

# Intel® Processor Trace on Linux

Tracing Summit 2015

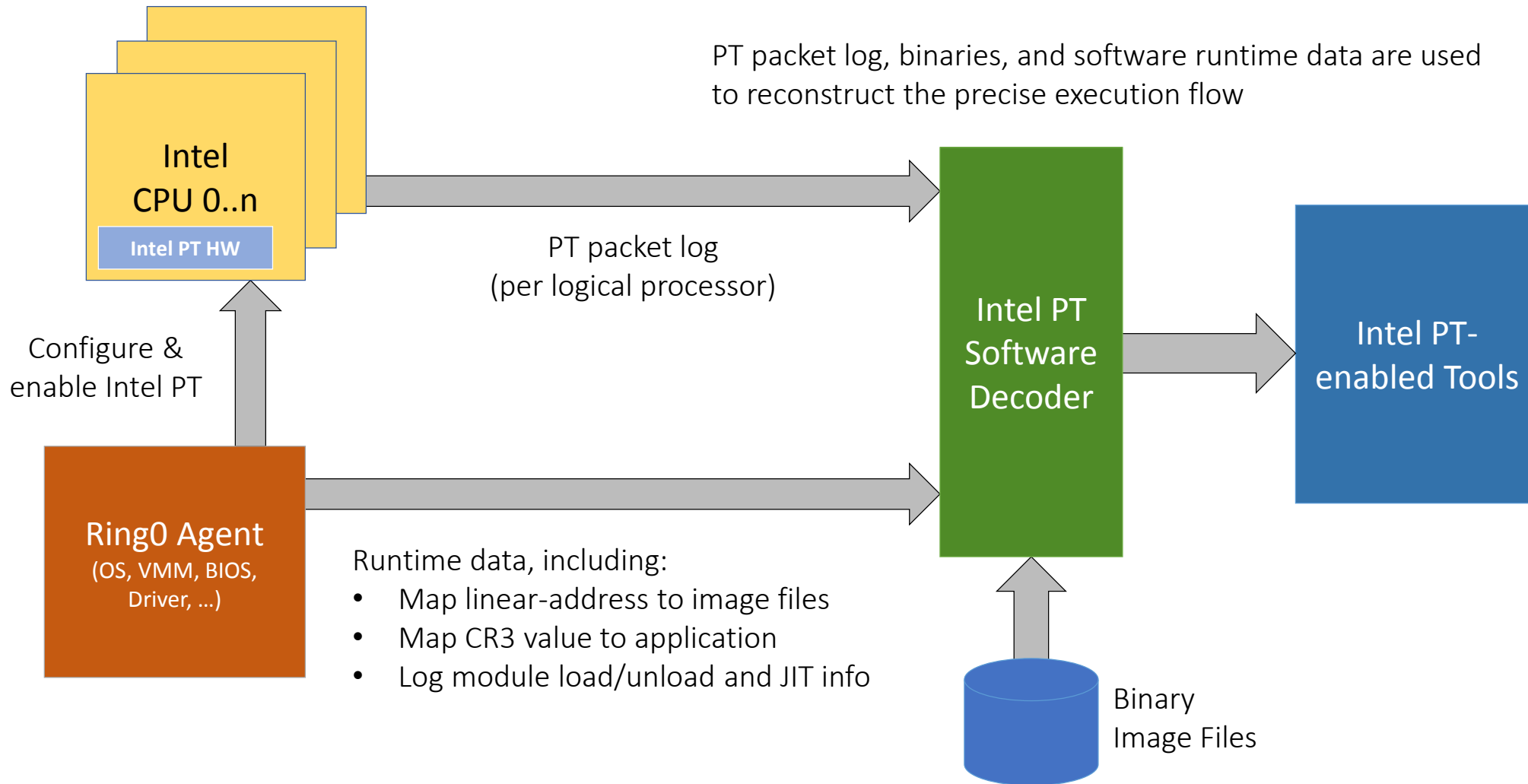
Andi Kleen

Beeman Strong

# What is Intel<sup>®</sup> Processor Trace (PT)?

- Intel PT is a hardware feature that logs information about software execution with minimal impact to system execution
- Supports control flow tracing
  - Decoder can determine exact flow of software execution from trace log
  - Target <5% performance overhead
    - Depends on processor generation and usage model
- Can store both cycle count and timestamp information
  - For deep performance analysis, and synch with other traces, screen shots, etc

