TDMA Version 2
Towards a Revised Media Access Control

University of Hannover

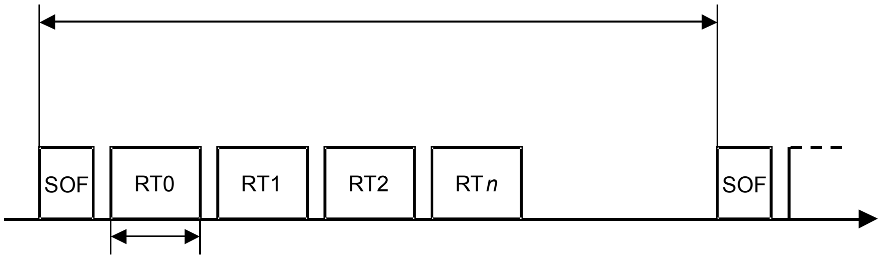
ISE – Real Time Systems Group

Outline

- Analysis of TDMA V1
- Additional Requirements
- Concepts
  - Configuration
  - Hot-plugging
  - Packet scheduling
- Discipline Interface (RTmac)
- Roadmap
- Discussion

TDMA V1 – The Current Situation

- Quite **stable, applicable implementation** for several releases

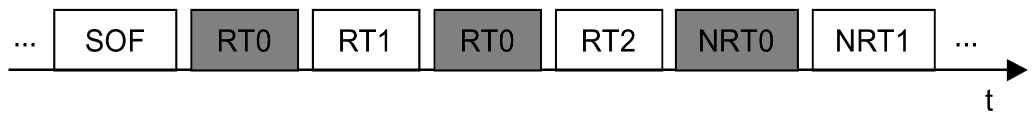


- Straight forward concept
  - Start of Frame issued periodically by a single master
  - Master and “clients” have each one payload transmission slot assigned
  - Outgoing payload frame selection based on local priorities
  - Master time contained in Start of Frame
  - IP-centric node configuration

Design Weaknesses

- Only fixed single slot per station and frame

  ➔ Freely assignable slot (offset, station, frequency, size)



- Configuration handshake is too unstable (under certain conditions) and too slow

  ➔ Define more robust handshake – or avoid it...

- IP orientation prevents IP-less RTnet

  ➔ Node identification shall use only MAC addresses, RTcfg can handle IP-to-MAC assignment

Further Weaknesses

- Undocumented state machine

  ➔ State machine as part of specification

- Unclean real-time/non-real-time interaction

  ➔ Use RTPC (Real-Time Procedure Call) mechanism

- No MTU enforcement

- Unhandy diagnosis interface

  ➔ Add **real-time-safe** /proc support, add IOCTLs

- Management tool still merged into rtifconfig

  ➔ Stand-alone tool ("rtmacconfig_tdma", "tdmaconfig", ?)

Additional Requirement

- Hot-plugging of preconfigured stations into a running real-time network
- Intelligent packet scheduling based on priority and **size**
- Fall-back master
- Sequence number in Start of Frame
- Improved time stamp precision (compensation of propagation time)
- Naming convention: “clients” should become “slaves” ☺

RTmac Discipline Interface

- Arbitrary disciplines can be registered with RTmac
- **`struct rtmac_disc {`**

```
    const char   *name;
    unsigned int priv_size;
    u16          disc_type;
    int          (*packet_rx)(...);
    int          (*rt_packet_tx/nrt_packet_tx)(...);
    int          (*attach/detach)(...);
    struct rtnet_ioctls ioctls;
    struct rtmac_proc_entry *proc_entries;
    ...
```

- Individual management interface is provided by specified IOCTLs via RTnet's misc device

RTmac Protocol Frame

- RTmac frame (as defined last year ;-))

32 bits

type	ver	res

  - **`type`**: **`ETH_TDMA`** (0x9031) – TDMA Version 1 frame
    other value – Encapsulated Ethernet frame
  - **`ver`**: 0x0001 – Version
  - **`res`**: reserved for future use

- Problem: Encapsulated non-real-time Ethernet frames may collide with discipline frame types

- Suggestion: Use **`res`** field to mark tunnelled frames
  `res` => **`tun`**, **`tun`** = 0: discipline frame, **`tun`** ≠ 0: tunnelled frame
  `ver` = 0x0002

RTcfg Mechanisms

- Generic protocol consisting of 3 stages
- Stage 1
  - Server invites expected participants
  - Also transmits required RTmac parameters (optional, not used with TDMA V1)
- Stage 2
  - Client sends identification message
  - Other clients reply reporting their addresses
  - Server delivers user defined configuration (optional)
  - Rendezvous point (used by current TDMA to start RT-mode)
- Stage 3 (optional)
  - Exchange ready notification between all stations

RTcfg Mechanisms (2)

- Start-up must not wait for all expected RTcfg clients, may proceed after timeout!
- Server can monitor active clients via heart beat mechanism
- Dead clients will be re-invited
- Attaching of new (or replaced) client automatically updates all ARP tables on running stations
- Management interface may provide information about client status (/proc entries, not yet implemented)

TDMA V2 – Configuration

- Parameters can be set by user mode tool

```
rtmacconfig_tdma <dev> master <cycle>
rtmacconfig_tdma <dev> fallback?
rtmacconfig_tdma <dev> slave
rtmacconfig_tdma <dev> slot <offs> <size> <freq>
rtmacconfig_tdma <dev> detach
```

- Parameters or configuration scripts will be distributed via RTcfg (stage 1)
- No configuration handshake at TDMA level
- On-the-fly changes of slot parameters shall be admissible

Hot-Plugging

- No start-up handshake –
  no need for common start procedure!
- Station start-up
  - Slave retrieves TDMA configuration via RTcfg,
    it does not transmit any packet yet!
  - Configuration is set by user mode tool (e.g. through a script)
  - Station waits for Start of Frame
  - Station sends packets in any assigned time slot
  - Slave can now actively finalise the RTcfg handshake (stage 2)
- Remember: RTcfg handles node failure and exchange
  - List of active stations
  - ARP table updates

Protocol Frames

- Do we need more than a (revised) Start of Frame?
- `struct tdma2_sof {`

```
    u32   frame_type;
    u32   frame_no;
    u64   time_stamp;
}
```

just in case we do need more...

