Optimizing OpenSIPS Performance

Vlad Paiu
OpenSIPS Summit 2025

Optimizing OpenSIPS Performance

- Why ?
- Practical Strategies for

  Optimizing OpenSIPS Performance
- Takeaways

Why optimize ?

Why optimize ?

OpenSIPS offers the foundation for high performance

- Written in C, fast :
  - SIP parsing
  - SIP processing & manipulation
  - core modules (TM, Dialog)
- Multi-process and asynchronous architecture
- Lean & Modular architecture

BUT…

Why optimize ?

- Suboptimal scripts can severely degrade OpenSIPS performance
- Production environments are different than a controlled lab
- Unexpected traffic spikes can overwhelm untuned systems

Performance Optimisation Strategies


I WILL CACHE YOU
AND I WILL FIND YOU

Use the Right Modules for the Right Job

Many OpenSIPS modules use startup caching & runtime reload

- Domain
- Permissions
- Dispatcher
- Load Balancer
- Drouting

Use the Right Modules for the Right Job

Wrong :

```
sql_query("SELECT client_id from my_clients where
ip_address='$si'","$avp(my_client_id)");

if ($rc > 0) {

        xlog("IP $si belongs to client $avp(my_client_id) \n");

}
```

Use the Right Modules for the Right Job

Right :

```
loadmodule “permissions.so”

if (get_source_group( $avp(my_client_id) ) {
    xlog("IP $si belongs to client $avp(my_client_id) \n");
}
```

Use the Right Modules for the Right Job

- Always question yourself if you are implementing things the right way
- Sometimes you might need to use DB Views to bring the data to the format that OpenSIPS knows to cache at startup

Cache all DB queries

- Sometimes you have custom tables that really don’t fit into the predefined OpenSIPS tables

- Do runtime caching for them

Cache all DB queries - Manual Approach

```
if (cache_fetch(“local”,”client_id_$avp(client_id)”,$avp(client_attr)) {
        xlog(“Fetched attr $avp(client_attr) for $avp(client_id) from cache\n");
} else {
        sql_query("SELECT client_attr from my_clients where id='$avp(client_id'","$avp(client_attr)");
        if ($rc > 0) {
                # cache it forever
                cache_store("local","client_id_$avp(client_id)","$avp(client_attr)",0);
                xlog(“Fetched attr $avp(client_attr) for $avp(client_id) from DB \n");
        }
}
# For reloading : opensips-cli -x mi cache_remove local client_id_1
```

Cache all DB queries - SQL_Cacher Approach

```
modparam("sql_cacher", "cache_table",
"id=my_clients
db_url=mysql://root:opensips@localhost/opensips_3_6
cachedb_url=local://
table=my_clients
key=id
columns=client_attr
on_demand=1")

$avp(client_attr) = $sql_cached_value(my_clients:client_attr:$avp(client_id));

# For reloading : opensips-cli -x mi sql_cacher_reload my_clients 1
```

