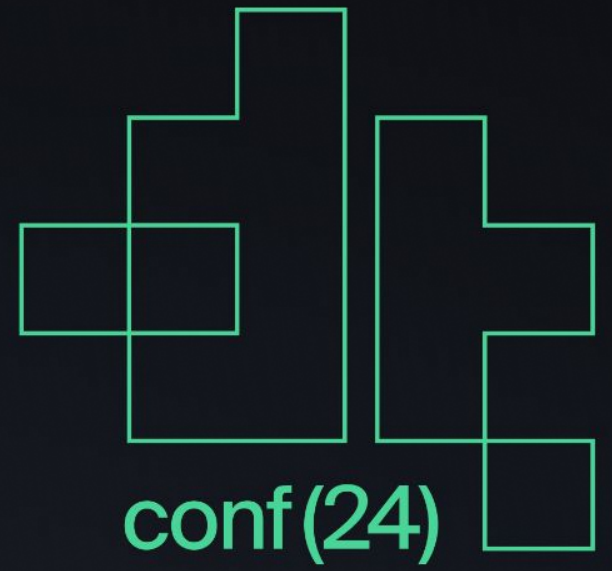
DError: Banishing `Debug::fmt` for nested enums



Kyle Simpson
kyle@oxide.computer

Context

- OPTE – our packet processing stack for virtual machines.
  - Kernel module.
- DTrace gives us a lot of value here
  - Statically Defined Tracing (SDT) probes used due to: name mangling, unpredictable inlining.
  - SDTs show port final decisions, per-layer processing, parse failures, serialisation failures.
- DTrace key for debugging & development.

Processing Result

```rust
#[derive(Debug)]
pub enum ProcessResult {
    Bypass,
    Drop { reason: DropReason },
    Modified,
    Hairpin(Packet<Initialized>),
}
```

```rust
#[derive(Clone, Debug)]
pub enum DropReason {
    HandlePkt,
    Layer { name: &'static str, reason: layer::DenyReason },
    TcpErr,
}
```

```rust
/// Why a given packet was denied.
#[derive(Clone, Copy, Debug, Eq, PartialEq)]
pub enum DenyReason {
    /// The packet was denied by the action itself.
    ///
    /// For example, a hairpin action might decide it can't parse the
    /// body of the packet it's attempting to respond to.
    Action,

    /// The packet was denied by the default action.
    ///
    /// In this case the packet matched no rules and the
    /// [`DefaultAction`] was taken for the given direction.
    Default,

    /// The packet was denied by a rule.
    ///
    /// The packet matched a [`Rule`] and that rule's action was
    /// [`Action::Deny`].
    Rule,
}
```

