We are using TRX protocol in OsmoBTS in order to "speak" with transceiver (e.g. OsmoTRX, FakeTRX). Basically it defines three interfaces: clock for TDMA frame clock indications, control (we agreed to call it TRXC) for transceiver management, and data (TRXD) for exchanging Rx/Tx bursts. It was first introduced in OpenBTS project, from which we forked OsmoTRX. For more information, please see: https://github.com/RangeNetworks/openbts/blob/master/TRXManager/README.TRXManager.

The protocol is being used for years, and it still serves us for good. However, it is extremely inflexible. For example, there is no way to send more information about received bursts on TRXD interface, than the fixed-size header already has (TDMA frame number, timeslot number, RSSI, ToA256). On TRXC interface, which is working over UDP as the other ones, there is no way to distinguish command retransmission, which may happen due to timeout, from a regular command. There is no way to notify the L1 (i.e. OsmoBTS) about some events (e.g. device has been disconnected), no way to indicate transceiver version, and so on...

During OsmoDevCon2019 (and before too) we have been discussing the idea of introducing the new version of TRX protocol, which would allow us to solve the mentioned problems. In order to keep backwards compatibility, both L1 and transceiver would initially use the old TRX protocol, while the new version can be optionally enabled by sending a command on TRXC interface. If the transceiver does support the new protocol, it would acknowledge the command. Otherwise, the command is rejected, so L1 would continue to work in "backwards compatibility" mode.

Finally, we need to write a proper protocol description like we already have for GSUP in https://git.osmocom.org/osmo-gsm-manuals/. But before starting to work on this, let's create some kind of a wish list - what would you like to see in the new version of TRX protocol?

Related issues:
- Related to OsmoBTS - Bug #1618: AMR adaption loop doesn't use C/I thresholds... Added 05/16/2019
- Related to OsmoTRX - Feature #3054: Extended (11-bit) RACH support in OsmoTRX Added 05/16/2019
- Related to OsmoBTS - Feature #1569: Report RF interference levels as part of ... Added 05/16/2019
- Related to OsmoTRX - Feature #4081: Add dissector for OsmoTRX protocol Added 05/16/2019

History
#1 - 05/16/2019 07:55 PM - fixeria
- Blocks Feature #1855: provide actual BER or C/I values from osmo-bts-trx into the PCU added

#2 - 05/16/2019 08:05 PM - fixeria
- Related to Bug #1618: AMR adaption loop doesn't use C/I thresholds, only BER added

#3 - 05/17/2019 01:08 PM - ipse

TRXC: the ability to distinguish command retransmissions
As an idea - we can prepend the command/response packets with a counter (sequence number). This will allow not only retransmission detection but also lost commands detection (gaps in the sequence numbers).

#5 - 05/18/2019 10:10 AM - laforge
On Fri, May 17, 2019 at 01:14:13PM +0000, ipse [REDMINE] wrote:

TRXC: the ability to distinguish command retransmissions

As an idea - we can prepend the command/response packets with a counter (sequence number). This will allow not only retransmission detection but also lost commands detection (gaps in the sequence numbers).

This would likely break all compatibility with the existing implementations, so if you change something as drastic as this, I would argue you could just as well move away from UDP altogether for the control protocol. Sure, I get why it makes sense for the bursts... but using an unreliable transport for controlling the transceiver? I don't see any advantage to that.

So if there's something incompatible new for the control side anyway, I would argue one could just as well go for a completely new protocol. The old protocol would continue to exist in parallel for backwards compatibility.

In any case, the important part right now is extending the burst data with more information (training sequence, C/I value), and I suggest to focus on implementing that before doing more radical changes.

#6 - 05/18/2019 10:37 AM - ipse

laforge wrote:

In any case, the important part right now is extending the burst data with more information (training sequence, C/I value), and I suggest to focus on implementing that before doing more radical changes.

Totally agree with this, btw. This is quite straightforward as we discussed in the Gerrit.

On Fri, May 17, 2019 at 01:14:13PM +0000, ipse [REDMINE] wrote:

TRXC: the ability to distinguish command retransmissions

As an idea - we can prepend the command/response packets with a counter (sequence number). This will allow not only retransmission detection but also lost commands detection (gaps in the sequence numbers).

