LNet Observability via Packet Tracing

Timothy Day

Expertise | Innovation | Partnership

What is Lustre doing?
And why?

Lustre Protocol

Although some documentation exists [1], the Lustre protocol is not formally specified. To understand what Lustre RPCs are sent to service a system call, you must (at present) read the Lustre source code.

[1] https://git.whamcloud.com/?p=doc/protocol.git;a=tree

How do we improve LNet observability?

We need request tracing!

… but existing solutions are not great.

Static trace points can be performance intensive and cannot be changed without a Lustre update

Modern tools like bfptrace still require some Lustre knowledge to use effectively

Tools like tcpdump are not network-type agnostic

Native LNet Packet Capture

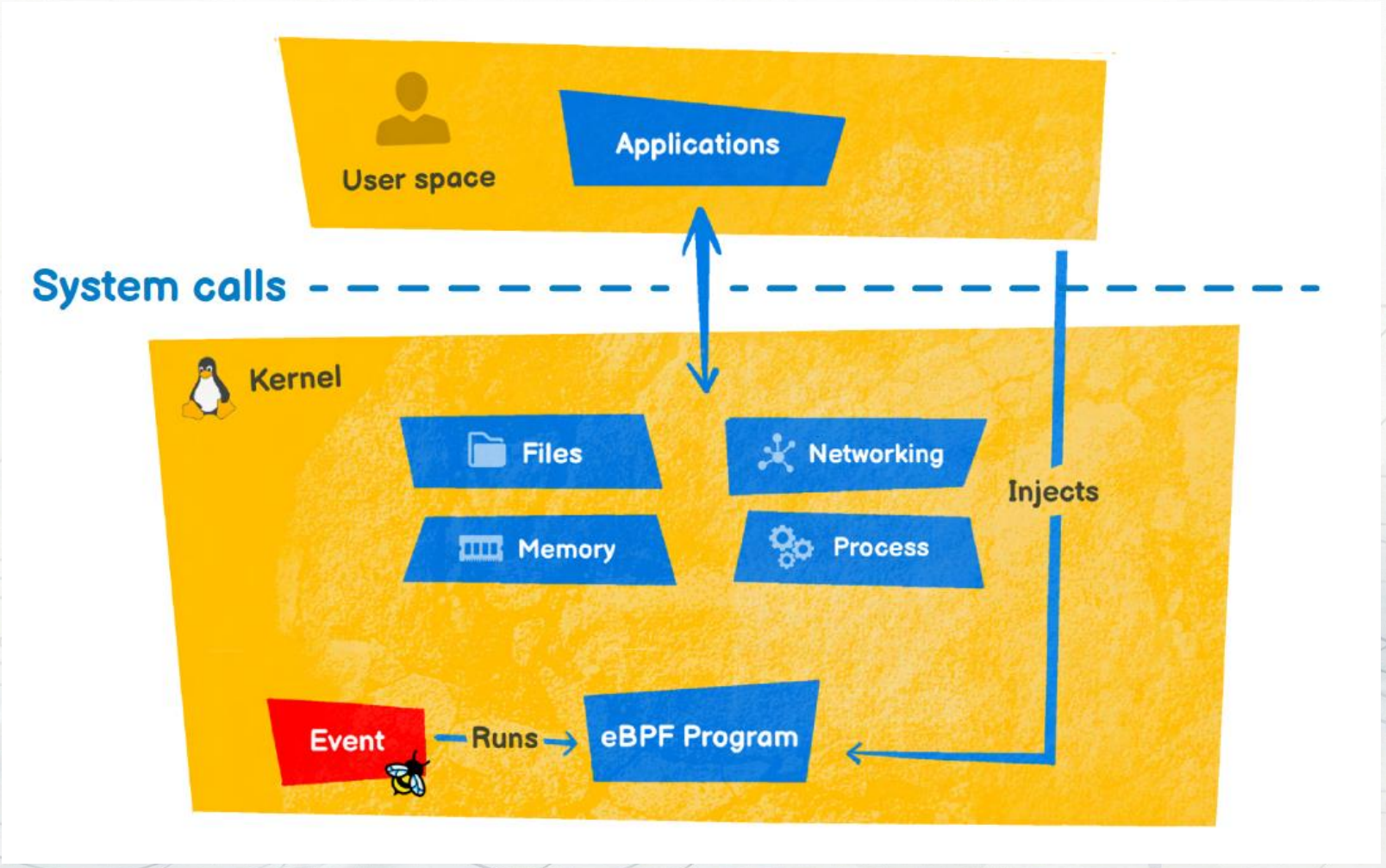
Leveraging eBPF for LNet packet capture

Implement custom eBPF hooks and tcpdump-style packet capture tool for LNet that can be run on production systems.

