Userspace Tracing in Linux

with eBPF

Mo Chen & Masaori Koshiba

What is Tracing?

- Observe a program at a specific location without stopping.
- Why?
  - Debug while running
  - Measure performance
  - Understand program behavior
- Small overhead

Where can I trace?

USDT

User Statically-Defined Tracing

- Inspired by DTrace
- Static tracing
  - Compiled into program
  - Stable API
  - Arguments
    - int, pointer, string
- comparable to kernel tracepoints

USDT
internals

- NOP instruction
- Metadata in ELF notes section
- When registered, NOP becomes breakpoint
- Switch to kernel when hit
- Optional semaphore - # of observers
  - skip unnecessary argument calculations
  - generated header

uprobe
and uretprobe

- Dynamic tracing
- Arbitrary tracepoint address
- Comparable to kernel kprobe/kretprobe
- Instruction overwritten by breakpoint

USDT

example

```
Displaying notes found in: .note.stapsdt
  Owner                Data size          Description
  stapsdt              0x00000040         NT_STAPSDT (SystemTap probe descriptors)
    Provider: foo
    Name: bar
    Location: 0x0000000000001188, Base: 0x0000000000002011, Semaphore: 0x0000000000000000
    Arguments: 8@-24(%rbp) -4@-28(%rbp)


void f(const char *string_arg, int integer_arg) {
  DTRACE_PROBE2(foo, bar, string, integer);
}
```

Tracing Tools

Goals

- Create uprobes
- Record events
- Observe variables
- Filter and transform events
- Aggregate stats
- Output

ftrace

- tracefs
  - /sys/kernel/tracing
- Supports uprobe, uretprobe
- Userspace can create uprobes using tracefs
- Record via a /sys/kernel/tracing/trace file
- Ring buffer

perf_events

- Create probes with perf_event_open syscall
- Create uprobes
- Can attach eBPF
- Also helps eBPF transfer data back to userspace

eBPF

- Run safe programs in kernel
  - Sandboxed
  - User bytecode + kernel JIT
  - Verifier
- Replaces classic BPF

eBPF for Tracing

- Attach to probes
- Do work without leaving kernel
  - counting, filtering, stats, transforms
- Helpers
  - maps
  - Read memory
  - Read/write perf events

eBPF for Tracing

```
struct perf_event_attr attr;
attr.type = 7; // /sys/devices/uprobe/type
attr.uprobe_path = ...;
attr.probe_offset = ...;

union bpf_attr bpf_attr;
attr.prog_type = BPF_TYPE_KPROBE;
attr.insns = ...;
attr.insns_cnt = ...;
...

int event_fd = perf_event_open(&attr, pid, ...);
int prog_fd = bpf(BPF_PROG_LOAD, &bpf_attr, ...);
ioctl(event_fd, PERF_EVENT_IOC_SET_BPF, prog_fd);
```

eBPF limitations

- 512 byte stack size
- complexity limit
  - 1M verified instructions
  - provably terminates
- hard to debug

How do I eBPF?

bpftrace

- Compile to eBPF
- Command-line utility
- Build complex tools

```
uprobe:libssl:SSL_read,
uprobe:libssl:SSL_write,
uprobe:libssl:SSL_do_handshake
{
    @start_ssl[tid] = nsecs;
    @func_ssl[tid] = func; // store for uretprobe
}

uretprobe:libssl:SSL_read,
uretprobe:libssl:SSL_write,
uretprobe:libssl:SSL_do_handshake
/@start_ssl[tid] != 0/
{
    $lat_us = (nsecs - @start_ssl[tid]) / 1000;
    if ((int8)retval >= 1) {
        @hist[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @stat[@func_ssl[tid]] = stats($lat_us);
    } else {
        @histF[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @statF[@func_ssl[tid]] = stats($lat_us);
    }
    delete(@start_ssl[tid]); delete(@func_ssl[tid]);
}
```

```
uprobe:libssl:SSL_read,
uprobe:libssl:SSL_write,
uprobe:libssl:SSL_do_handshake
{
    @start_ssl[tid] = nsecs;
    @func_ssl[tid] = func; // store for uretprobe
}
```

```
uretprobe:libssl:SSL_read,
uretprobe:libssl:SSL_write,
uretprobe:libssl:SSL_do_handshake
/@start_ssl[tid] != 0/
{
    $lat_us = (nsecs - @start_ssl[tid]) / 1000;
    if ((int8)retval >= 1) {
        @hist[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @stat[@func_ssl[tid]] = stats($lat_us);
    } else {
        @histF[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @statF[@func_ssl[tid]] = stats($lat_us);
    }
    delete(@start_ssl[tid]); delete(@func_ssl[tid]);
}
```