# Intel® Processor Trace Components



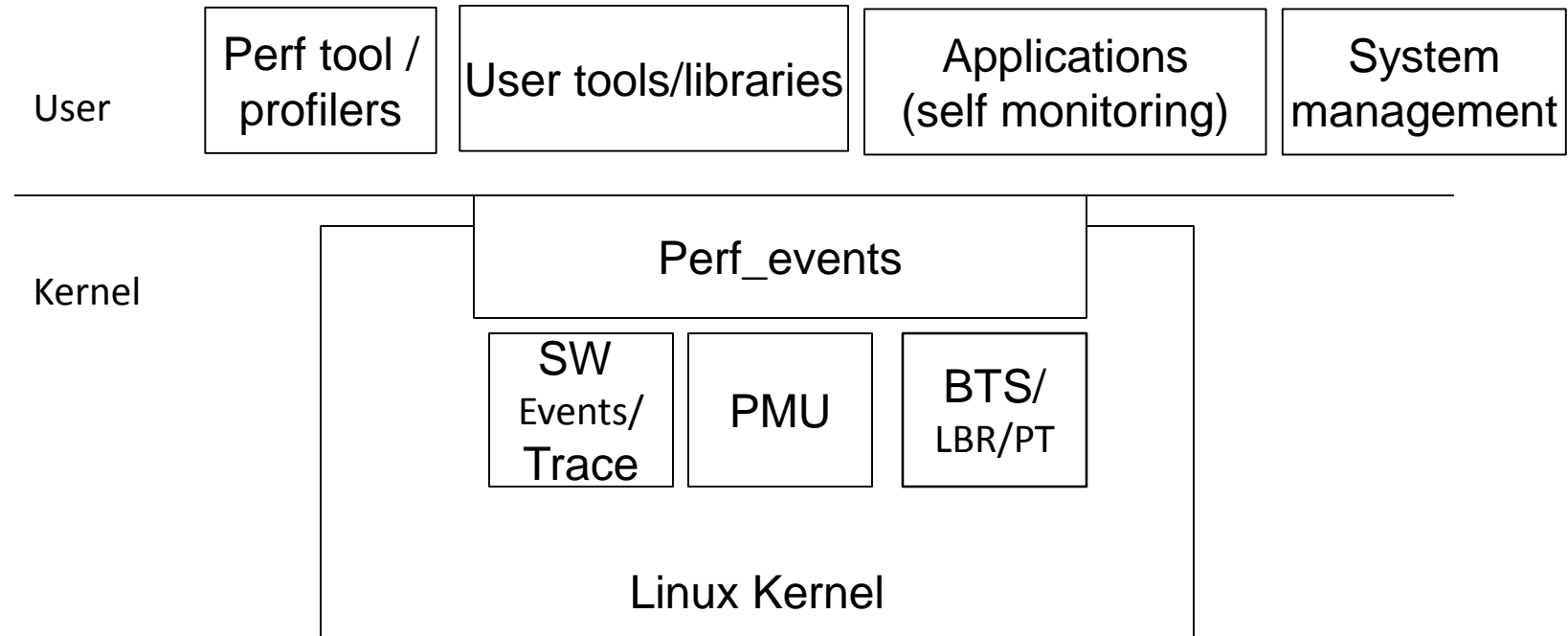
# Intel<sup>®</sup> Processor Trace Trace Details

- Trace information is generated only for non statically known control flow changes
  - i.e., conditional branches generate only a taken/not (1 bit) indication
  - Minimizes output bandwidth, average <1 bit per retired instruction
- Certain (processor) mode changes are captured
  - CR3, 32-bit/64-bit mode, VMCS pointer, TSX transaction state, etc
- Periodic sync points with current values of traced state
  - IP, TSC, mode, frequency, SW context, etc
- Can filter trace by CPL, CR3, or IP ranges
  - VMM can opt-out of tracing
  - SMM & SGX filtered out by default, can opt-in

