Building keymashed

Mashing the keyboard to make your network *faster*

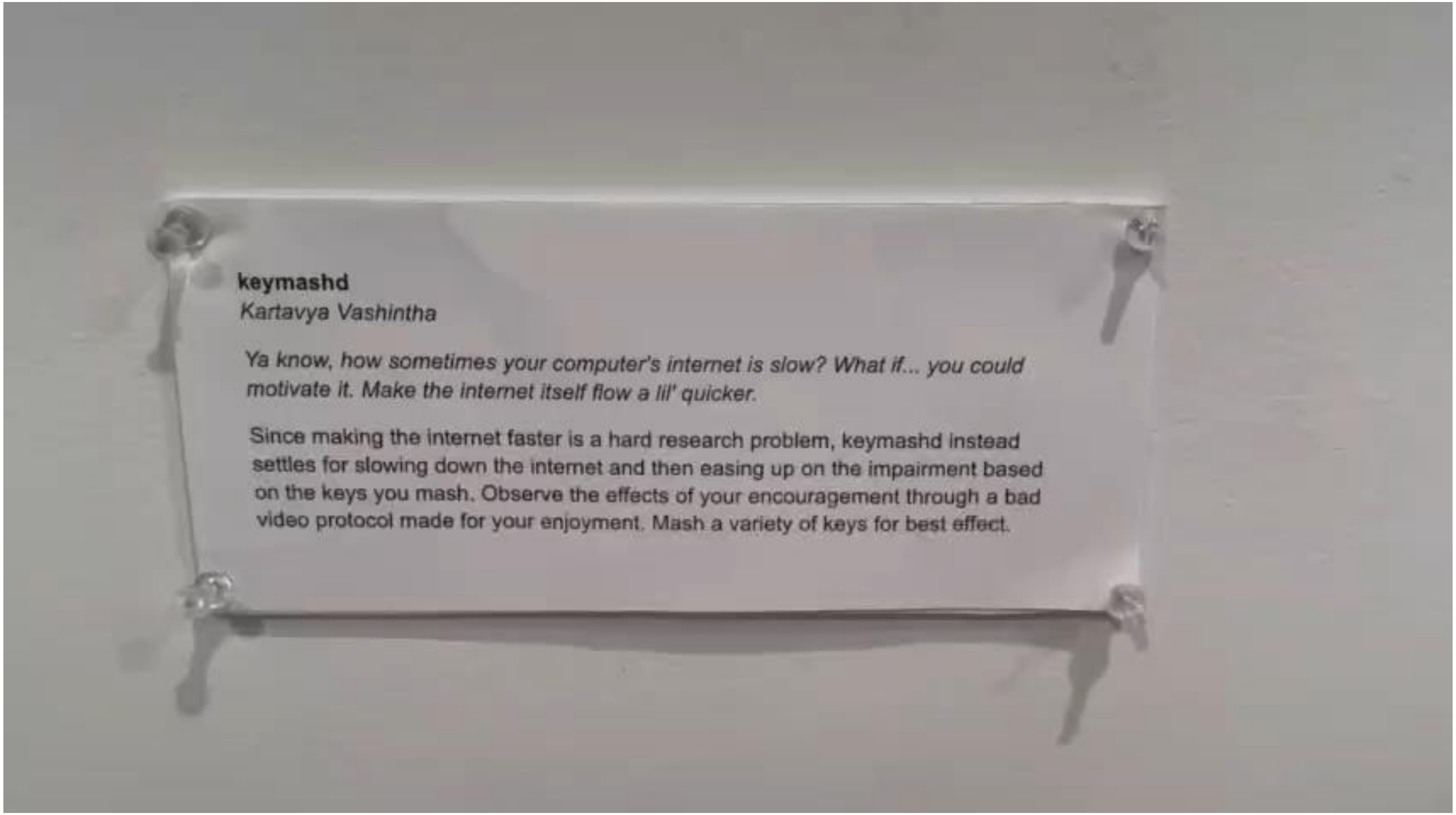
(by reducing packet loss)

Background

Keymashed is an interactive art installation.