- Frame number is incremented once per cycle
- Time stamp resolution is still 1 nanosecond
  (With hardware support, we may reach sub-microsecond precision some day...)

Packet Scheduling

- Scenario on some station:
  Slot 1, 300 µs offset, max. 200 bytes, every TDMA frame
  Slot 2, 500 µs offset, max. 1500 bytes, every 2nd TDMA frame

Packet 1, high priority, 1000 bytes	=> Slot 2
Packet 2, low priority, 100 bytes	=> Slot 1 or 2?
Packet 3, medium priority, 200 bytes	=> Slot 1 or 2?

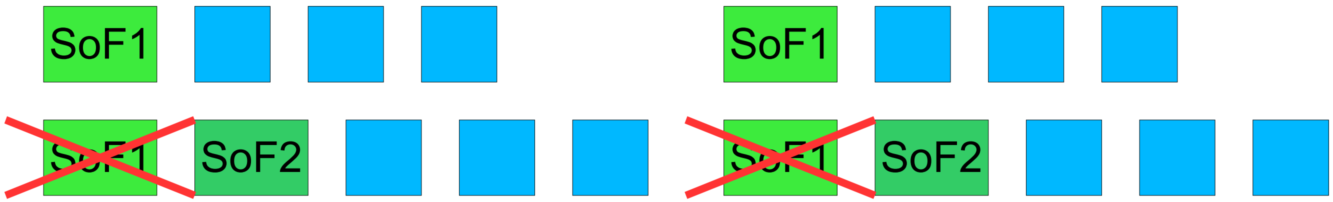
- Scheduling becomes much more complicated with multiple slots of different sizes!
- Schedule automatically based on size and priority? Or allow explicit slot selections by the application?
- Which MTU shall be reported to higher layers?
- Scheduling intelligence may increase worst case delay...

Packet Scheduling (2)

- *Approach A:* One priority queue per slot
  Benefits: - Enqueue packet according to required slot size.
  - Scheduling is performed at the cost of the sender.
  Drawbacks: - Packets may stall in overloaded large slot queues while smaller but still fitting queues remain unused.
  - Which slot shall be selected if several fit?
- *Approach B:* One queue for equally sized default slots, additional queues for other slots which are dedicated to selected applications (sockets) or services (VNIC, RTcfg, etc.)
  Benefits: - simple scheduling with few overhead
  - unambiguous MTU
  Drawbacks: - requires adapted applications and new tweaking parameters of RTnet components

Fall-Back Master

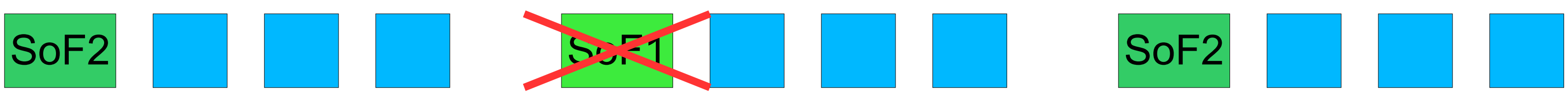
- *Approach A:* Secondary master takes over if primary fails



Benefits: - Simple implementation
- No modification and overhead on slave side

Drawbacks: - Failure detection and take-over delay increases worst-case packet transmission time

- *Approach B:* Both masters send SoF alternately

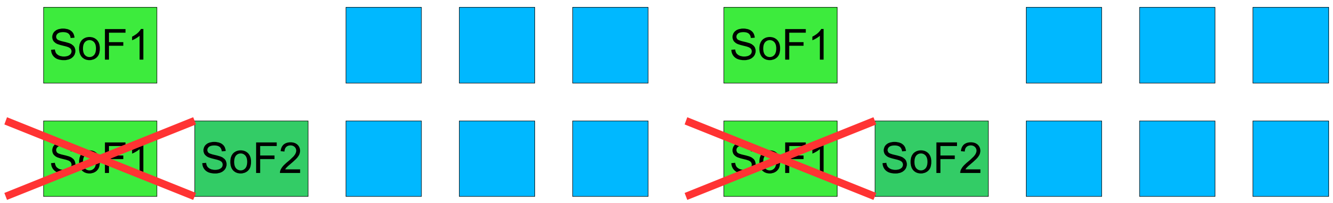


Benefits: - No detection and take-over delay

Drawbacks: - Slaves have to handle the missing SoF somehow

Fall-Back Master (2)

- *Approach C:* Reserve slot for secondary master



Benefits: - No detection and take-over delay on slave side
- Secondary only sends if primary does not

Drawbacks: - Slaves have to adjust their slot offsets
- Reserved slot is lost for data exchange

- Generic challenge:
  - Clock synchronisation between primary and secondary
  - Potential crack in time stamps when switching over (need to be quantified)

Roadmap

Goal: **RTnet 1.0**

Core Requirement: TDMA V2

- Define TDMA Version 2 protocol, state machine, and management interface soon (within a 3-6 months)
- Include hooks for unsolved issues (scheduling, fall-back master, etc.)
- 0.8.0 or at least 0.9.0 shall include TDMA V2!

Discussion!

www.rts.uni-hannover.de/rtnet

kiszka@rts.uni-hannover.de