Try to Cache All External Dependencies

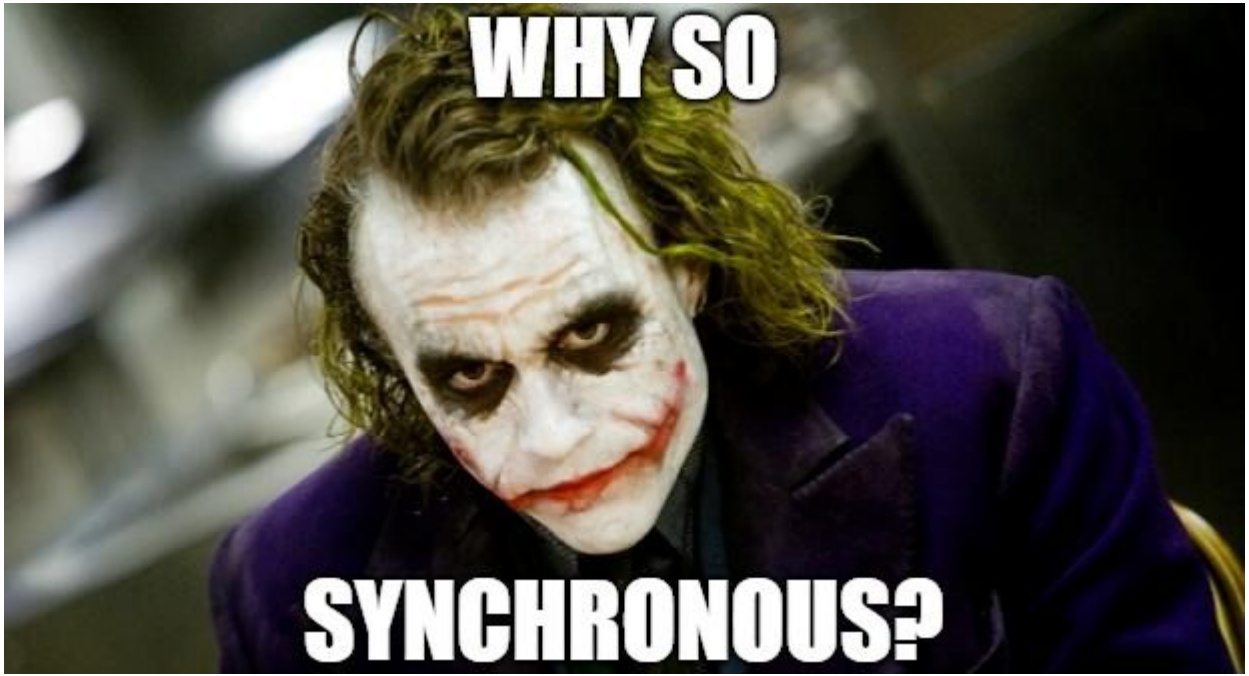
- HTTP queries
  - Manual Script caching approach
- DNS queries

```
loadmodule “dns_cache.so”

modparam("dns_cache", "min_ttl",300) # new in 3.6
```

Try not to shoot yourself in the foot

- Maintain a balance between observability and performance
  - Avoid having a too verbose log
  - If you using Homer to replicate traffic, do it over UDP and do not use a DNS hostname for Homer itself
  - Monitor Homer aggressively and disable replicating if it’s down
    - opensips-cli -x mi trace mode=off