There's a livestream on the monitor. Mashing the keyboard:

- reduces packet loss.
- decreases lossy compression.

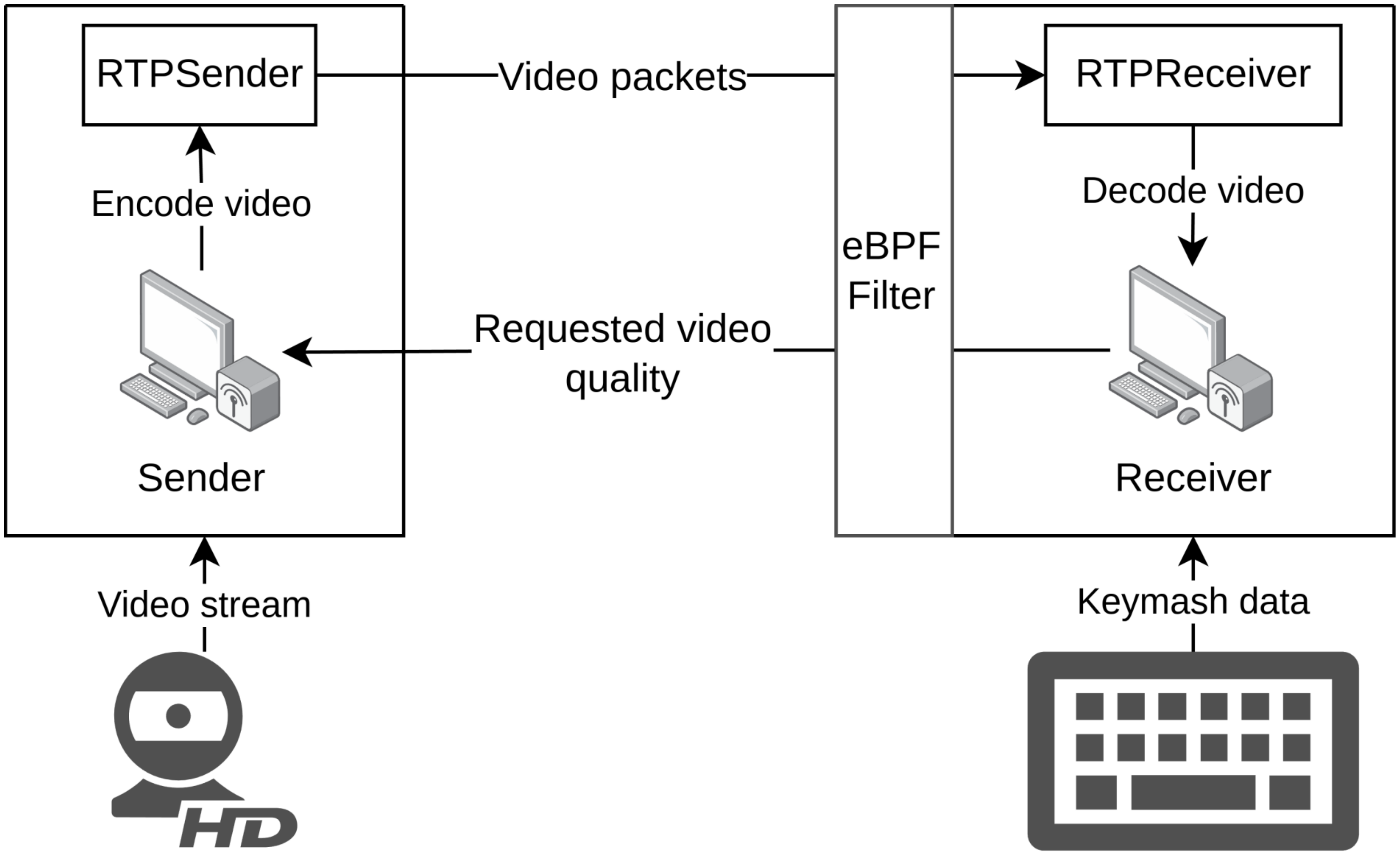

keymashd
Kartavya Vashintha
Ya know, how sometimes your computer's internet is slow? What if... you could motivate it. Make the internet itself flow a lil' quicker.
Since making the internet faster is a hard research problem, keymashd instead settles for slowing down the internet and then easing up on the impairment based on the keys you mash. Observe the effects of your encouragement through a bad video protocol made for your enjoyment. Mash a variety of keys for best effect.

Agenda

- **Problem:** Packet loss is a fact of life. How do we reduce it?