Parsing Errors

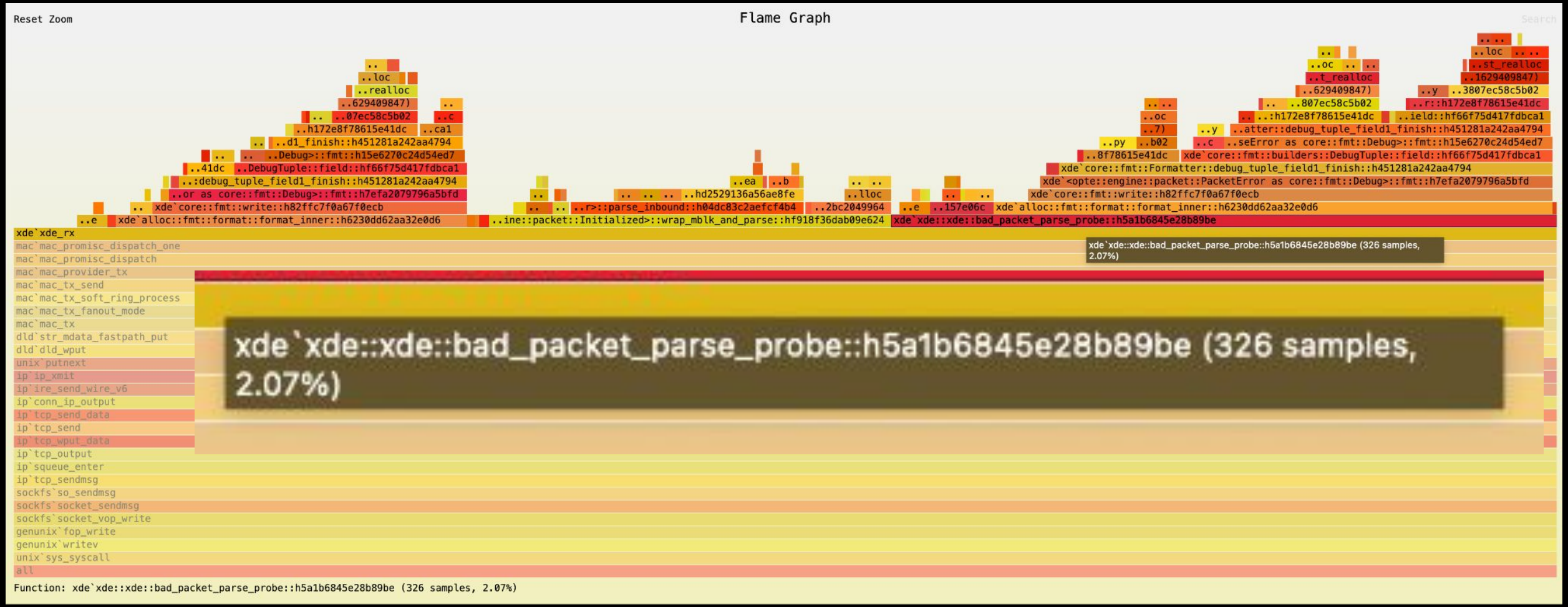
```rust
#[derive(Clone, Debug)]
pub enum PacketError {
    Parse(ParseError),
    Wrap(WrapError),
}
```

```rust
pub enum WrapError {
    /// We tried to wrap a NULL pointer.
    NullPtr,
}
```

```rust
#[derive(Clone, Debug, Eq, PartialEq)]
pub enum ParseError {
    BadHeader(String),
    BadInnerIpLen { expected: usize, actual: usize },
    BadInnerUlpLen { expected: usize, actual: usize },
    BadOuterIpLen { expected: usize, actual: usize },
    BadOuterUlpLen { expected: usize, actual: usize },
    BadRead(ReadErr),
    TruncatedBody { expected: usize, actual: usize },
    UnexpectedEtherType(super::ether::EtherType),
    UnsupportedEtherType(u16),
    UnexpectedProtocol(Protocol),
    UnexpectedDestPort(u16),
    UnsupportedProtocol(Protocol),
}
```

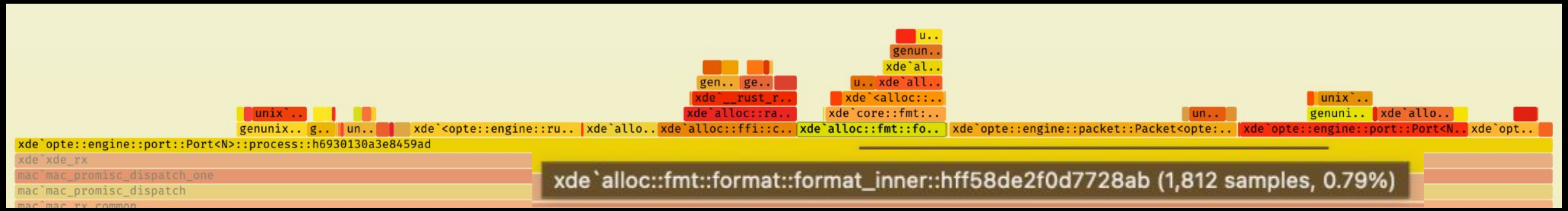
```rust
#[derive(Clone, Copy, Debug, Eq, PartialEq)]
pub enum ReadErr {
    BadLayout,
    EndOfPacket,
    NotEnoughBytes,
    OutOfRange,
    StraddledRead,
    NotImplemented,
}
```

Excess work – dropped packets (underlay)



[Suspicious amount of time spent in `bad_packet_probe` · Issue #458 · oxidecomputer/opte · GitHub]

Excess work – the ‘fast’ path



That’s roughly the same time as we take to re-emit a packet!