probes


```
uprobe:libssl:SSL_read,
uprobe:libssl:SSL_write,
uprobe:libssl:SSL_do_handshake
{
    @start_ssl[tid] = nsecs;
    @func_ssl[tid] = func; // store for uretprobe
}

uretprobe:libssl:SSL_read,
uretprobe:libssl:SSL_write,
uretprobe:libssl:SSL_do_handshake
/@start_ssl[tid] != 0/
{
    $lat_us = (nsecs - @start_ssl[tid]) / 1000;
    if ((int8)retval >= 1) {
        @hist[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @stat[@func_ssl[tid]] = stats($lat_us);
    } else {
        @histF[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @statF[@func_ssl[tid]] = stats($lat_us);
    }
    delete(@start_ssl[tid]); delete(@func_ssl[tid]);
}
```

action

```
uprobe:libssl:SSL_read,
uprobe:libssl:SSL_write,
uprobe:libssl:SSL_do_handshake
{
    @start_ssl[tid] = nsecs;
    @func_ssl[tid] = func; // store for uretprobe
}

uretprobe:libssl:SSL_read,
uretprobe:libssl:SSL_write,
uretprobe:libssl:SSL_do_handshake
/@start_ssl[tid] != 0/
{
    $lat_us = (nsecs - @start_ssl[tid]) / 1000;
    if ((int8)retval >= 1) {
        @hist[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @stat[@func_ssl[tid]] = stats($lat_us);
    } else {
        @histF[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @statF[@func_ssl[tid]] = stats($lat_us);
    }
    delete(@start_ssl[tid]); delete(@func_ssl[tid]);
}
```

```
uprobe:libssl:SSL_read,
uprobe:libssl:SSL_write,
uprobe:libssl:SSL_do_handshake
{
    @start_ssl[tid] = nsecs;
    @func_ssl[tid] = func; // store for uretprobe
}

uretprobe:libssl:SSL_read,
uretprobe:libssl:SSL_write,
uretprobe:libssl:SSL_do_handshake
/@start_ssl[tid] != 0/
{
    $lat_us = (nsecs - @start_ssl[tid]) / 1000;
    if ((int8)retval >= 1) {
        @hist[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @stat[@func_ssl[tid]] = stats($lat_us);
    } else {
        @histF[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @statF[@func_ssl[tid]] = stats($lat_us);
    }
    delete(@start_ssl[tid]); delete(@func_ssl[tid]);
}
```

condition

```
uprobe:libssl:SSL_read,
uprobe:libssl:SSL_write,
uprobe:libssl:SSL_do_handshake
{
    @start_ssl[tid] = nsecs;
    @func_ssl[tid] = func; // store for uretprobe
}

uretprobe:libssl:SSL_read,
uretprobe:libssl:SSL_write,
uretprobe:libssl:SSL_do_handshake
/@start_ssl[tid] != 0/
{
    $lat_us = (nsecs - @start_ssl[tid]) / 1000;
    if ((int8)retval >= 1) {
        @hist[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @stat[@func_ssl[tid]] = stats($lat_us);
    } else {
        @histF[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @statF[@func_ssl[tid]] = stats($lat_us);
    }
    delete(@start_ssl[tid]); delete(@func_ssl[tid]);
}
```

```
uprobe:libssl:SSL_read,
uprobe:libssl:SSL_write,
uprobe:libssl:SSL_do_handshake
{
    @start_ssl[tid] = nsecs;
    @func_ssl[tid] = func; // store for uretprobe
}

uretprobe:libssl:SSL_read,
uretprobe:libssl:SSL_write,
uretprobe:libssl:SSL_do_handshake
/@start_ssl[tid] != 0/
{
    $lat_us = (nsecs - @start_ssl[tid]) / 1000;
    if ((int8)retval >= 1) {
        @hist[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @stat[@func_ssl[tid]] = stats($lat_us);
    } else {
        @histF[@func_ssl[tid]] = lhist($lat_us, 0, 1000, 200);
        @statF[@func_ssl[tid]] = stats($lat_us);
    }
    delete(@start_ssl[tid]); delete(@func_ssl[tid]);
}
```