This would likely break all compatibility with the existing implementations,

Well, it would be enabled only if negotiated between the OsmoTRX and OsmoBTS. Any protocol change would break compatibility so I just assume that would be enabled after a negotiation based on the "classical" protocol.

so if you change something as drastic as this, I would argue you could just as well move away from UDP altogether for the control protocol. Sure, I get why it makes sense for the bursts... but using an unreliable transport for controlling the transceiver? I don't see any advantage to that.

Just curious, what do you think of here? SCTP in reliable datagram mode?

I frankly don't see issues in using UDP here - implementing retransmissions and seq.numbers is not difficult but allows us to be in control on how exactly we do the retransmissions unlike with TCP where we have to rely on the OS implementation. E.g. I don't think we want exponential back off here.

#7 - 05/19/2019 07:48 AM - fixeria

- Related to Feature #3054: Extended (11-bit) RACH support in OsmoTRX added

#8 - 05/19/2019 07:50 AM - fixeria

- Checklist item [ ] TRXC: the ability to enable / disable EDGE burst / 11-bit RACH detection added
Checklist item: TRXD: "no burst" indication (e.g. when nothing has been detected)

I just realized that having such indications would devalue the clock interface: there will be no need to send clock indications on a separate interface if OsmoBTS receives either bursts or "no burst" indications on TRXD eight times per TDMA frame (excluding IDLE slots).

#10 - 06/03/2019 10:46 AM - fixeria
- Blocks Feature #3428: Implement handling of NOPE / IDLE indications from Transceiver added

#11 - 06/19/2019 08:20 AM - laforge
- Assignee set to fixeria

#12 - 06/19/2019 12:50 PM - Hoernchen
- Related to Feature #1569: Report RF interference levels as part of RF RESOURCE INDICATION added

#13 - 06/19/2019 02:39 PM - Hoernchen

Getting the noise level during idle timeslots is useful for the RSL RF RESOURCE INDICATION (which, in turn, is useful for handover decisions), which requires averaging the noise level during 1-31 SACCH multiframes (480ms). osmo-trx can currently only report a not really useful average noise of the last 20 idle timeslots using the NOISELEV command.

#14 - 06/19/2019 03:00 PM - laforge

The conclusion of a discussion between fixeria and myself on IRC today was:

In order to make quick progress with the important changes (C/I ratio, RACH training sequence) we want to make a minimal set of incremental changes now [first]:

- OsmoTRX will in the future always start up with the "traditional" frame format
- we add a new "SETFORMAT" control command which allows the BTS to select a new format, identified by a version string like "201906".
- if the TRX rejects that command, the BTS knows it is dealing with an old TRX and it has to continue with the old format
- if the TRX acknowledges that command, the frames will follow the new "201906" format.

this means that every implementation will have to use several different function to encode/decode TRXD messages, one for each format. The SETFORMAT will result in function pointers being set, and the actual fast path (processing of burst data) does not have any additional complexity.

The "201906" format will start with the following modifications (please amend if you know of any):

- adding C/I fields in uplink bursts
- adding traning sequence in uplink bursts

#15 - 06/19/2019 03:50 PM - ipse

First of all - I agree with the proposed solution. I think it's the best way forward and will allow us to extend in the future if needed.

I believe we should have a normal protocol version numbering: 1, 2, 3, etc instead of date-based. The reason for this is below:

We should also add the protocol version to the burst header as the first byte (RTP-like). The reason is to make decoding easier for captures. Otherwise, Wireshark will need to guess the header version which is not great.

Frankly, I'm not sure how useful is backward compatibility between OsmoTRX and OsmoBTS is, and in which directions - new OsmoTRX to support old OsmoBTS? New OsmoBTS to support old OsmoTRX? If the effort is minor, we can implement the backward compatibility. If it's a significant effort, I would just declare no backward compatibility between protocol versions. This will also make negotiation between OsmoTRX and OsmoBTS easier. Right now the proposal above only allows switching between "protocol verion 0" (aka current) and "the latest protocol version". But what is OsmoBTS supports version N and OsmoTRX supports N-1 or N-1, or vice versa? The support matrix gets quite complicated quickly.

#16 - 06/21/2019 06:00 PM - fixeria
- Status changed from New to In Progress
- Priority changed from High to Urgent

#17 - 06/22/2019 06:59 AM - fixeria
Hi Harald, Alexander!

a new format, identified by a version string like "201906" [...]