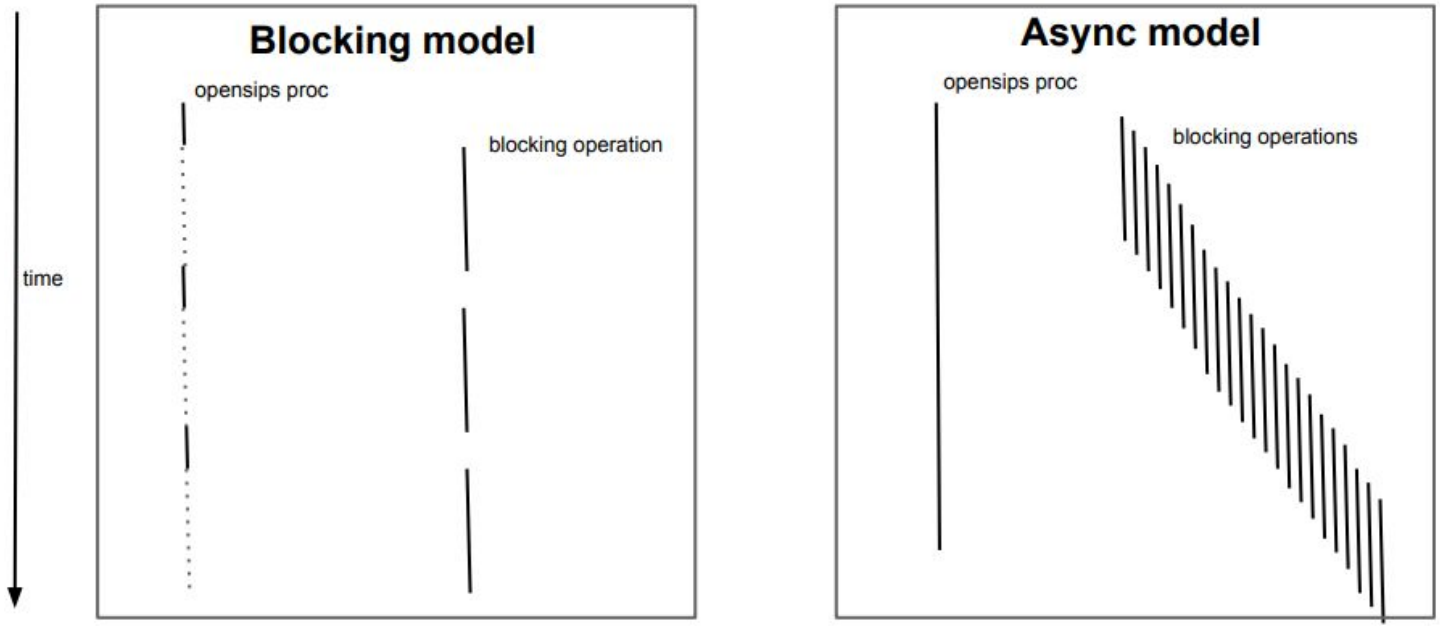

WHY SO
SYNCHRONOUS?

Asynchronous OPS in the Script

- **Aim for your OpenSIPS script to become CPU bound**

- **You never want to blockingly wait for an external dependency**
  - OpenSIPS processes will eventually get stuck waiting for the result
  - UDP/TCP queues will fill up
  - Packets & calls will start to drop

Asynchronous OPS in the Script



Asynchronous OPS in the Script

- dramatically improves scalability of OpenSIPS scripts
- better resilience to external service failures
- better usage of system resources

OpenSIPS waiting for RTPEngine

Asynchronous OPS in the Script

New in OpenSIPS 3.6

- **Run async statements in onreply_route**

- **Async RTPEngine requests**

Asynchronous RTPEngine Flow

```
route {
...
async(rtpengine_offer("$var(offer_flags)",$var(rtpengine_socket),$avp(sdp_body)),"RESUME_RTPENGINE_OFFER",2);
}
route[RESUME_RTPENGINE_OFFER] {
        if ($avp(sdp_body) != NULL) {
                # apply the changes in the SDP
                route(relay);
        } else {
                # failed to engage RTPEngine
                t_reply(503,"Unavailable");
                exit;
        }
}
```

Asynchronous RTPEngine Flow

```
onreply_route[reply_processing] {

...

async(rtpengine_answer("$var(answer_flags)",,$avp(r_sdp_body)),"RESUME_RTPENGINE_ANSWER",2);

}


route[RESUME_RTPENGINE_ANSWER] {

        if ($avp(r_sdp_body) != NULL) {

                # apply the changes in the SDP

                # default action in reply context is for it to be relayed

        } else {

                # failed to engage rtpengine

                t_cancel_branch();

        }

}
```

Asynchronous RTPEngine Flow

```
if (is_method("BYE")) {
        # launch the rtpengine_delete() and don’t care whether it succeeds for not
        launch(rtpengine_delete());
}
```