Availability

- 3.16: eBPF
- 4.3: uprobes
- 4.7: eBPF attached to uprobes

Tracers compared

Tracer	Mechanism	Where	Filter/ Transform	Frontend	uprobe	USDT
ftrace	tracefs	kernel	N/A	trace-cmd kernelshark	✅	❌
perf_events	syscall + ring buffer	kernel	eBPF	perf bpftrace	✅	✅
SystemTap	kernel module	kernel	custom module	stap	✅	✅
LTTng	user library	user	N/A?	GUI	✅	✅

Performance Overhead

	Attached	Overhead per hit
USDT	no	<1 ns
USDT	yes	~280 ns
uprobe	yes	~680 ns

Test environment: Intel i9-12900K, Ubuntu 22.04 LTS, kernel 5.18.0, bpftrace v0.14.0

Tracing with eBPF in ATS

Adding a USDT in ATS

- USDT already exists in ATS
  - apt install systemtap-sdt-dev
  - ./configure --enable-systemtap
  - ts/sdt.h
- ATS_PROBE(...) macros
  - probe name
  - arguments

Example

filtering by host and URL

```
case PARSE_RESULT_DONE:
  SMDebug("http", "done parsing client request header");

  host = t_state.hdr_info.client_request.host_get(&host_len);
  path = t_state.hdr_info.client_request.path_get(&path_len);
  ATS_PROBE5(httpsm, parse_result_done, host, host_len, path, path_len, sm_id);
```

```
usdt:/opt/ats/bin/traffic_server:ats_httpsm:parse_result_done
{
    // Get Host header and URL path from USDT arguments
    $host = str(arg0, arg1);
    if (arg2 != 0) {
        $path = str(arg2, arg3);
    }

    // Try to match
    if ($path == @filter_path && $host == @filter_host) {
        @smid = (int64)arg4;
        printf("matched sm_id: %d\n", arg4);
    }
}
```

Demo

Links

- https://github.com/iovisor/bpftrace
- https://github.com/moonchen/trafficserver/tree/usdt
- https://github.com/moonchen/bpftrace-ats

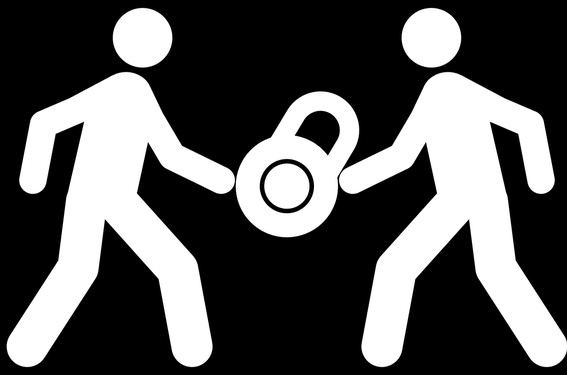
Tracing with uprobe

Motivation

Several way to profile Mutex Lock

- perf - sched:sched_switch [*1]
- systemtap - futexes.stp [*2]

  ...etc



*1: Performance Analysys Adn Tuning On Modern CPUs - Denis Bakhvalov
*2: https://sourceware.org/systemtap/examples/process/futexes.stp

Motivation

How about Mutex "Try" Lock?

```
class Example : public Continuation
{
public:
  int
  event_handler(int event, void *data)
  {
    EThread *ethread = this_ethread();

    MUTEX_TRY_LOCK(lock, this->mutex, ethread);
    if (!lock.is_locked()) {
        // retry event
        ethread->schedule_in_local(this, HRTIME_MSECONDS(10), event, data);
        return EVENT_DONE;
    }
    // do something
```

eBPF Tracing

	Static	Dynamic
Kernel	Tracepoints	kprobes
Userland	USDT	uprobes

bcc - uprobe/uretprobe

C and Python wrapper

```
BPF.attach_uprobe(name, sym, fn_name)
  - when: the tracing function (sym) is called
  - what: attach given C function (fn_name)
          you can read arguments with PT_REGS_PARM

BPF.attach_uretprobe(name, sym, fn_name)
  - when: the tracing function (sym) return
  - what: attach given C function (fn_name)
          you can read return value with PT_REGS_RC
```