I believe we should have a normal protocol version numbering: 1, 2, 3, etc instead of date-based.

Agreeing with Alexander here. Something like "201906" would take 6 octets, what is almost as long as the current header length - 8 octets. I don't think we need such overhead, and I also don't think we would ever have more than 16 versions. Thus I vote for having version numbers, not strings.

We should also add the protocol version to the burst header as the first byte (RTP-like). The reason is to make decoding easier for captures. Otherwise, Wireshark will need to guess the header version which is not great.

ACK. Without the format / version indicator in the header Wireshark (or any other sniffer like trx_sniff.py) would have to follow the TRXC conversation, looking for SETFORMAT command, or even worse - guess it using the frame length. But instead of prepending an additional byte, I suggest to reuse 3 bits (MSB) of the first octet which indicates TDMA TN (Time-slot Number) in the current version. Why? This would prevent Wireshark from misinterpreting a TRXD packet of version X as a packet of version Y. For example, the version indicator 0x01 might be interpreted as TDMA TN=1. Also, this way we can save one octet.

Resuming the above sentences, I suggest to assign the current TRXD header version 0, and implement the next version 1 as follows:

<table>
<thead>
<tr>
<th>AA</th>
<th>CC</th>
<th>DD</th>
<th>EE</th>
<th>FF</th>
<th>GG</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TDMA FN</td>
<td>RSSI</td>
<td>ToA</td>
<td>TS</td>
<td>C/I</td>
<td>Burst len</td>
</tr>
</tbody>
</table>

AA (1 octet) - HDR version + TDMA TN (Time-slot Number)

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
<th>Bit numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>X X X X . . . .</td>
<td>HDR version (0..15)</td>
</tr>
<tr>
<td>X X X X</td>
<td>TDMA TN (0..7)</td>
</tr>
<tr>
<td>. . . . X .</td>
<td>Reserved for UMTS TN range extension (0)</td>
</tr>
</tbody>
</table>

BB (4 octets) - TDMA FN (Frame Number), big endian
CC (1 octet) - RSSI (without negative sign, e.g. -50 is 0x32)
DD (2 octets) - ToA (Timing of Arrival) in 1/256 units of symbol, big endian

EE (1 octet) - TS (Training Sequence) + BT (Burst Type)

<table>
<thead>
<tr>
<th>7 6 5 4 3 2 1 0</th>
<th>Bit numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>X X X X</td>
<td>Training Sequence number (0..7)</td>
</tr>
<tr>
<td>. . . . X X X</td>
<td>Modulation, TS set number</td>
</tr>
<tr>
<td>0 0 0 0 X X X</td>
<td>GMSK, TS set 0..3</td>
</tr>
<tr>
<td>0 1 0 X X X</td>
<td>8-PSK, TS set 0..2</td>
</tr>
<tr>
<td>0 1 1 X X X</td>
<td>AQPSK, TS set 0..2</td>
</tr>
<tr>
<td>1 0 0 X X X</td>
<td>16QAM, TS set 0..2</td>
</tr>
<tr>
<td>1 0 1 X X X</td>
<td>32QAM, TS set 0..2</td>
</tr>
<tr>
<td>1 1 1 X X X</td>
<td>Reserved</td>
</tr>
<tr>
<td>X 0 0 0 0 0 0 0</td>
<td>IDLE / nope frame indication (1)</td>
</tr>
</tbody>
</table>

FF (2 octets) - C/I (match score) of the TS, big endian
GG (2 octets) - Burst length, big endian

So the extension of byte AA should be clean. One reserved bit may be useful to extend the range of TDMA TN from 0..7 to 0..15 in case anybody would ever want to transfer UMTS bursts, where the amount of time-slots is 15 IIRC. Fields BB, CC, and DD are the same as in the current version, at the same positions.

The field EE is aimed to indicate training sequence set, number, and the modulation. I know, we only need GMSK and 8-PSK, this is just to keep some room for potential features. The bit number 7 is set to high when either nothing has been detected, or during IDLE frames, so we can deliver noise levels. The field GG shall be set to 0x00. Otherwise it indicated the following burst length, so there would be no need to do pkt_len - hdr_len, and would be easier to detect short reads.

Any objections or recommendations? I am slowly starting to extend the FakeTRX toolkit, so after that I am going to write a TTCN-3 test case that would verify C/I processing in OsmoBTS.