How does eBPF work?

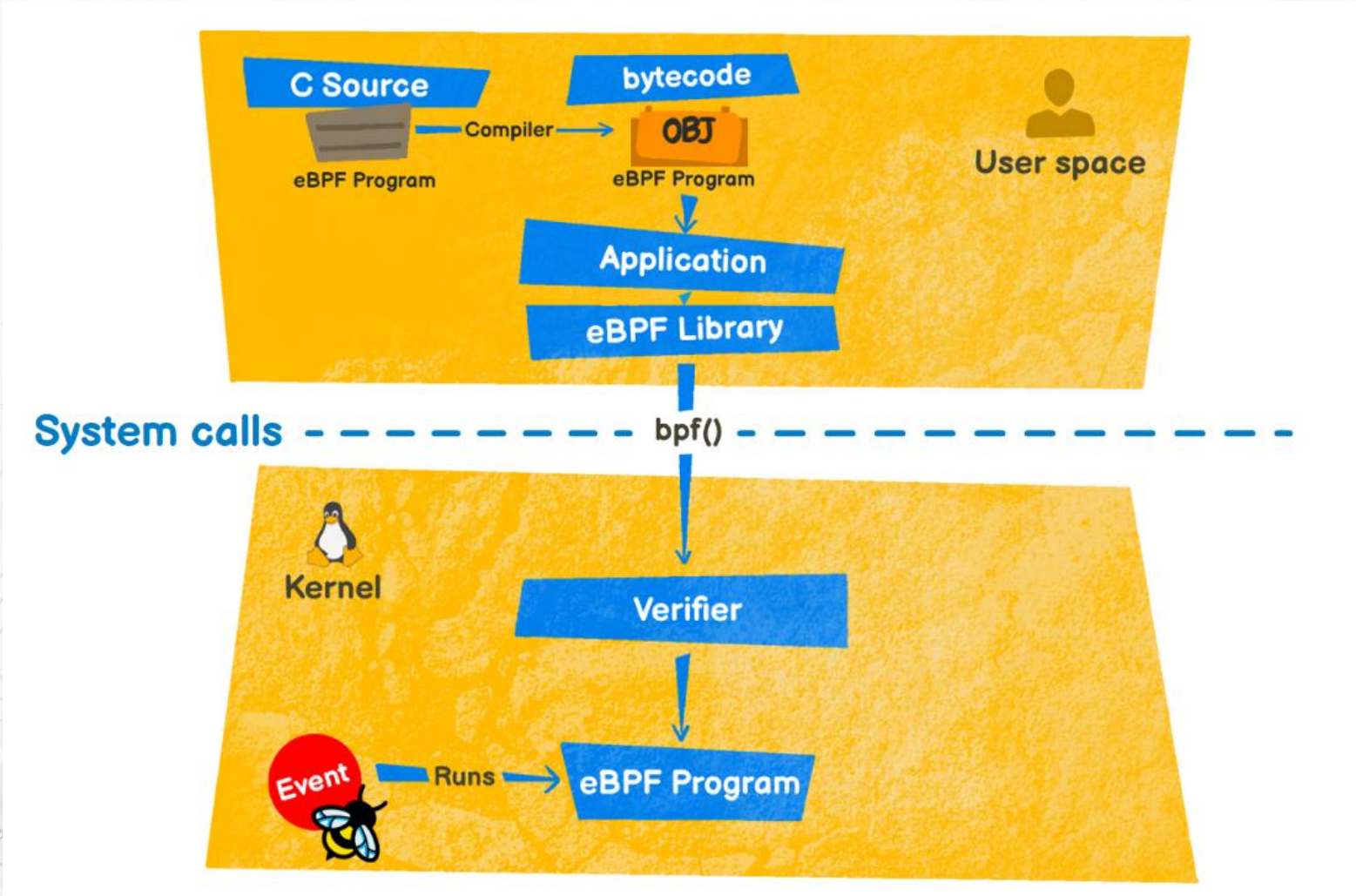
eBPF implements a lightweight virtual machine that allows userspace to safely read kernel data structures.