Suspicious amount of time spent in `bad_packet_probe` · Issue #458 · oxidecomputer/opte · GitHub

So, what's happening?

- Calling SDTs themselves is basically free.
- Getting data into the right shape *is not*.
  - **Certainly not for every packet we admit.**
- Idiomatic Rust errors encourage storing chains like this.
  - As operators, root causes are useful!
  - Not `#[repr(C)]` friendly, and we don't want to manually decode in D script for every new `enum`.

```
port-process-return {
        this->dir = DIR_STR(arg0);
        this->name = stringof(arg1);
        this->flow_before = (flow_id_sdt_arg_t *)arg2;
        this->flow_after = (flow_id_sdt_arg_t *)arg3;
        this->epoch = arg4;
        this->mp = (mblk_t *)arg5;
        /* If the result is a hairpin packet, then hp_mp is non-NULL. */
        this->hp_mp = (mblk_t *)arg6;
        this->res = stringof(arg7);

        if (num >= 10) {
                printf(HDR_FMT, "NAME", "DIR", "EPOCH", "FLOW BEFORE",
                    "FLOW AFTER", "LEN", "RESULT");
                num = 0;
        }

        this->af = this->flow_before->af;

        if (this->af != AF_INET && this->af != AF_INET6) {
                printf("BAD ADDRESS FAMILY: %d\n", this->af);
        }
}
```

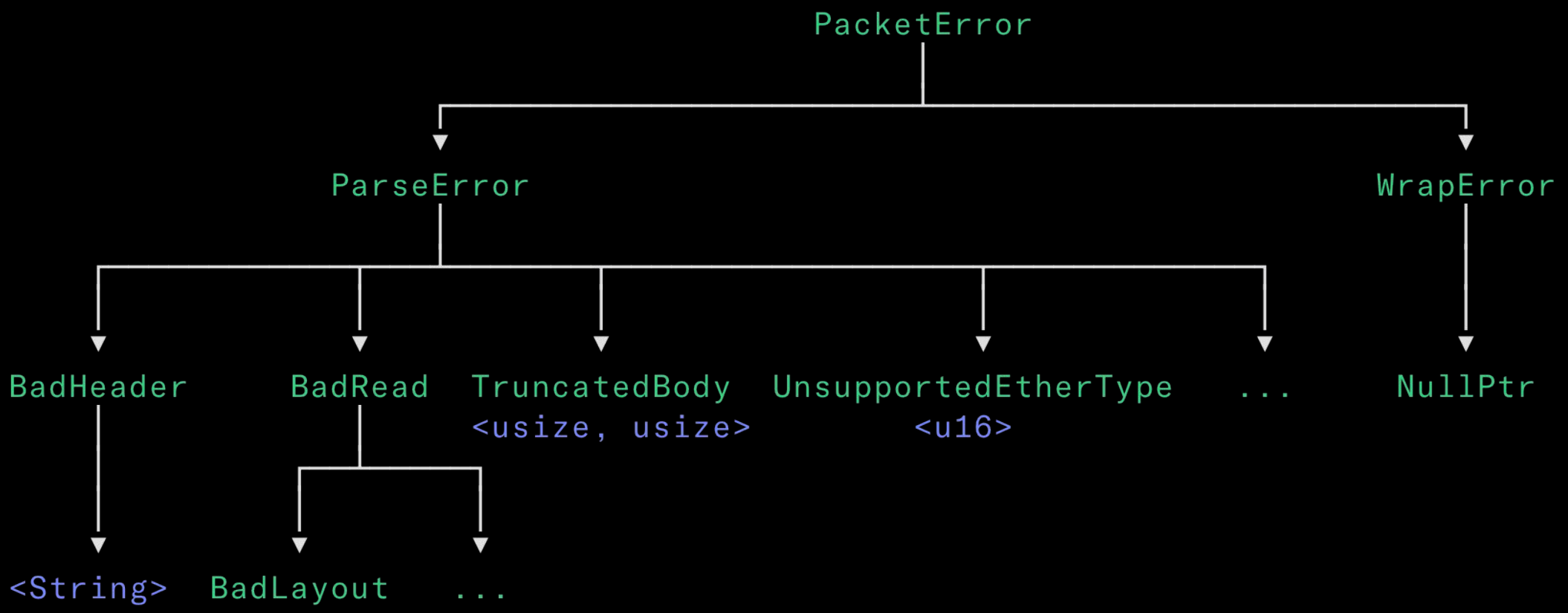
Thinking it through

```
ProcessResult::Drop{reason: Layer {name: “nat”, reason: Rule}}
```