#18 - 06/22/2019 03:30 PM - ipse

I agree with the approach.

I would just recommend the following formatting of the EE octet to make it easier to understand:

<table>
<thead>
<tr>
<th>AA</th>
<th>BB BB BB BB</th>
<th>CC</th>
<th>DD DD</th>
<th>EE</th>
<th>FF FF</th>
<th>GG GG</th>
</tr>
</thead>
</table>

03/08/2022
I would just recommend the following formatting of the EE octet to make it easier to understand:

Thanks, I'll consider this when writing the documentation.

TRX Toolkit (DATAMSG class) has been updated, please see:

https://gerrit.osmocom.org/#/c/osmocom-bb/+/14575/
https://gerrit.osmocom.org/#/c/osmocom-bb/+/14576/
https://gerrit.osmocom.org/#/c/osmocom-bb/+/14579/

It's still unclear to me, how should we encode C/I values.

I asked tnt, and he wrote:

(17:41:50) fixeria: tnt: any ideas how should we encode C/I on the TRXD interface?
(17:42:35) fixeria: tnt: computeCI() in your branch returns float
(17:43:32) fixeria: should we round it to the closest integer as we already do for ToA: (int) (ToA * 256.0 + 0.5)?

(17:45:05) tnt: fixeria: it's a dB value but can also be negative, it's more like SNR or something like that.
(17:46:06) tnt: fixeria: precision is really not that important because that value is an estimation based on very few samples so it's a very "noisy" value and will need to be averaged by the upper layers to get something that makes sense, so for instance mapping it to a int8_t would be perfectly appropriate.

On the PCU interface, we're using int16_t (2 octets, big endian), but there it's reasonable, because we need to average several C/I values of the received bursts, and we don't want to lose precision. So should I use one octet instead of two? What is the min / max value range?

TRXD version negotiation on TRXC is not (yet) implemented, so OsmoBTS is still using the version 0x00 at startup. However, my attempt to switch TRXD version at run-time by injecting SETVER command (using my ctrl_cmd.py tool) was successful ;)

As was discussed with pespin, most likely we don't need the burst length field (octets GG). I am going to update the existing changes.

OsmoBTS has been updated, see:

https://gerrit.osmocom.org/c/osmo-bts/+/14592 osmo-bts-trx/trx_if.c: introduce TRXD header version handling
https://gerrit.osmocom.org/c/osmo-bts/+/14593 osmo-bts-trx/trx_if.c: introduce TRXD header version 0x01 support

Please note that only the message parsing logic is changed. Handling of TSC and C/I is behind the scope of this ticket.

03/08/2022
Some related work in osmo-trx:

remote: https://gerrit.osmocom.org/c/osmo-trx/+/14632 Transceiver: Drop last 2 garbage bytes sent at end in uplink bursts
remote: https://gerrit.osmocom.org/c/osmo-trx/+/14629 Introduce structs to encode TRXD packets
remote: https://gerrit.osmocom.org/c/osmo-trx/+/14630 Transceiver: refactor: gather uplink burst parameters in struct
remote: https://gerrit.osmocom.org/c/osmo-trx/+/14631 Transceiver: Move nbits calculation to pullRadioVector()

#24 - 07/04/2019 05:25 PM - fixeria

- Checklist item [x] TRXD: “no burst” indication (e.g. when nothing has been detected) set to Done
- Checklist item [x] TRXD: detected training sequence (and it's C/I weight) set to Done
- Checklist item [x] TRXD: noise level indication set to Done
- Checklist item [x] TRXD: facilitate further extensibility? set to Done

That's what we have been working on so far. The new TRXD header format version (1) solves the following problems (marked as done).

#25 - 07/04/2019 05:58 PM - fixeria

The TRXD header format negotiation is being discussed right now. We agreed on the TRXC command name - 'SETFORMAT', but some aspects still need to be discussed. The key point is that OsmoBTS will send this command at start up (before POWERON). If the transceiver does support the format negotiation, it will respond to that command. Otherwise, it would respond with 'RSP ERR 1', so we stick to the old version (0). FakeTRX is a little bit special, because it confirms all unknown commands with status code 0.

