OsmocomTETRA - Bug #1897

Inefficient sequence search in tetra_find_train_seq

12/27/2016 12:24 AM - jenda

Status:	New	**Start date:**	12/27/2016
Priority:	Low	**Due date:**	
Assignee:		**% Done:**	0%
Category:			
Target version:			
Resolution:		**Spec Reference:**	

Description

Hi,

finding synchronization eats 3 to 4 times more CPU than the actual decoding on my machine (i3-2350M). According to the Perf profiler, all time is spent in tetra_find_train_seq, particularly in the memcmp calls.

Rewriting the search using a more efficient algorithm, for example Aho–Corasick, may speed it up an order of magnitude.

How to reproduce:
1. Capture valid Tetra bits.
2. Generate the same amount of random bits.
3. Observe that the random bits take several times longer for tetra-rx to process.

History

#1 - 12/27/2016 08:15 PM - laforge

Hi jenda,

On Tue, Dec 27, 2016 at 12:24:27AM +0000, jenda [REDMINE] wrote:

> finding synchronization eats 3 to 4 times more CPU than the actual decoding on my machine (i3-2350M). According to the Perf profiler, all time is spent in tetra_find_train_seq, particularly in the memcmp calls.

thanks for pointing this out. osmo-tetra is a minimal-effort proof-of-concept implementation that was done in my spare time.

> Rewriting the search using a more efficient algorithm, for example Aho–Corasick, may speed it up an order of magnitude.

I am very much looking forward to receiving and integrating such a controbution!

--
- Harald Welte <laforge@gnumonks.org> http://laforge.gnumonks.org/
==========
"Privacy in residential applications is a desirable marketing option."
(ETSI EN 300 175-7 Ch. A6)

#2 - 04/10/2017 01:04 AM - jenda

- File tetra.patch added

Hi,
what do you think about this patch? Implementing full-blown Aho-Corasick was finally not necessary.

- tetra_burst_sync_in() was eating 64 bits a time, but it was always running memmove(3) and tetra_find_train_seq() on the whole 4KiB array. I have moved the file-reading stuff inside and we now always fill the whole buffer. (also, why use POSIX read(2) when we have standard and buffered file I/O)
- We now shift the input bitstream to a variable and compare it with a single comparison instead of memcmp(3).

Now, finding training sequence is 40 times faster when compiled with -O0 and 110 times faster when compiled with -O3 ;)

I have tried to run it on several captures of the local network and I get the exact same results (sha1sums of output files match), so I hope it is working (no idea if some testsuite exists).

I will be happy to get suggestions regarding the logic, coding etc.

Files

tetra.patch	7.52 KB	04/10/2017	jenda