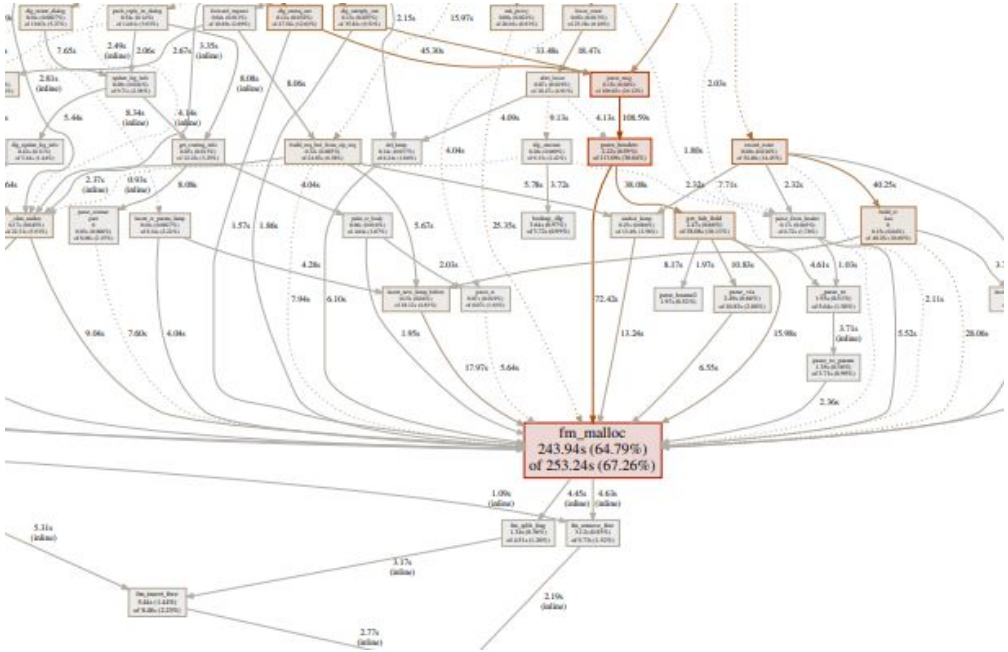
My script is now CPU bound !


SHM LOCK
OPENSIPS CPU BOUND SCRIPT
imgflip.com

OpenSIPS Shared Memory Allocator

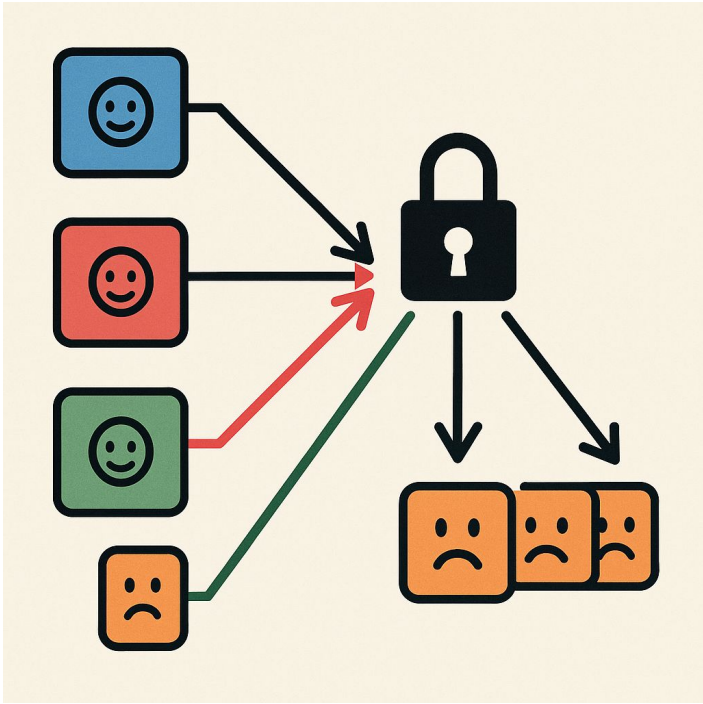
- The shared memory allocator uses a global lock
- In a high CPS & high CPUs environment, in combination with a very complex OpenSIPS script, this will lead to contention & starvation

OpenSIPS Shared Memory Allocator



https://www.opensips.org/About/PerformanceTests-3-4

OpenSIPS Shared Memory Allocator



OpenSIPS Shared Memory Allocator

- Try to avoid contention
  - Use proper variable scope
    - $var is per process allocated memory
    - The rest ($avp, $dlg_var, $shv) are allocated in shared memory
  - When DB querying, if you know you'll get a single row :
    - Use sql_select_one(), which allows you to store output in a $var