PTHREAD_MUTEX_TRYLOCK(3)

```
NAME
     pthread_mutex_trylock – attempt to lock a mutex without blocking

SYNOPSIS
     #include <pthread.h>

     int
     pthread_mutex_trylock(pthread_mutex_t *mutex);

DESCRIPTION
     The pthread_mutex_trylock() function locks mutex.  If the mutex is already locked,
     pthread_mutex_trylock() will not block waiting for the mutex, but will return an error condition.

RETURN VALUES
     If successful, pthread_mutex_trylock() will return zero, otherwise an error number
     will be returned to indicate the error.

ERRORS
     The pthread_mutex_trylock() function will fail if:

     [EINVAL]           The value specified by mutex is invalid.

     [EBUSY]            Mutex is already locked.
```

Profiling pthread_mutex_trylock

```
from bcc import BPF

text = """
#include <linux/ptrace.h>

int probe_mutex_trylock_return(struct pt_regs *ctx) {
  if (PT_REGS_RC(ctx) != 0) {
    ++fail_count;
  }
  return 0;
}
"""

def attach(bpf, pid):
    bpf.attach_uretprobe(name="pthread", sym="pthread_mutex_trylock",
                         fn_name="probe_mutex_trylock_return", pid=pid)

def run(pid):
    attach(BPF(text), pid)
```

Profiling pthread_mutex_trylock

```
from bcc import BPF

text = """
#include <linux/ptrace.h>

int probe_mutex_trylock_return(struct pt_regs *ctx) {
  if (PT_REGS_RC(ctx) != 0) {
    ++fail_count;
  }
  return 0;
}
"""
```

python wrapper

```
def attach(bpf, pid):
    bpf.attach_uretprobe(name="pthread", sym="pthread_mutex_trylock",
                         fn_name="probe_mutex_trylock_return", pid=pid)

def run(pid):
    attach(BPF(text), pid)
```

Profiling pthread_mutex_trylock

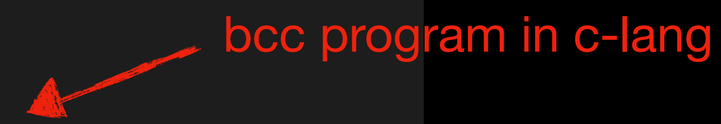
```
from bcc import BPF

text = """
#include <linux/ptrace.h>

int probe_mutex_trylock_return(struct pt_regs *ctx) {
  if (PT_REGS_RC(ctx) != 0) {
    ++fail_count;
  }
  return 0;
}
"""

def attach(bpf, pid):
    bpf.attach_uretprobe(name="pthread", sym="pthread_mutex_trylock",
                         fn_name="probe_mutex_trylock_return", pid=pid)

def run(pid):
    attach(BPF(text), pid)
```



Profiling ATS

Cache Enabled (almost 100% Hit)

```
thread 38875
        mutex [unknown] ::: wait time 0.00us ::: hold time 24692.07us ::: enter count 3051 ::: try-lock failure count 13693
                CacheVC::openReadStartHead(int, Event*)+0xd8 [traffic_server] (55e891616368)
                EThread::process_event(Event*, int)+0x276 [traffic_server] (55e89170b586)
                EThread::execute_regular()+0x33d [traffic_server] (55e89170bf0d)
                EThread::execute()+0x171 [traffic_server] (55e89170c361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (55e89170a8a5)
                start_thread+0xc5 [libpthread-2.17.so] (7f1216cf8ea5)

        mutex [unknown] ::: wait time 0.00us ::: hold time 10263.61us ::: enter count 1372 ::: try-lock failure count 6294
                CacheVC::openReadClose(int, Event*)+0xab [traffic_server] (55e89161a1db)
                EThread::process_event(Event*, int)+0x276 [traffic_server] (55e89170b586)
                EThread::execute_regular()+0x33d [traffic_server] (55e89170bf0d)
                EThread::execute()+0x171 [traffic_server] (55e89170c361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (55e89170a8a5)
                start_thread+0xc5 [libpthread-2.17.so] (7f1216cf8ea5)

        mutex [unknown] ::: wait time 0.00us ::: hold time 3041.71us ::: enter count 337 ::: try-lock failure count 1391
                Cache::open_read(Continuation*, ats::CryptoHash const*, HTTPHdr*, OverridableHttpConfigParams const*, CacheFra
                CacheProcessor::open_read(Continuation*, HttpCacheKey const*, HTTPHdr*, OverridableHttpConfigParams const*, lo
                HttpCacheSM::open_read(HttpCacheKey const*, URL*, HTTPHdr*, OverridableHttpConfigParams const*, long)+0x9d [tr
                HttpSM::do_cache_lookup_and_read()+0x169 [traffic_server] (55e89147d929)
                HttpSM::set_next_state()+0x4d1 [traffic_server] (55e8914820f1)
                HttpSM::set_next_state()+0xb48 [traffic_server] (55e891482768)
                HttpSM::set_next_state()+0xa06 [traffic_server] (55e891482626)
```