# Usages for Intel® Processor Trace

- Trigger save of Intel® Processor Trace log for post-mortem analysis
  - Save on crash, core dump, software event(s), ...
- Debug short-lived, non-steady-state performance issues
  - i.e., responsiveness problems, glitches/janks
  - Hard to catch with sampling, but PT captures everything with precise timing info
- Server cluster sampling
  - Enable trace on fraction of nodes (to limit bandwidth & performance impact)
  - Collect log on systems that are impacted by performance or functional issues
- Replace call-stack info with full, timed control flow trace
  - Provides path history even when stack is corrupted
  - Provides not just function hierarchy, but *why* that path was taken
- And more...
  - Unexpected wakes, code coverage analysis, TSX transaction behavior analysis, ...

# Linux “perf events” overview



Naming:

- API, framework “perf events”
- User tool “perf”

# Perf implementation

- PT is integrated into perf events. Uses perf metadata to generate perf events in user space decoder.
  - PT branch data is output as perf events
- Fully integrated into OS
  - Context switched per thread or cgroup
  - Available to non-root users
- Kernel driver is in Linux 4.1, perf user tools will be in 4.3

# PT modes

## Current

- Full trace mode
  - Continuous tracing while writing data to disk
  - Trace as long as the disk keeps up
- Snapshot mode
  - Run ring buffer, stop trace on event of interest
  - Save only tail of trace

## Upcoming

- Sampling mode
  - Sample workload, collect PT context from ring buffer around sample
- Core dump
  - Enable with rlimit, run PT as ring buffer in background
  - Dump as part of core dump when program crashes
- System crash mode
  - Run global PT as ring buffer
  - Dump as part of system crash dump