OpenSIPS New Shared Memory Allocator

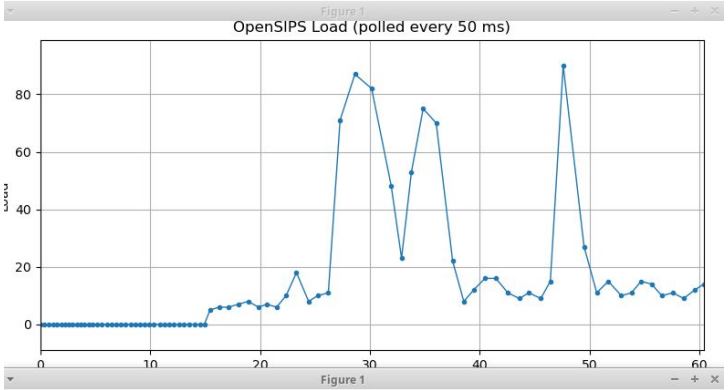
- OpenSIPS 3.6 adds a new F_PARALLEL_MALLOC allocator
  - Splits the total shared memory size into 32 separate memory chunks
  - Each memory chunk has its own allocator lock
  - When a process allocates shared memory it picks a random memory pool
  - When a process frees up memory, it puts it back in the original memory pool
  - Can be set to be used at runtime via :
    - `opensips -f etc/opensips.cfg -m 4096 -a F_PARALLEL_MALLOC -k F_MALLOC`

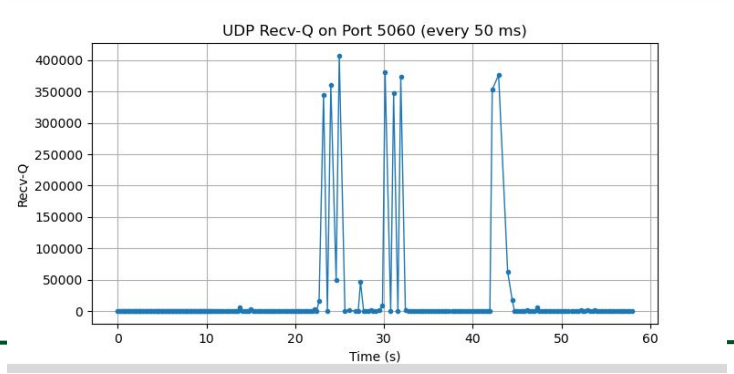
OpenSIPS New Shared Memory Allocator

OpenSIPS New Shared Memory Allocator

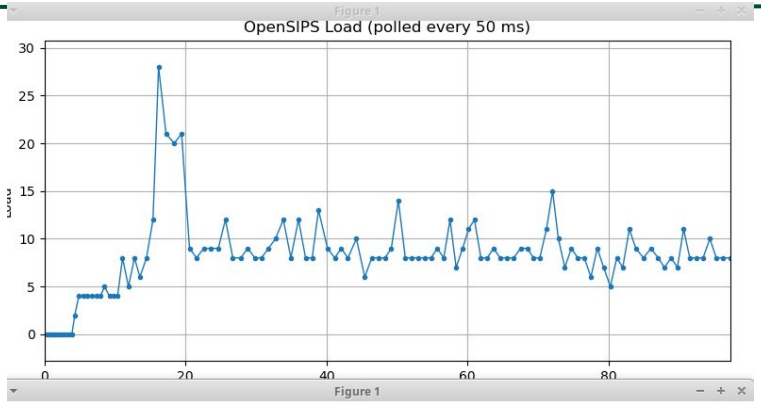
- Testing with a TM/DIALOG/ACC memory intensive script :
  - Setting 50 different $AVPs at INVITE time
  - Setting 20 different $dlg_vals at INVITE time
  - Pushing 50 values to ACC extra fields
- i7-1355U 6 cores / 12 threads
- No logging, SIPP with call duration of 10 seconds, running on the same box as OpenSIPS
- Plotting the load:load and UDP buffer queue size