- **Solution:** *Artificially introducing more* so we can reduce it later.
  We use an eBPF filter.

- **Problem:** Existing video codecs compensate for packet loss very well.
- **Solution:** Custom network and video protocol which produces interesting *compression* and *packet loss* artifacts.

Architecture



eBPF Packet Loss Deep Dive

Recall:

We want to artificially introduce more packet loss.

Internet packets are routed by the kernel networking stack.

Can we drop packets at the kernel layer?

eBPF Packet Loss Deep Dive

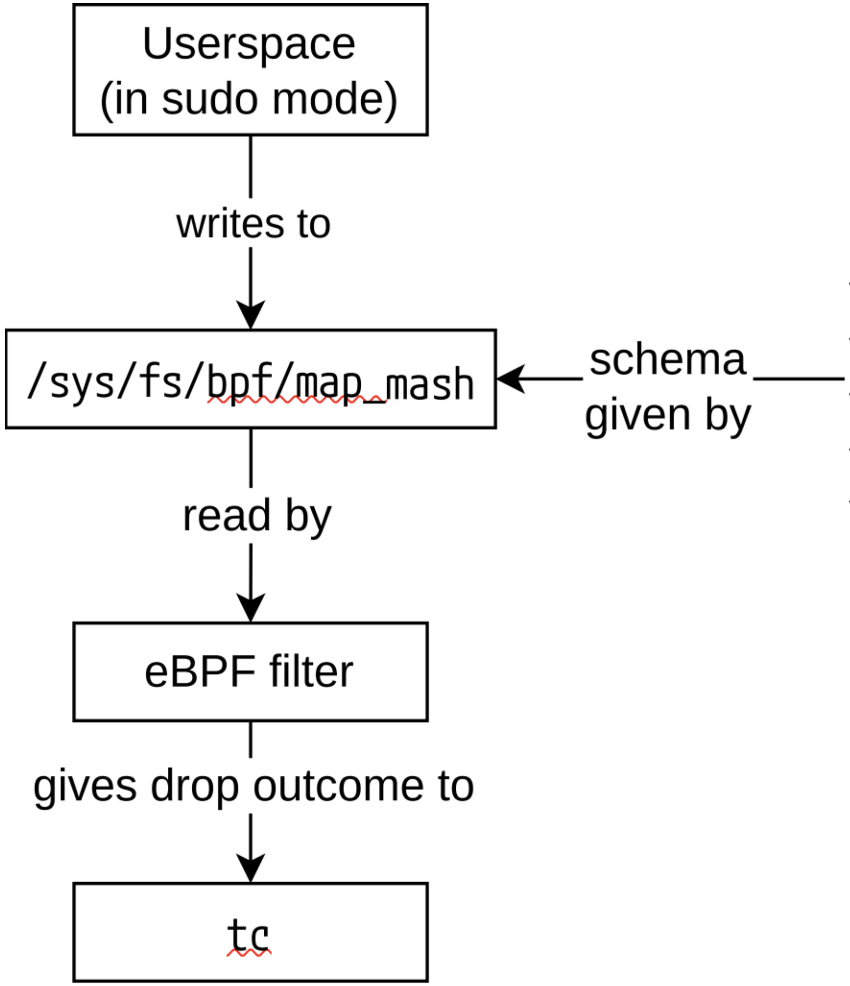
- Compiled to bytecode.
- Allows limited in-kernel user-supplied code execution.
- Can communicate with user programs.
- Useful for observability, security, and networking.