How does eBPF work?



[1] https://ebpf.io/books/buzzing-across-space-illustrated-childrens-guide-to-ebpf.pdf

How does eBPF work?

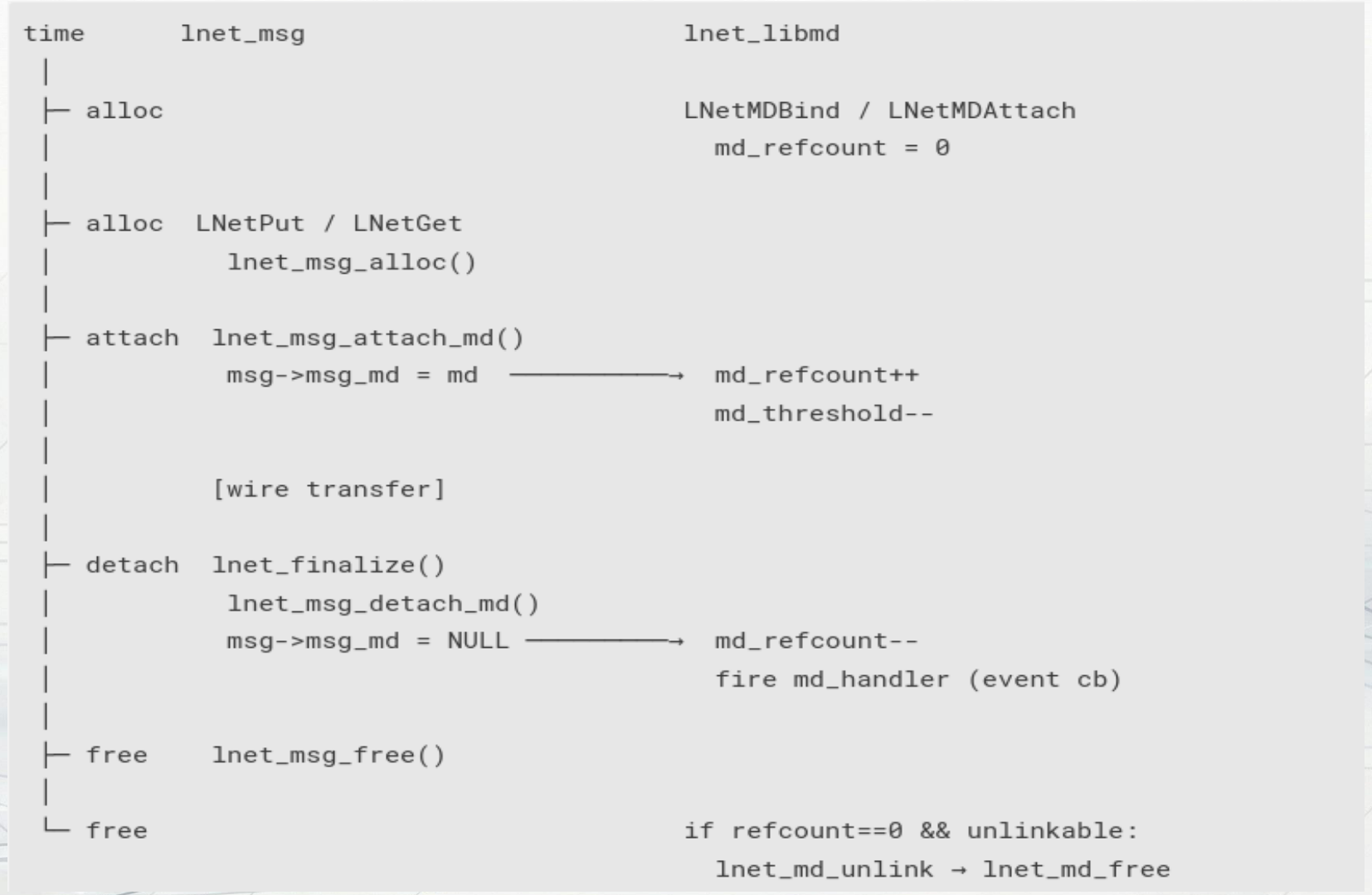


[1] https://ebpf.io/books/buzzing-across-space-illustrated-childrens-guide-to-ebpf.pdf

How does an LNet message work?