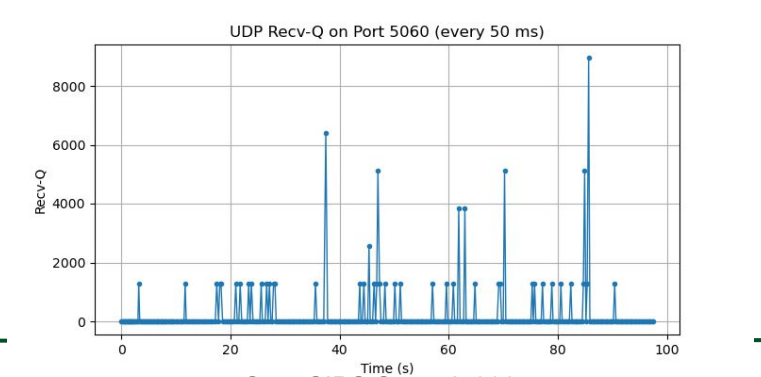
F_MALLOC - 1800 CPS



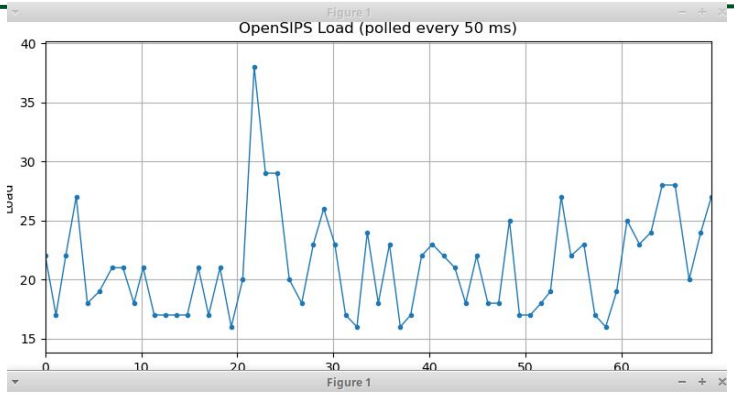


F_PARALLEL_MALLOC - 1800 CPS





F_PARALLEL_MALLOC - 3000 CPS



OpenSIPS New Shared Memory Allocator

- F_PARALLEL_MALLOC allows you to squeeze ~ 2x more CPS compared to F_MALLOC in high contention/starvation environments

- Much more resilient to temporary spikes and retransmission floods

What else can we improve ?

Memory footprint & Reload Times

- What about the memory footprint ?
- What about the reload times
  - How fast can you actually apply changes?

Caching 10 Million 10D prefixes

- Use drouting module for caching the prefixes

- SHM Memory Usage :
  - 10 GB ; peak memory usage 20GB when doing a dr_reload
- dr_reload time on a local SQL :
  - 140 seconds

Caching 10 Million 10D prefixes

- OpenSIPS 3.6 adds a new trie module
  - https://opensips.org/html/docs/modules/3.6.x/trie.html
- SHM Memory Usage :
  - 4 GB = ~ 2.5x less memory usage vs drouting prefix caching ; 5x less memory usage since there is no need for a full reload to apply changes
- Deleting numbers is instantaneous :
  - `opensips-cli -x mi trie_number_delete partition_name=dids number=["1234567890","0987654321"]`
- Adding numbers is instantaneous :
  - `opensips-cli -x mi trie_number_upsert partition_name=dids number=["1234","5678"] attrs=["1234attrs","5678attrs"]`

Takeaways

Takeaways

- Performance is in the hands of the OpenSIPS user/script writer

- OpenSIPS 3.6 has many improvements which allow you to squeeze as much performance as possible

Questions ?

- Vlad Paiu
  - OpenSIPS Project: www.opensips.org
  - Email: vladpaiu@opensips.org