We feed this to the tc utility.
Runs for every packet received on the network interface.

```
__section("classifier")
int mash_bpf(struct __sk_buff *skb)
{
    uint32_t key = 0, *val = 0;

    val = map_lookup_elem(&map_mash, &key);
    if (val && get_prandom_u32() < *val) {
        return TC_ACT_SHOT; // Drop packet
    }
    return TC_ACT_OK; // Pass packet
}
```

eBPF Packet Loss Deep Dive




```
struct {
__uint(type, BPF_MAP_TYPE_ARRAY);
__uint(key_size, sizeof(uint32_t));
__uint(value_size, sizeof(uint32_t));
__uint(max_entries, 1);
__uint(pinning, LIBBPF_PIN_BY_NAME);
} map_mash __section(".maps")
```

RTP Protocol Deep Dive

We can now artificially increase packet loss.

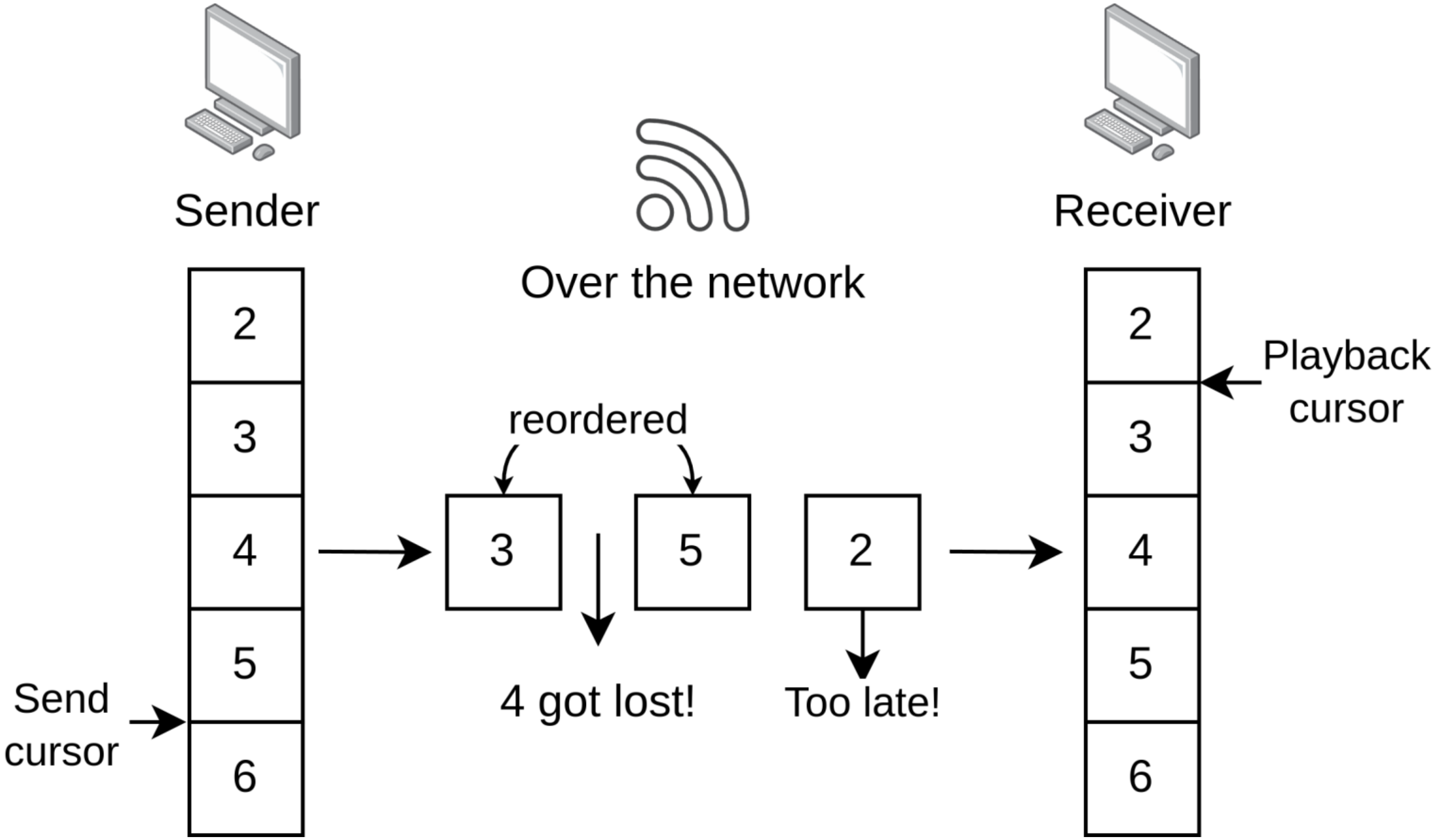
Recall:

Existing video codecs are *really good* at compensating for packet loss.

Solution:

We're going to build the network and video protocols ground-up to make them more interesting at high packet loss.

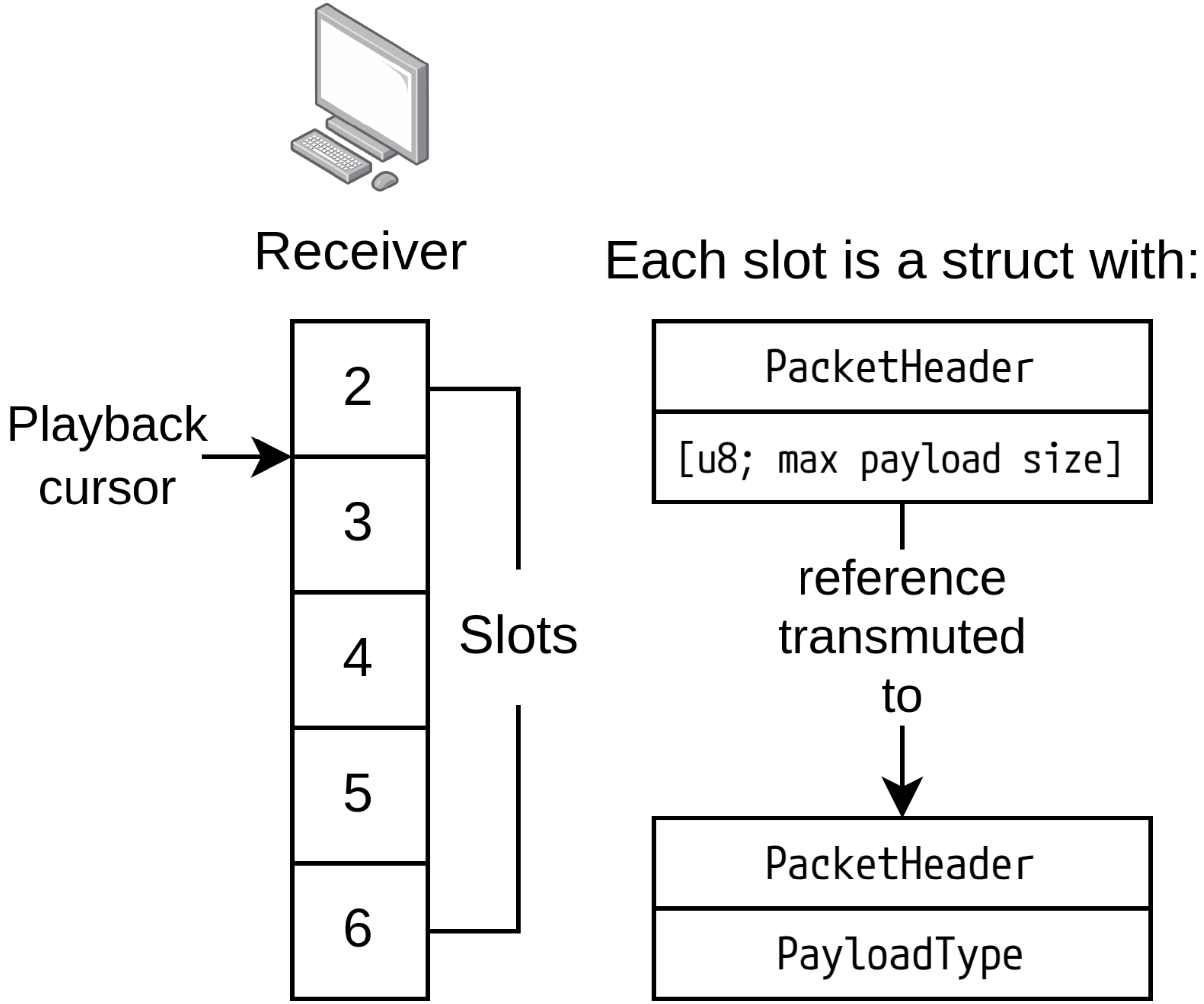
RTP Protocol Deep Dive



- **Problem:** send many packets, few arrive.
- **Challenge:** TCP tries for in-order guaranteed delivery, causing stutters.
- **Solution:** build protocol over UDP.

We implement a real-time streaming protocol library *from scratch*.

RTP Protocol Deep Dive



- Ring buffer holds packets.
- Receiver thread listens to UDP port for new packets.
- Acquires lock and writes received packets to *slots*.
- Playback cursor advances through *slots* sequentially.
- Uses Google's zerocopy library for safe transmutation.
- Highly generic over PayloadType. (more in next slide)

RTP Protocol Deep Dive

- Payloads are allowed to be unsized (**?Sized**).
  Still requires an upper-bound on the size of the payload (**SLOT_SIZE**).
- Can't automatically determine alignment of unsized objects (**AlignPayloadTo**).

```
/// An RTP receiver that receives packets over the network.
/// Arguments:
/// - `Payload`: The type of the data that is being sent.