An LNet message is a combination of a `struct lnet_msg` and `struct lnet_md`. The former describes the message, and the latter holds its contents.

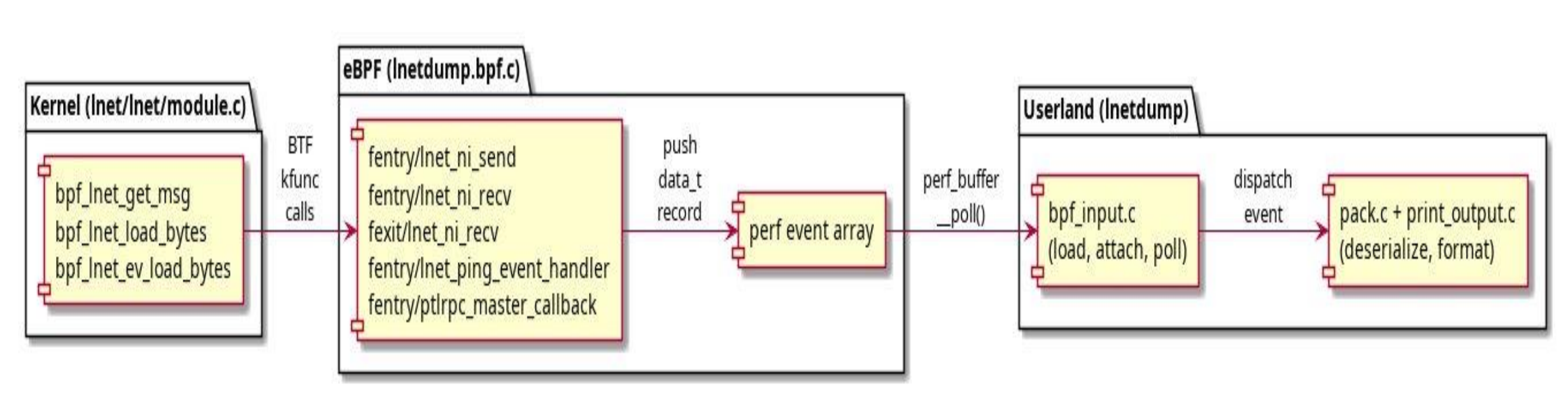
How does an LNet message work?



Implementing an eBPF Hook to Read LNet Messages

The kernel will provide a new interface to copy an LNet messages' content to an eBPF buffer. The eBPF program create a ring buffer that will be populated by messages captured at predefined entry points. The userspace tool will read and parse the messages from the ring buffer.

Implementing an eBPF Hook to Read LNet Messages



Features and Future Plans...

A more powerful CLI for understanding Lustre RPCs

Create a simple, pluggable tool for understanding the Lustre protocol.

- Basic packet capture and text output should be ready for 2.18
- Tool can be extended to emit metrics, binary logs, sequence charts, etc.
- Implement performant packet filtering