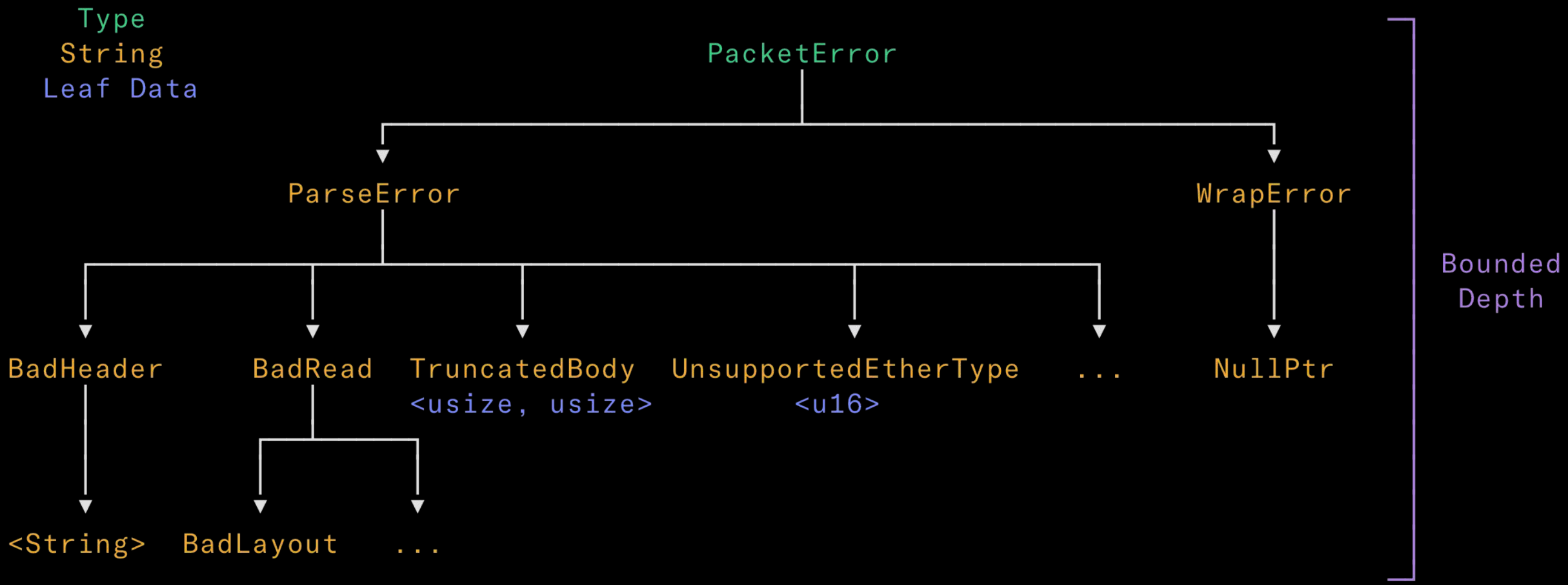
…it’s unwieldy, but it gives us a lot of information we want.