Profiling ATS

Cache Enabled (almost 100% Hit)

```
thread 38875
        mutex [unknown] ::: wait time 0.00us ::: hold time 24692.07us ::: enter count 3051 ::: try-lock failure count 13693
                CacheVC::openReadStartHead(int, Event*)+0xd8 [traffic_server] (55e891616368)
                EThread::process_event(Event*, int)+0x276 [traffic_server] (55e89170b586)
                EThread::execute_regular()+0x33d [traffic_server] (55e89170bf0d)
                EThread::execute()+0x171 [traffic_server] (55e89170c361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (55e89170a8a5)
                start_thread+0xc5 [libpthread-2.17.so] (7f1216cf8ea5)

        mutex [unknown] ::: wait time 0.00us ::: hold time 10263.61us ::: enter count 1372 ::: try-lock failure count 6294
                CacheVC::openReadClose(int, Event*)+0xab [traffic_server] (55e89161a1db)
                EThread::process_event(Event*, int)+0x276 [traffic_server] (55e89170b586)
                EThread::execute_regular()+0x33d [traffic_server] (55e89170bf0d)
                EThread::execute()+0x171 [traffic_server] (55e89170c361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (55e89170a8a5)
                start_thread+0xc5 [libpthread-2.17.so] (7f1216cf8ea5)

        mutex [unknown] ::: wait time 0.00us ::: hold time 3041.71us ::: enter count 337 ::: try-lock failure count 1391
                Cache::open_read(Continuation*, ats::CryptoHash const*, HTTPHdr*, OverridableHttpConfigParams const*, CacheFra
                CacheProcessor::open_read(Continuation*, HttpCacheKey const*, HTTPHdr*, OverridableHttpConfigParams const*, lo
                HttpCacheSM::open_read(HttpCacheKey const*, URL*, HTTPHdr*, OverridableHttpConfigParams const*, long)+0x9d [tr
                HttpSM::do_cache_lookup_and_read()+0x169 [traffic_server] (55e89147d929)
                HttpSM::set_next_state()+0x4d1 [traffic_server] (55e8914820f1)
                HttpSM::set_next_state()+0xb48 [traffic_server] (55e891482768)
                HttpSM::set_next_state()+0xa06 [traffic_server] (55e891482626)
```

Profiling ATS

Cache Enabled (almost 100% Hit)

```
thread 38875
        mutex [unknown] ::: wait time 0.00us ::: hold time 24692.07us ::: enter count 3051 ::: try-lock failure count 13693
                CacheVC::openReadStartHead(int, Event*)+0xd8 [traffic_server] (55e891616368)
                EThread::process_event(Event*, int)+0x276 [traffic_server] (55e89170b586)
                EThread::execute_regular()+0x33d [traffic_server] (55e89170bf0d)
                EThread::execute()+0x171 [traffic_server] (55e89170c361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (55e89170a8a5)
                start_thread+0xc5 [libpthread-2.17.so] (7f1216cf8ea5)
```

```
int
CacheVC::openReadStartHead(int event, Event *e)
{
  ...
  {
    CACHE_TRY_LOCK(lock, vol->mutex, mutex->thread_holding);
    if (!lock.is_locked()) {
      VC_SCHED_LOCK_RETRY();
    }
```

https://github.com/apache/trafficserver/blob/c983006eccbce9365224c2cd30372528ac8df843/iocore/cache/CacheRead.cc#L1069

```
ailure count 6294

lure count 1391
rams const*, CacheFra
nfigParams const*, lo
onst*, long)+0x9d [tr

                HttpSM::set_next_state()+0xb48 [traffic_server] (55e891482768)
                HttpSM::set_next_state()+0xa06 [traffic_server] (55e891482626)
```