Powerful CLI for tracing Lustre RPCs

```
sudo lnetdump -p "%-5c [%-15Lm] %-5Ls %-5Lt %-20LP [%-16Pj] %-30Po"
...
26660 [0x69F14CE4BFE80] SEND  PUT   OST_IO                [kworker.0        ] OST_READ
26661 [0x69F14CE4BFE80] RECV  PUT   OST_IO                [kworker.0        ] OST_READ
26662 [0x69F14CE4BFE80] SEND  PUT   OSC_REPLY             [EMPTY            ] OST_READ
26663 [0x69F14CE4BFE80] RECV  PUT   OSC_REPLY             [EMPTY            ] OST_READ
26664 [0x69F14CE4BFF00] SEND  PUT   LDLM_CANCEL_REQUEST   [ldlm_bl.0        ] LDLM_CANCEL
26665 [0x69F14CE4BFF00] RECV  PUT   LDLM_CANCEL_REQUEST   [ldlm_bl.0        ] LDLM_CANCEL
26666 [0x69F14CE4BFF00] SEND  PUT   LDLM_CANCEL_REPLY     [EMPTY            ] LDLM_CANCEL
26667 [0x69F14CE4BFF00] RECV  PUT   LDLM_CANCEL_REPLY     [EMPTY            ] LDLM_CANCEL
26668 [0x69F14CE4BFF80] SEND  PUT   MDS_REQUEST           [systemd-timesyn.996] LDLM_ENQUEUE
26669 [0x69F14CE4BFF80] RECV  PUT   MDS_REQUEST           [systemd-timesyn.996] LDLM_ENQUEUE
26670 [0x69F14CE4BFF80] SEND  PUT   MDC_REPLY             [EMPTY            ] LDLM_ENQUEUE
26671 [0x69F14CE4BFF80] RECV  PUT   MDC_REPLY             [EMPTY            ] LDLM_ENQUEUE
26672 [0x69F14CE4C0000] SEND  PUT   OST_REQUEST           [systemd-timesyn.996] LDLM_ENQUEUE
26673 [0x69F14CE4C0000] RECV  PUT   OST_REQUEST           [systemd-timesyn.996] LDLM_ENQUEUE
26674 [0x69F14CE4C0000] SEND  PUT   OSC_REPLY             [EMPTY            ] LDLM_ENQUEUE
26675 [0x69F14CE4C0000] RECV  PUT   OSC_REPLY             [EMPTY            ] LDLM_ENQUEUE
26676 [0x69F14CE4C0080] SEND  PUT   OST_IO                [kworker.0        ] OST_READ
26677 [0x69F14CE4C0080] RECV  PUT   OST_IO                [kworker.0        ] OST_READ
26678 [0x69F14CE4C0080] SEND  PUT   OSC_REPLY             [EMPTY            ] OST_READ
26679 [0x69F14CE4C0080] RECV  PUT   OSC_REPLY             [EMPTY            ] OST_READ
26680 [0x69F14CE4C0100] SEND  PUT   OST_IO                [kworker.0        ] OST_READ
26681 [0x69F14CE4C0100] RECV  PUT   OST_IO                [kworker.0        ] OST_READ
26682 [0x69F14CE4C0100] SEND  PUT   OSC_REPLY             [EMPTY            ] OST_READ
26683 [0x69F14CE4C0100] RECV  PUT   OSC_REPLY             [EMPTY            ] OST_READ
26684 [0x69F14CE4C0180] SEND  PUT   OST_IO                [kworker.0        ] OST_READ
26685 [0x69F14CE4C0180] RECV  PUT   OST_IO                [kworker.0        ] OST_READ
26686 [0x69F14CE4C0180] SEND  PUT   OSC_REPLY             [EMPTY            ] OST_READ
26687 [0x69F14CE4C0180] RECV  PUT   OSC_REPLY             [EMPTY            ] OST_READ
26688 [0x69F14CE4C0200] SEND  PUT   MDS_REQUEST           [systemd-timesyn.996] LDLM_ENQUEUE
26689 [0x69F14CE4C0200] RECV  PUT   MDS_REQUEST           [systemd-timesyn.996] LDLM_ENQUEUE
26690 [0x69F14CE4C0200] SEND  PUT   MDC_REPLY             [EMPTY            ] LDLM_ENQUEUE
26691 [0x69F14CE4C0200] RECV  PUT   MDC_REPLY             [EMPTY            ] LDLM_ENQUEUE
26692 [0x69F14CE4C0300] SEND  PUT   MDS_REQUEST           [systemd-timesyn.996] MDS_REINT
26693 [0x69F14CE4C0300] RECV  PUT   MDS_REQUEST           [systemd-timesyn.996] MDS_REINT
26694 [0x69F14CE4C0300] SEND  PUT   MDC_REPLY             [EMPTY            ] MDS_REINT
26695 [0x69F14CE4C0300] RECV  PUT   MDC_REPLY             [EMPTY            ] MDS_REINT
```

RPC-level Lustre Regression Testing

Assert that Lustre performs certain operations when executing system calls.

Hypothetical example:

lnetdump --filter msg_type=PUT,portal=MDS_REQUEST --assert count=4 mycommand

How much is actually synced to the server when flushing caches?

How often do we reacquire the same locks when executing a command?

Are we sending unneeded RPCs?

Thanks!

Find out how The Lustre Collective can empower your AI and HPC storage vision

thelustrecollective.com

info@thelustrecollective.com