Keeping that in mind, we decided to abuse the status code in the response to indicate the actual header version supported by the transceiver:

BTS -> TRX: CMD SETFORMAT VER
BTS <- TRX: RSP SETFORMAT CODE VER

== BTS requests version 2, TRX confirms that it's supported and now used
BTS -> TRX: CMD SETFORMAT 2
BTS <- TRX: RSP SETFORMAT 2 2

== BTS requests version 2, but TRX only supports version 1
BTS -> TRX: CMD SETFORMAT 2
BTS <- TRX: RSP SETFORMAT 1 2

== BTS requests version 2, old FakeTRX does not support the format negotiation
BTS -> TRX: CMD SETFORMAT 2
BTS <- TRX: RSP SETFORMAT 0 2

So the CODE can be either equal or lower than VER. The question is how should we interpret / implement CODE < VER?

- Should we interpret this as an indication of the highest supported version, and resend 'SETFORMAT' with that version?
- Or should we interpret this as an indication of the highest supported and actually applied version?

IMHO, if the indicated version Y is lower than the one requested by OsmoBTS X, we definitely want at least Y. I don't see any benefits of resending 'SETFORMAT' with Y, and would prefer to follow the second approach. What do you guys think?

#26 - 07/04/2019 10:08 PM - pespin

I'd resend SETFORMAT, this way we let BTSTRX decide what it wants to do, instead of TRX applying a version Y nobody requested (BTS requested X).

#27 - 07/06/2019 04:31 AM - fixeria

- Status changed from In Progress to Feedback

[...] this way we let BTSTRX decide what it wants to do [...] Not sure if the BTS would ever need to keep the old TRXD header version if Y < X. Resending 'SETFORMAT' in that case is a kind of explicit negotiation. Making it implicit (i.e. applying version Y <= X) would help us to avoid dealing with detecting duplicate responses. Let's see what others think about this...

#28 - 07/16/2019 05:13 PM - fixeria

- Blocks deleted (Feature #1855: provide actual BER or C/I values from osmo-bts-trx into the PCU)
#29 - 07/24/2019 03:54 PM - fixeria

- Checklist item [] TRXD: document the recent changes added

(22:19:09) LaF0rge: fixeria: the osmo-trx user manual still contains an incomplete old description ...

(22:32:00) LaF0rge: http://git.osmocom.org/osmo-gsm-manuals/tree/common/chapters/trx_if.adoc

(22:32:15) LaF0rge: it's there as it probably gets included in both the osmo-bts and the osmo-trx user manual

#30 - 07/24/2019 04:07 PM - pespin

Stuff from last comment is in line with https://osmocom.org/issues/4125

#31 - 07/25/2019 05:18 PM - pespin

- Checklist item [x] TRXD: document the recent changes set to Done

Documentation improvements submitted to gerrit: https://gerrit.osmocom.org/c/osmo-gsm-manuals/+14940 common: trx_if.adoc: Add documentation about TRXDv1 and SETFORMAT

https://gerrit.osmocom.org/c/osmo-gsm-manuals/+14938 common: trx_if.adoc: Improve documentation

#32 - 08/05/2019 10:03 PM - fixeria

- Related to Feature #4081: Add dissector for OsmoTRX protocol added

#33 - 08/06/2019 11:38 AM - fixeria

Regarding the first two points:

[ ] Notifications from transceiver (e.g. device has been disconnected)
[ ] Info / feature negotiation (e.g. version, device type / name)

I've got an idea. The transceiver can send this information on the "clock" socket (well, we call it "clock" because it's only used for clock indications at the moment) using the existing message type - "IND". For example:

- "IND EVENT dev_disconnect" (over-/under-runs, device has been disconnected, etc.),
- "IND INFO VERSION 1.0.0-90-g6b30ab0",
- "IND INFO DEVTYPE 0 UHD B210".

What do you think?

#34 - 08/06/2019 11:49 AM - pespin

Hi fixeria, indeed, I wanted to explore the IND possibilities, but we'd need to check how would that work in backward compatibility sense, I mean would old versions of osmo-bts-trx accept this kind of packets? or would they exit?

#35 - 12/01/2019 09:11 AM - laforge

I have the feeling that this ticket should be updated to be in sync with reality?

#36 - 12/01/2019 11:15 AM - fixeria

- Status changed from Feedback to Stalled

[...] would old versions of osmo-bts-trx accept this kind of packets? or would they exit?