- Manually decoding in DTrace is out of the question.
  - What happens when we reach another library’s error type?
  - Brittle under change.
- Flattening out into a ‘god error’ similarly brittle.
- No `is_enabled` for us – we’re in the illumos kernel.

Mapping out an error



Mapping out an error



```rust
pub trait DError {
    /// Provide the name of an error's discriminant.
    fn discriminant(&self) -> &'static CStr;

    /// Provide a reference to the next error in the chain.
    fn child(&self) -> Option<&dyn DError>;

    /// Store data from a leaf error to be bundled with a probe.
    fn leaf_data(&self, _data: &mut [u64]) {}
}

static EMPTY_STRING: &CStr = c"";

#[derive(Debug)]
#[repr(C)]
pub struct LabelBlock<const L: usize> {
    len: usize,
    more: bool,
    data: [u64; 2],
    entries: [*const i8; L],
}
```

DError

- Any error/result must provide a label, and the next node.
- When needed for an SDT, we fill a `LabelBlock`.
  - Compile-time fixed storage.
  - Push discriminant, move to child, write leaf data if terminal.
- `more` denotes a chain deeper than `L`.
- We can still push our own `String` at the last step.

```rust
#[derive(Clone, Debug, Eq, PartialEq, DError)]
#[derror(leaf_data = ParseError::data)]
pub enum ParseError {
    BadHeader(HeaderReadErr),
    BadInnerIpLen {
        expected: usize,
        actual: usize,
    },
    BadInnerUlpLen {
        expected: usize,
        actual: usize,
    },
    BadOuterIpLen {
        expected: usize,
        actual: usize,
    },
    BadOuterUlpLen {
        expected: usize,
        actual: usize,
    },
    BadRead(ReadErr),
    TruncatedBody {
        expected: usize,
        actual: usize,
    },
    #[leaf]
    UnexpectedEtherType(super::ether::EtherType),
```

```rust
impl ParseError {
    fn data(&self, data: &mut [u64]) {
        match self {
            Self::BadInnerIpLen { expect
            | Self::BadInnerUlpLen { exp
            | Self::BadOuterIpLen { expe
            | Self::BadOuterUlpLen { exp
            | Self::TruncatedBody { expected, actual } => {
                [data[0], data[1]] = [*expected as u64, *actual as u64]
            }
            Self::UnexpectedEtherType(eth) => data[0] = u16::from(*eth).into(),
            Self::UnsupportedEtherType(eth) => data[0] = *eth as u64,
            Self::UnexpectedProtocol(proto) => {
                data[0] = u8::from(*proto).into()
            }
            Self::UnexpectedDestPort(port) => data[0] = (*port).into(),
            Self::UnsupportedProtocol(proto) => {
                data[0] = u8::from(*proto).into()
            }

            _ => {}
        }
    }
}
```

DError

- Proc-macro for most of our types – `#[derive(DError)]`.
  - Automatically generate, e.g., c“`BadHeader`”, child walker for tuple variants .
  - `#[leaf]` annotation for terminal tuple variants.
- Made it easy to convert `BadHeader(String)` to its true type: `HeaderReadErr`.
- Leaf data fn handled explicitly.