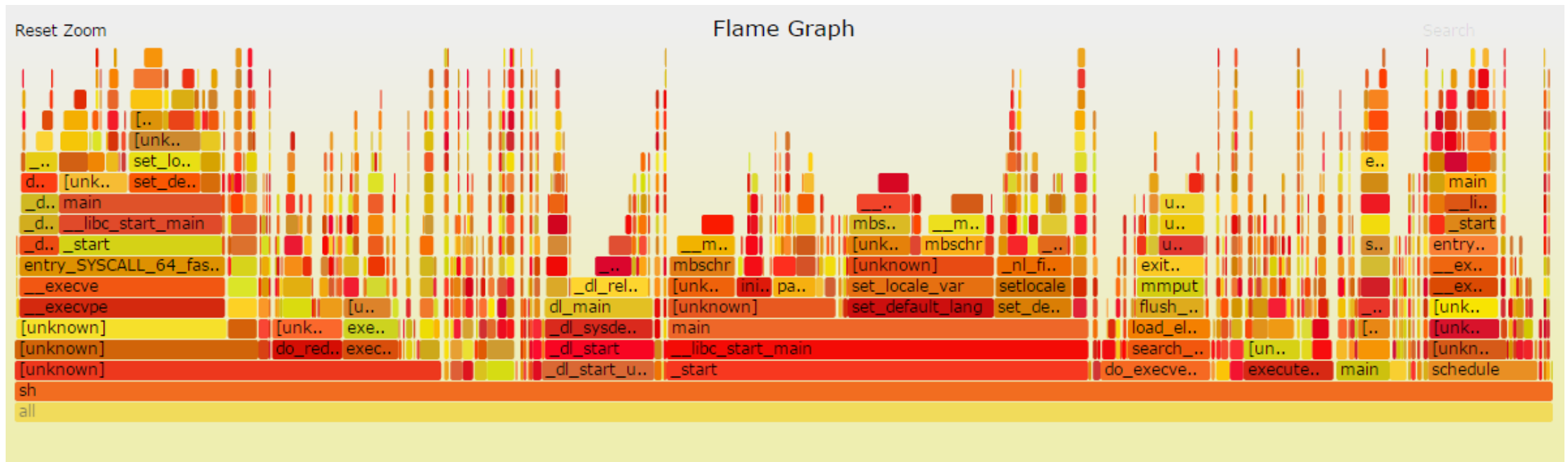
# gdb

- Gdb 7.10 supports PT for “backwards debugging” (reverse-step)
  - Uses perf interface, works as non root
- On break point or crash can look backwards to see what happened

```
(gdb) reverse-step
do_crash () at src/stack.c:59
59         return comp(&arg);
(gdb) record instruction-history -
67         0x0000000000400623 <do_crash+20>:      mov     -0x8(%rbp),%rax
68         0x0000000000400627 <do_crash+24>:      test    %rax,%rax
69         0x000000000040062a <do_crash+27>:      jne     0x400635
70         0x0000000000400635 <do_crash+38>:      mov     -0x8(%rbp),%rax
71         0x0000000000400639 <do_crash+42>:      mov     -0x8(%rbp),%rdx
72         0x000000000040063d <do_crash+46>:      mov     0x8(%rdx),%rdx
73         0x0000000000400641 <do_crash+50>:      add     (%rax),%rdx
74         0x0000000000400644 <do_crash+53>:      mov     %rdx,%rax
75         0x0000000000400647 <do_crash+56>:      leaveq
76         0x0000000000400648 <do_crash+57>:      retq
```

# Perf Output Options: Histograms

- With perf report
- Or using Brendan Gregg's flamegraph tools with PT input



# Perf Output Options: SQL

```
[mosh] ak@tassilo:~  
80216718187974 | 7138 | vim | _int_malloc | 39  
80216718187974 | 7138 | vim | _int_malloc | 45  
80216718187974 | 7138 | vim | _int_malloc | 53  
80216718187974 | 7138 | vim | _int_malloc | 64  
(10 rows)  
  
pt=# select time,tid,command,symbol,sym_offset from samples_view where symbol = '_int_malloc' limit 10;  
      time      | tid  | command | symbol      | sym_offset  
-----+-----+-----+-----+-----  
80216718187974 | 7138 | vim     | _int_malloc | 0  
80216718187974 | 7138 | vim     | _int_malloc | 4  
80216718187974 | 7138 | vim     | _int_malloc | 8  
80216718187974 | 7138 | vim     | _int_malloc | 12  
80216718187974 | 7138 | vim     | _int_malloc | 20  
80216718187974 | 7138 | vim     | _int_malloc | 30  
80216718187974 | 7138 | vim     | _int_malloc | 39  
80216718187974 | 7138 | vim     | _int_malloc | 45  
80216718187974 | 7138 | vim     | _int_malloc | 53  
80216718187974 | 7138 | vim     | _int_malloc | 64  
(10 rows)  
  
pt=# █
```

- Can write branch trace data into SQL database
- Supports custom analysis of branch data

# Processing PT data in own tools

- Multiple possibilities:
  - Process perf script output
  - Write own decoding tool reading from perf.data using libipt
  - Write on top of kernel interface
  - Write as backend for SQL database representation
- Lots of data available – can you make use of it?

# More Tools

- [Intel® System Studio](#), including Intel® VTune™ Amplifier for Systems, GDB, WinDBG plug-in, and Intel® System Debugger
- [ASSET InterTech\\* SourcePoint\\* for Intel](#)
- [Lauterbach\\* TRACE32\\*](#)
- simple-pt -- standalone Linux PT tool
- libipt decoder



- Also, [Intel® Platform Analysis Library](#) (Intel® PAL) provides libraries for trace decode and control flow reconstruction, to ease development of Intel PT-enabled tools

# Summary

- Processor Trace available today on 5<sup>th</sup>+ gen Intel Core CPUs
- Integrated in Linux perf 4.1/4.3 and gdb 7.10
- Provides rich data on program execution
- References:
  - Perf PT overview <https://lwn.net/Articles/649576/>
  - libipt decoding library for writing own processing tools <https://github.com/01org/processor-trace>
  - gdb with PT support: in gdb 7.10 with libipt on kernel 4.1+
  - simple-pt <https://github.com/andikleen/simple-pt>