I will check this soon. Looking at the code, I don't think this would cause any problems.

I have the feeling that this ticket should be updated to be in sync with reality?

It's is in sync with the reality: the control plane still needs several improvements :)

#37 - 12/01/2019 02:44 PM - fixeria
I will check this soon. Looking at the code, I don't think this would cause any problems.

Ok, my experiment confirms this (osmo-bts-trx just warns us, but works):

DTRX NOTICE trx_if.c:112 phy0.0: Unknown message on clock port: IND INFO Mahlzeit!

#38 - 02/17/2020 09:27 AM - fixeria
- Blocks deleted (Feature #3428: Implement handling of NOPE / IDLE indications from Transceiver)

#39 - 05/18/2020 04:05 PM - fixeria
- Priority changed from Urgent to Low

#40 - 06/23/2020 02:16 PM - pespin

#41 - 09/05/2020 06:46 AM - fixeria
- Checklist item [ ] TRXD: burst "concatenation" in multi-TRX setups added

As was discussed with Hoernchen, in a multi-TRX setup we may experience weird burst reordering issues. This happens because each transceiver has a dedicated UDP/TRXD connection, thus the number of transceivers equals to the number of sockets that we need to take care of.

The idea is to extend the TRXD protocol, so we could "concatenate" several Downlink bursts into a single UDP/TRXD packet, and send them all together. We can group bursts either by the timeslot number, so we would always send 8 packets per TDMA frame regardless of how many transceivers do we have, or by TDMA frame number, so we send 1 packet per TDMA frame.

Just to illustrate the difference, let's say we have 8 transceivers and they all Tx/Rx on all timeslots (100% load). With the current code, we would have to send 8 * 8 = 64 UDP/TRXD packets and thus call send() 64 times per a TDMA frame. Grouping bursts by the timeslot number or TDMA frame number would reduce this number 8 or 64 times respectively.

#42 - 09/05/2020 07:03 AM - fixeria
- Checklist item [ ] TRXD: indicate type of burst in TRX2L1 messages added

If some day we will need to implement handling of PACKET CONTROL ACKNOWLEDGEMENT message (see 3GPP TS 44.060, 11.2.2) in form of 4 Access Bursts in osmo-bts, we would need to know what kind of bursts we receive from the transceiver: Access or Normal. This way we would immediately know how to decode the message.

#43 - 09/05/2020 09:40 AM - laforge
Hi,

On Sat, Sep 05, 2020 at 06:47:01AM +0000, fixeria [REDMINE] wrote:

    Checklist item [ ] TRXD: burst "concatenation" in multi-TRX setups added

maybe create a separate ticket for this?

    The idea is to extend the TRXD protocol, so we could "concatenate" several Downlink bursts into a single UDP/TRXD packet, and send them all together. We can group bursts either by the timeslot number, so we would always send 8 packets per TDMA frame regardless of how many transceivers do we have, or by TDMA frame number, so we send 1 packet per TDMA frame.

I suggest to use 'batching' or 'bundling'. 'Concatenation' sounds more like you are making a single burst longer. In the end, we may be able to re-use the terminology of the OSMOX protocol? there we do a similar operation but for RTP codec frames.

    Just to illustrate the difference, let's say we have 8 transceivers and they all Tx/Rx on all timeslots (100% load). With the current code, we would have to send 8 * 8 = 64 UDP/TRXD packets and thus call send() 64 times per a TDMA frame. Grouping bursts by the timeslot number or TDMA frame number would reduce this number 8 or 64 times respectively.

For the sake of completeness: We could - at least within one TRX - also move from 8x write() to 1x writev(). I'm not saying we should, I'm just pointing out possible intermediate options.

I think we should study the potential implications on bot CPU
consumption and thread synchronization.

If every radio carrier has its own socket / UDP port (like now), they can each be processed in a separate thread (at least on the TRX side), and there is no need to synchronize / dispatch from a single socket to multiple threads.

So changing to one global per-BTS socket with all TRX inside may solve one optimization problem but create another potential bottleneck?

The above would be in favor of just 'batching' all the frames (timeslots) of one TRX in one UDP packet. This, on the other hand has the consequence of adding latency (we need to wait for 7 timeslots to complete before sending a UDP packet, i.e. 4ms delay). Given that we just reduced fn-advance, we might be able to afford this?