Profiling ATS

Cache Disabled

```
thread 44144
        mutex [unknown] ::: wait time 0.00us ::: hold time 8658.67us ::: enter count 1987 ::: try-lock failure count 2981
                HostDBProcessor::getby(Continuation*, void (Continuation::*)(HostDBRecord*), HostDBHash&, HostDBProcessor::Opt
                HostDBProcessor::getbyname_imm(Continuation*, void (Continuation::*)(HostDBRecord*), char const*, int, HostDBP
                HttpSM::do_hostdb_lookup()+0x3f7 [traffic_server] (561f63d43df7)
                HttpSM::set_next_state()+0xab7 [traffic_server] (561f63d4b6d7)
                HttpSM::set_next_state()+0xb48 [traffic_server] (561f63d4b768)
                HttpSM::set_next_state()+0xa06 [traffic_server] (561f63d4b626)
                HttpSM::set_next_state()+0xb48 [traffic_server] (561f63d4b768)
                HttpSM::set_next_state()+0xb48 [traffic_server] (561f63d4b768)
                HttpSM::state_read_client_request_header(int, void*)+0xe8d [traffic_server] (561f63d27b7d)
                HttpSM::main_handler(int, void*)+0xe6 [traffic_server] (561f63d25cf6)
                HttpSM::state_add_to_list(int, void*)+0x1ef [traffic_server] (561f63d260af)
                HttpSM::attach_client_session(ProxyTransaction*)+0x512 [traffic_server] (561f63d26ca2)
                Http1ClientSession::new_transaction()+0x58 [traffic_server] (561f63d0cef8)
                Http1ClientSession::state_keep_alive(int, void*)+0xa0 [traffic_server] (561f63d0c900)
                read_signal_and_update(int, UnixNetVConnection*)+0x1ed [traffic_server] (561f63fb3d1d)
                UnixNetVConnection::net_read_io(NetHandler*, EThread*)+0x5ac [traffic_server] (561f63fb283c)
                NetHandler::process_ready_list()+0x32f [traffic_server] (561f63fa12af)
                NetHandler::waitForActivity(long)+0x6a4 [traffic_server] (561f63fa1dc4)
                non-virtual thunk to NetHandler::waitForActivity(long)+0xd [traffic_server] (561f63fa1e9d)
                EThread::execute_regular()+0x51f [traffic_server] (561f63fd50ef)
                EThread::execute()+0x171 [traffic_server] (561f63fd5361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (561f63fd38a5)
                start_thread+0xc5 [libpthread-2.17.so] (7fdf97293ea5)
```

Profiling ATS

Cache Disabled

```
thread 44144
        mutex [unknown] ::: wait time 0.00us ::: hold time 8658.67us ::: enter count 1987 ::: try-lock failure count 2981
                HostDBProcessor::getby(Continuation*, void (Continuation::*)(HostDBRecord*), HostDBHash&, HostDBProcessor::Opt
                HostDBProcessor::getbyname_imm(Continuation*, void (Continuation::*)(HostDBRecord*), char const*, int, HostDBPr
                HttpSM::do_hostdb_lookup()+0x3f7 [traffic_server] (561f63d43df7)
                HttpSM::set_next_state()+0xab7 [traffic_server] (561f63d4b6d7)
```

```
Action *
HostDBProcessor::getby(Continuation *cont, cb_process_result_pfn cb_process_result, HostD
&hash, Options const &opt)
{
  ...
      MUTEX_TRY_LOCK(lock2, bucket_mutex, thread);
      if (lock2.is_locked()) {
```

https://github.com/apache/trafficserver/blob/c983006eccbce9365224c2cd30372528ac8df843/iocore/hostdb/HostDB.cc#L713

```
                NetHandler::waitForActivity(long)+0x6a4 [traffic_server] (561f63fa1dc4)
                non-virtual thunk to NetHandler::waitForActivity(long)+0xd [traffic_server] (561f63fa1e9d)
                EThread::execute_regular()+0x51f [traffic_server] (561f63fd50ef)
                EThread::execute()+0x171 [traffic_server] (561f63fd5361)
                spawn_thread_internal(void*)+0x55 [traffic_server] (561f63fd38a5)
                start_thread+0xc5 [libpthread-2.17.so] (7fdf97293ea5)
```

Summary

- Userspace Tracing with eBPF
  - USDT : User Statically-Defined Tracing
  - uprobe : User Level Dynamic Tracing

Work to do

- Add more USDT tracepoints to ATS
- Associate tracepoint with a transaction
  - sm_id, connection_id, etc.
  - propagate it to cache calls, hostdb calls, etc
- Write tracing scripts