# Backup

# perf script raw output

- % perf record -e intel\_pt//u ls
- % perf script -F time,comm,cpu,sym,dso,ip,srcline
- ls [001] 454279.326516: 0 [unknown] ([unknown])
- ls [001] 454279.326516: 7fdeb58b1630 \_start (/lib/x86\_64-linux-gnu/ld-2.17.so)
- .:0
- ls [001] 454279.326527: 0 [unknown] ([unknown])
- ls [001] 454279.326527: 7fdeb58b1633 \_start (/lib/x86\_64-linux-gnu/ld-2.17.so)
- .:0
- ls [001] 454279.326527: 7fdeb58b4fbf \_dl\_start (/lib/x86\_64-linux-gnu/ld-2.17.so)
- get-dynamic-info.h:44
- ls [001] 454279.326532: 0 [unknown] ([unknown])
- ls [001] 454279.326532: 7fdeb58b4fc6 \_dl\_start (/lib/x86\_64-linux-gnu/ld-2.17.so)
- rtld.c:385
- ls [001] 454279.326539: 0 [unknown] ([unknown])
- ls [001] 454279.326539: 7fdeb58b4fe1 \_dl\_start (/lib/x86\_64-linux-gnu/ld-2.17.so)
- rtld.c:414



# Perf Output Options: Histogram Report

```
ak@odilo:~/data
# Samples: 1K of event 'instructions'
# Event count (approx.): 324186460
#
# Overhead  Command  Shared Object      Symbol
# .....
#
# 16.46%  vim      libc-2.17.so      [.] __strncmp_sse42
#      |
#      |---__strncmp_sse42
#      |
#      |---87.44%-- 0x9373e
#      |
#      |---98.46%-- do_autocmd
#      |              do_cmdline
#      |
#      |---76.56%-- __GI___libc_read
#      |              _IO_file_underflow@@GLIBC_2.2.5
#      |              _IO_default_uflow
#      |              _IO_getline_info
#      |              _IO_fgets
#      |              0x6f8e8
#      |              getsourceline
#
```

- Supports high-frequency exact sampling

# Perf Output Options: Raw Dump

```
ak@odilo:~/data
. ... raw event: size 48 bytes
. 0000: 47 00 00 00 00 00 30 00 00 00 20 00 00 00 00 00 G.....0... ..
. 0010: 00 00 00 00 00 00 00 00 ec ce ec da 94 ae 00 00 .....
. 0020: 02 00 00 00 e2 1b 00 00 02 00 00 00 00 00 00 00 .....
.
0xcb0 [0x30]: PERF_RECORD_AUXTRACE size: 0x200000 offset: 0 ref: 0xae94daecceec idx
.
. ... Intel Processor Trace data: size 2097152 bytes
. 00000000: 02 82 02 82 02 82 02 82 02 82 02 82 02 82 02 82 PSB
. 00000010: 19 a1 90 27 d8 94 ae 00 TSC 0xae94d82790a1
. 00000018: 02 43 00 66 ab 09 00 00 PIP 0x4d5b300
. 00000020: 02 03 07 00 CBR 0x7
. 00000024: 99 20 MODE.TSX TxAabort:0 InTX:
. 00000026: 99 01 MODE.Exec 64
. 00000028: 7d 68 29 09 81 ff ff 00 FUP 0xffff81092968
. 00000030: 02 23 00 00 00 00 00 00 PSBEND
. 00000038: 71 6a 29 09 81 ff ff 00 TIP.PGE 0xffff8109296a
. 00000040: 6d ff e3 06 81 ff ff 00 TIP 0xffff8106e3ff
. 00000048: 6d ef e6 06 81 ff ff 00 TIP 0xffff8106e6ef
. 00000050: 6d 32 f0 06 81 ff ff 00 TIP 0xffff8106f032
. 00000058: 06 00 00 00 00 00 00 00 TNT T (1)
. 00000060: 6d 59 97 15 81 ff ff 00 TIP 0xffff81159759
. 00000068: 02 a3 cf 02 00 00 00 00 TNT NTTNTTTT (9)
:
```