Also, doing CPU processing of all four timeslots at the same time vs. distributed over time might change our CPU utilization pattern. There would be larger peaks with longer gaps if we process 8 timeslots in one socket-read.

I would guess that the CPU processing pattern changes are more than compensated by the reduction in CPU waste for a factor of 7 syscall reduction.

If we’re less worried about ‘packets per second in the kernel network stack’ and only/mostly worried about ‘number of system calls in our programs’, then using io_uring might also be a be a good solution. It allows us to receive/transmit from/to any number of sockets within a single syscall. The disadvantage here is that a number of distributions including CentOS8 don’t have kernels with io_uring yet.

#44 - 09/05/2020 03:19 PM - Hoernchen

The most reasonable approach would be to only go for one packet for one TS, so 8 per Frame - this would not change any latencies, but ensure in-order delivery among the same ts and reduce syscall overhead to 8 per frame instead of 8*ts per frame, so basically the one trx case. The lo mtu was changed to 65k in 2012

That being said there is also ZMQ or nanomsg and so on that would conveniently and transparently wrap transport options including pipes and still offer a way to transfer a block of data from a to b without having to worry about packet boundaries. As exciting as io_uring is, we’re not really able to enforce the "only the newest kernels please" rule, so it’s of limited usefulness right now - but it doesn’t interfere with the idea of batching on a per-TS basis and is more of an internal implementation detail that could be added later. In the batching per frame case the kernel thread would probably go to sleep even in the SQPOLL case and require a wakeup syscall, so no gain there.

A reasonable implementation would read from a socket into one preallocated buffer from a buffer pool, send pointers that point at the per thread/trx data within that buffer to the threads handling distinct trx operations, and ensure the last thread (whichever that may be) returns that large buffer to the buffer pool upon finishing handling the data, to circumvent allocation operations. No dispatch problem there, and only one "large" read, with the "cost" of one dispatch thread wakeup, the per-trx threads have to wake up anyway.

#45 - 09/14/2020 10:18 AM - fixeria

- Related to Bug #4658: Wrong burst order in a multi-trx setup added

#46 - 02/06/2021 02:15 PM - laforge

- Priority changed from Low to Normal

#47 - 02/17/2021 02:09 PM - fixeria

- Status changed from Stalled to In Progress

#48 - 02/17/2021 02:10 PM - fixeria

- Related to Feature #4941: VAMOS support in OsmoBTS added

#49 - 04/22/2021 07:07 PM - fixeria

- Checklist item changed from TRXD: burst "concatenation" in multi-TRX setups to TRXD: burst batching in multi-TRX setups
- Checklist item [x] TRXD: burst batching in multi-TRX setups set to Done
- Checklist item [x] TRXD: indicate type of burst in TRX2L1 messages set to Done
- % Done changed from 50 to 70
Hello everyone,

I am glad to announce that we've completed the TRXDv2 specification! Here is a brief summary:

- Introduced the concept of burst batching (many bursts in one message);
- Changed the field ordering (facilitating aligned access);
- New field: batching indicator;
- New field: TRX number;
- New field: SCPIR for VAMOS.


Wireshark dissector for TRXDv2 can be found here:

https://gitlab.com/axilirator/wireshark/-/tree/fixeria/trxd

Updated by fixeria 8 months ago:

[x] TRXD: indicate type of burst in TRX2L1 messages added

Addressed in section 19.3.2.1: "The transceiver shall use '0110'B as the modulation type to indicate an Access Burst on PDTCH".

Updated by fixeria 8 months ago:

[x] TRXD: burst batching in multi-TRX setups

Addressed in section 19.3.4 "PDU batching".

#50 - 04/30/2021 06:48 PM - fixeria

Updated by fixeria 8 days ago:

Wireshark dissector for TRXDv2 can be found here:

https://gitlab.com/axilirator/wireshark/-/tree/fixeria/trxd

Here is a Pull Request adding TRXDv2 support to Wireshark:

https://gitlab.com/wireshark/wireshark/-/merge_requests/2886

I fixed some cosmetic problems, and finally submitted it upstream.

#51 - 10/27/2021 10:22 PM - fixeria

- Status changed from In Progress to Stalled

#52 - 10/27/2021 10:36 PM - fixeria

- Related to Feature #5283: Implement TRXDv2 support added