DError in practice

- DTrace needed to handle this is slightly awkward.
  - But it does not change, and is opaque to error implementation.
  - Pattern reusable for all error types.
- Output is readable and captures the full chain.

```
#define LINE_FMT    "%-12s %-3s 0x%-16p %s[%d, %d]\n"
#define EL_DELIMIT  "->"
#define EL_FMT      "->%s"

bad-packet
/this->msg_len > 0/
{
    this->res = strjoin(this->res, stringof(this->msgs->entry[0]));
}

bad-packet
/this->msg_len > 1/
{
    this->res = strjoin(this->res, EL_DELIMIT);
    this->res = strjoin(this->res, stringof(this->msgs->entry[1]));
}

bad-packet
/this->msg_len > 2/
{
    this->res = strjoin(this->res, EL_DELIMIT);
    this->res = strjoin(this->res, stringof(this->msgs->entry[2]));
}
```

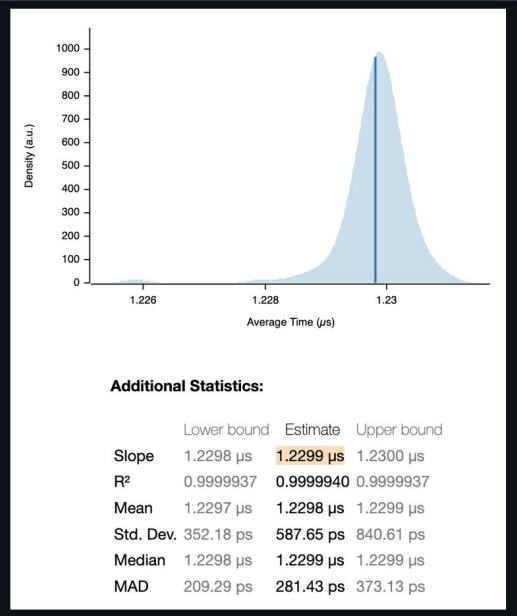
```
bad-packet {
    printf(LINE_FMT,
        this->port, this->dir, this->mblk,
        this->res, this->msgs->data[0], this->msgs->data[1]
    );
}
```

```
PORT         DIR MBLK              MSG+DATA
unknown      IN  0xffffe6a0e799420 Parse->UnexpectedEtherType [35020, 0]
unknown      IN  0xffffe6a042c6480 Parse->UnexpectedProtocol [58, 0]
unknown      IN  0xffffe6b8ab774c0 Parse->UnexpectedEtherType [35020, 0]
unknown      IN  0xffffe6a1250e1a0 Parse->UnexpectedEtherType [35020, 0]
unknown      IN  0xffffe6a0e7dbd60 Parse->UnexpectedEtherType [35020, 0]
unknown      IN  0xffffe6a0e509da0 Parse->UnexpectedEtherType [35020, 0]
unknown      IN  0xffffe69e9238700 Parse->UnexpectedEtherType [16724, 0]
unknown      IN  0xffffe6b5e089c40 Parse->UnexpectedEtherType [16724, 0]
unknown      IN  0xffffe6a0eedea60 Parse->BadHeader->BadLength { len: 4 } [0, 0]
```

Performance

For unexpected packets:

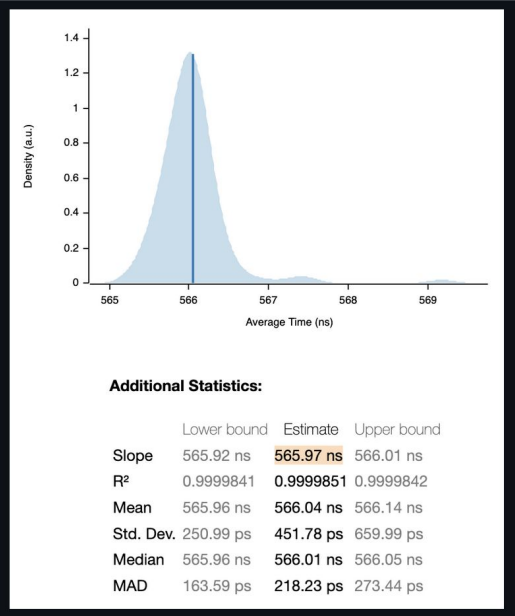
Before



Additional Statistics:

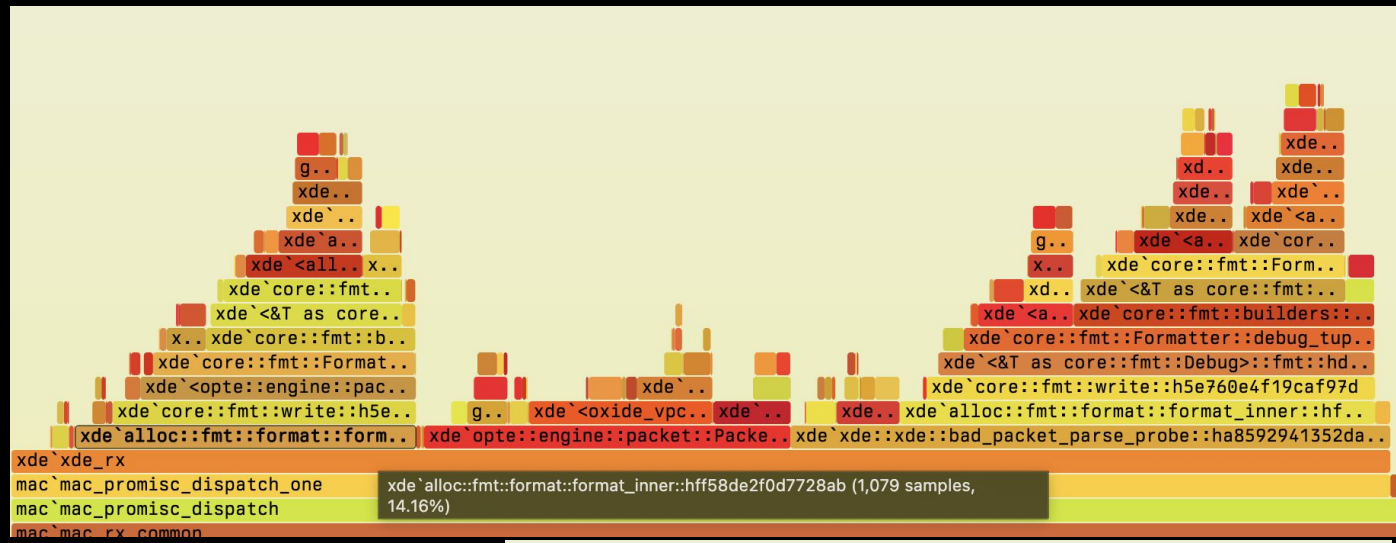
	Lower bound	Estimate	Upper bound
Slope	1.2298 µs	1.2299 µs	1.2300 µs
R²	0.9999937	0.9999940	0.9999937
Mean	1.2297 µs	1.2298 µs	1.2299 µs
Std. Dev.	352.18 ps	587.65 ps	840.61 ps
Median	1.2298 µs	1.2299 µs	1.2299 µs
MAD	209.29 ps	281.43 ps	373.13 ps

