,
                                      const struct dect_tbc_id *, bool, u8);
        ...
```

Implementation
MAC layer

This indirection is intended for building distributed multi-cell system, but the network protocol hasn't been actually tested yet.

The CCF implement Multi-Bearer Control, Bearer handover, control and monitoring of Traffic Bearer setup and configuration of the CSF. Currently only connections using a single traffic bearer are supported.

I- and C-channel data received from the CSF is passed to the higher DLC layer at frame boundaries or, in case of the minimal delay audio service, at slot boundaries.

Implementation
Data Link Control layer

The DLC layer contains a generic part interacting with the MAC layer and modules implementing the U-plane and C-plane services.

The generic part performs MAC connection management and up-/downlink routing of data between the MAC layer and the C-plane/U-plane based on the used logical data channel.

Implementation
Data Link Control layer

The C_S and C_F channel are routed from/to the C-plane, the I channel is routed to the U-plane.

New MAC connections are indicated by the MAC layer or requested from the MAC layer based on service demands. When service demands cease, the DLC layer releases MAC connections or, in case of release indicated by the MAC layer, notifies the higher layer entities.

Implementation
Data Link Control layer

The DLC C-plane consists of the Lc and LAPC entities and a socket API, providing SOCK_SEQPACKET sockets to userspace.

The socket API is in large parts similar to that of other connection oriented socket families, it supports sending and receiving packets, binding sockets to specific addresses identifying the FP/PP/connection instance, connecting and listening for new connections.

The main difference to other families are a set of DECT specific operation for handing ciphering.

Implementation
Data Link Control layer

The socket API contains two setsockopt options for setting the cipher key of the underlying MAC connection of a socket and requesting cipher activation/deactivation from the lower layers.

Cipher state change notifications from the lower layers are wrapped into messages and queued to sockets error queue. In addition to dequeuing packets from the normal receive queue, the recvmsg() function dequeues these indications from the error queue and supplies them to userspace in the ancillary message data (cmsgs).

Implementation
Data Link Control layer

In the future the setsockopt options will be extended to also support specifying lower layer service attributes or indicating achieved service attributes to userspace.

Implementation
Data Link Control layer

The generic part of the U-plane currently only supports the FBn entity, which is basically a NOP and only passes on data between the DLC and the higher LUx entities.

Of the higher U-plane entities, only the LU1 entity currently exists, which is used for the minimal delay G.726 audio service. To userspace the LU1 entity presents a standard stream socket API. It does not support opening of new MAC connections directly however, it can only be routed through existing MAC connections established through the C-plane.

Implementation
Data Link Control layer

One thing diverging from the behaviour of regular stream sockets is the support for seamless handover. During handover, multiple bearers operating in different time slots are supplying and requesting data from the socket.

The data delivered to the bearers depends on the exact point in time at which it is requested in order to let the receiver switch between the bearers without noticable interruptions. Data is therefore kept in a queue even after delivery to the DLC and removed from the queue at a rate corresponding to the transmission rate.

Implementation
Network layer

The NWK layer is implemented as a userspace library called “libdect”. The library currently supports Call Control (CC), Mobility Management (MM), Supplementary Services (SS) and the Connectionless Messaging Service (CLMS). Additionally it contains support for the authentication, key generation, some lower layer management operations and raw socket frame reception and transmission.

Implementation
Network layer

To use the library, the application needs to supply a set of callback functions for event and timer handling and a set of primitives for each subsystem it wishes to use.

The registered primitives correspond to the indication and confirmation primitives specified in ETSI EN 300 175-5, section 16.3 “Primitives to IWU”. The request and response primitives are exported by libdect as regular functions.

Implementation
Network layer

The application then opens a handle to a specific cluster by calling dect_open_handle(&ops, cluster_name) and waits for events (external or from libdect). In PP mode the application additionally needs to specify its identity by calling dect_pp_set_ipui(&ipui) before entering the event loop.

Implementation
Network layer

struct dect_ops contains references to the subsystem specific ops:

```
struct dect_ops {
        void                                *(*malloc)(size_t size);
        void                                (*free)(void *ptr);

        const struct dect_event_ops     *event_ops;
        const struct dect_llme_ops_     *llme_ops;
        const struct dect_lce_ops       *lce_ops;
        const struct dect_cc_ops        *cc_ops;
        const struct dect_mm_ops        *mm_ops;
        const struct dect_ss_ops        *ss_ops;
        const struct dect_clms_ops      *clms_ops;
        const struct dect_raw_ops       *raw_ops;
};
```

Implementation
Network layer

Example of subsystem specific (CC) ops:

```
struct dect_cc_ops {
        size_t  priv_size;
        void    (*mncc_setup_ind)(struct dect_handle *dh, struct dect_call *call,
                                  struct dect_mncc_setup_param *param);
        void    (*mncc_setup_ack_ind)(struct dect_handle *dh, struct dect_call *call,
                                      struct dect_mncc_setup_ack_param *param);

        void    (*mncc_reject_ind)(struct dect_handle *dh, struct dect_call *call,
                                   enum dect_causes cause,
                                   struct dect_mncc_release_param *param);

        void    (*mncc_call_proc_ind)(struct dect_handle *dh, struct dect_call *call,
                                      struct dect_mncc_call_proc_param *param);

        void    (*mncc_alert_ind)(struct dect_handle *dh, struct dect_call *call,
                                  struct dect_mncc_alert_param *param);

        void    (*mncc_connect_ind)(struct dect_handle *dh, struct dect_call *call,
                                    struct dect_mncc_connect_param *param);
        void    (*mncc_connect_cfm)(struct dect_handle *dh, struct dect_call *call,
                                    struct dect_mncc_connect_param *param);
```

Implementation
Network layer

The arguments to the individual primitives are usually a subsystem specific endpoint identifier, identifying an instance of the subsystem (“call”), and a structure encapsulating the information elements applicable to the primitive.

Implementation
Network layer

The encapsulating argument structures as well as its individual members are allocated by libdect and are reference counted. The application can keep references to either the entire collection of arguments or individual information elements.

This is useful when a procedure is interrupted, f.i. to perform authentication, to continue processing once the interrupting procedure has completed. The information elements can also be passed back to a different libdect primitive, f.i. when performing feature or codec negotiation. Arguments passed to libdect functions may also be allocated on the stack.

Implementation
Network layer

Example of an argument structure: each struct embeds a “struct dect_ie_collection”, on which common operations like reference counting can be performed.

```
struct dect_mncc_modify_param {
        struct dect_ie_collection               common;
        struct dect_ie_service_change_info      *service_change_info;
        struct dect_ie_list                     iwu_attributes;
        struct dect_ie_list                     iwu_to_iwu;
        struct dect_ie_escape_to_proprietary    *escape_to_proprietary;
};

struct dect_ie_collection {
        unsigned int                    refcnt;
        unsigned int                    size;
        struct dect_ie_common           *ie[];
};
```

Implementation
Network layer

Information elements contain the parsed or to-be constructed S-Format message contents. Similar to IE collections, each Information Element embeds a struct dect_ie_common for operations common to all Information elements (reference counting and list operations).

Values are usually defined as enums where applicable with names corresponding to those defined in the standard.

Example of an Information Element:

```
enum dect_setup_capabilities {
        DECT_SETUP_SELECTIVE_FAST_SETUP                         = 0x0,
        DECT_SETUP_NO_FAST_SETUP                                = 0x1,
        DECT_SETUP_COMPLETE_FAST_SETUP                          = 0x2,
        DECT_SETUP_COMPLETE_AND_SELECTIVE_FAST_SETUP  = 0x3,
};

/** <<SETUP-CAPABILITY>> IE */
struct dect_ie_setup_capability {
        struct dect_ie_common           common;
        enum dect_page_capabilities     page_capability;
        enum dect_setup_capabilities    setup_capability;
};

struct dect_ie_common {
        struct dect_ie_common  *next;
        unsigned int           refcnt;
};
```

Implementation
Network layer

Interaction with the library through primitives follows the procedures specified in the standard, not much to say except to look at the standard.

All primitives and many internal operations are instrumented with debugging code, messages are fully decoded during parsing and construction, usually easy to trace what's going on if something doesn't work.

Thanks

- Andreas Schuler and Matthias Wenzel from the dedected.org project their initial Com-on-Air driver, the reverse engineered DSC and DSAA algorithms and help with working on the SC1442x firmware

- Harald Welte for hosting the dect.osmocom.org website

- unnamed people from the DECT industry for helping me understand the SC1442x chips, help with the firmware and useful information