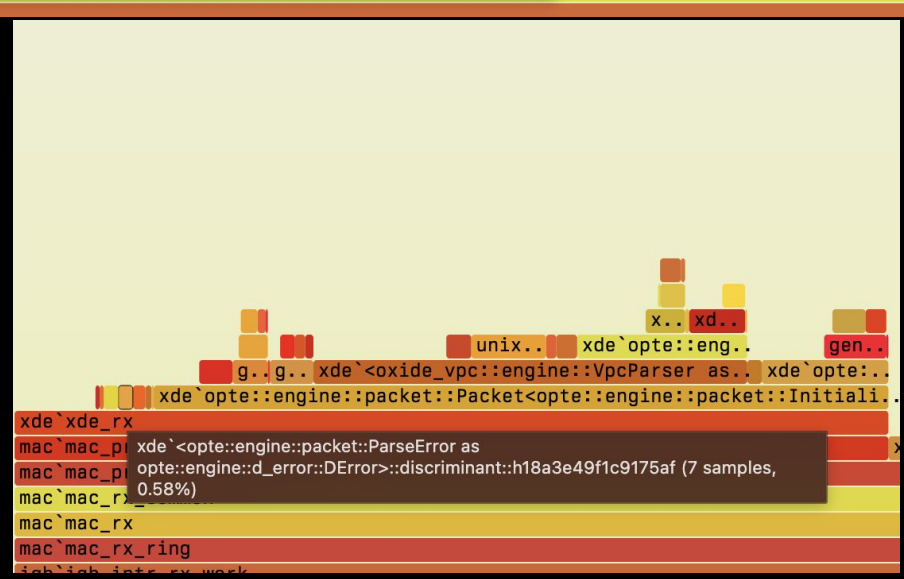
After



Additional Statistics:

	Lower bound	Estimate	Upper bound
Slope	565.92 ns	565.97 ns	566.01 ns
R²	0.9999841	0.9999851	0.9999842
Mean	565.96 ns	566.04 ns	566.14 ns
Std. Dev.	250.99 ps	451.78 ps	659.99 ps
Median	565.96 ns	566.01 ns	566.05 ns
MAD	163.59 ps	218.23 ps	273.44 ps





Plus a minor speed boost for fastpath packets.

Rework bad packet notification for dtrace probes · Pull request #459 · oxidecomputer/opte · GitHub

Post-parser stack rewrite

```
unknown      IN  0xfffffe69e0d48be0 IngotError->outer_udp->Unwanted[0, 0]
unknown      IN  0xfffffe69e0d48be0 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69eb3028a0 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69e9b525a0 IngotError->outer_udp->Reject[0, 0]
unknown      IN  0xfffffe69ec7b55e0 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69fdc3ae20 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69e0d481e0 IngotError->outer_udp->Reject[0, 0]
unknown      IN  0xfffffe69eb3027a0 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69e1473ae0 IngotError->outer_v6->Unwanted[0, 0]
PORT         DIR MBLK               MSG+DATA
unknown      IN  0xfffffe69e30d41e0 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69fdc3a020 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69ee4801a0 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69fdc3a060 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69e9b525a0 IngotError->outer_udp->Reject[0, 0]
unknown      IN  0xfffffe69ec283e80 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69e91540e0 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69e0d481e0 IngotError->outer_udp->Reject[0, 0]
unknown      IN  0xfffffe69e296a600 IngotError->outer_v6->Unwanted[0, 0]
unknown      IN  0xfffffe69fdc3a1e0 IngotError->outer_v6->Unwanted[0, 0]
```

```rust
impl ParseError {
    /// Return the name of the error variant as a [`CStr`].
    #[inline]
    pub fn as_cstr(&self) -> &'static CStr {
        match self {
            ParseError::Unwanted => c"Unwanted",
            ParseError::NeedsHint => c"NeedsHint",
            ParseError::TooSmall => c"TooSmall",
            ParseError::StraddledHeader => c"StraddledHeader",
            ParseError::NoRemainingChunks => c"NoRemainingChunks",
            ParseError::CannotAccept => c"CannotAccept",
            ParseError::Reject => c"Reject",
            ParseError::IllegalValue => c"IllegalValue",
        }
    }
}
```

```rust
impl DError for PacketParseError {
    #[inline]
    fn discriminant(&self) -> &'static core::ffi::CStr {
        self.header().as_cstr()
    }

    #[inline]
    fn child(&self) -> Option<&dyn DError> {
        Some(self.error())
    }
}

impl DError for ingot::types::ParseError {
    #[inline]
    fn discriminant(&self) -> &'static core::ffi::CStr {
        self.as_cstr()
    }

    #[inline]
    fn child(&self) -> Option<&dyn DError> {
        None
    }
}
```

Future work?

- Variable-length / generic data.
- Encoding max depth into DError trait.
  - Compile-time assurance that `LabelBlock` is large enough.
- Less `&dyn` when filling `LabelBlocks`.

Main lesson: make sure your SDTs have `#[repr(C)]`, cheap inputs.

- See also: *OPTE#475*.

Thank you!

dt conf(24)