/// - `AlignPayloadTo`: The type that has the correct alignment for the payload.
/// - `SLOT_SIZE`: The size of each slot in bytes (excluding packet header).
/// - `BUFFER_LENGTH`: The number of packets that can be stored in the buffer.
pub struct RtpReceiver<
    Payload: TryFromBytes + IntoBytes + KnownLayout + Immutable + ?Sized,
    AlignPayloadTo: TryFromBytes + IntoBytes + KnownLayout + Immutable,
    const SLOT_SIZE: usize,
    const BUFFER_LENGTH: usize,
> {
    rtp_circular_buffer:
        Arc<Mutex<RtpCircularBuffer<Payload, AlignPayloadTo, SLOT_SIZE, BUFFER_LENGTH>>>
}
```

RTP Protocol Deep Dive

```
/// An RTP receiver that receives packets over the network, specialized for a `[T]` payload.
/// Arguments:
/// - `SlicedPayload`: The type of the data that is being sent.
/// - `MAX_SLICE_LENGTH`: The maximum number of elements in the slice.
/// - `BUFFER_LENGTH`: The number of packets that can be stored in the buffer.
pub type RtpSlicePayloadReceiver<
    SlicedPayload: TryFromBytes + IntoBytes + KnownLayout + Immutable,
    const MAX_SLICE_LENGTH: usize,
    const BUFFER_LENGTH: usize,
> = RtpReceiver<
    [SlicedPayload],
    SlicedPayload,
    { size_of::<SlicedPayload>() * MAX_SLIC
    BUFFER_LENGTH,
>;
```

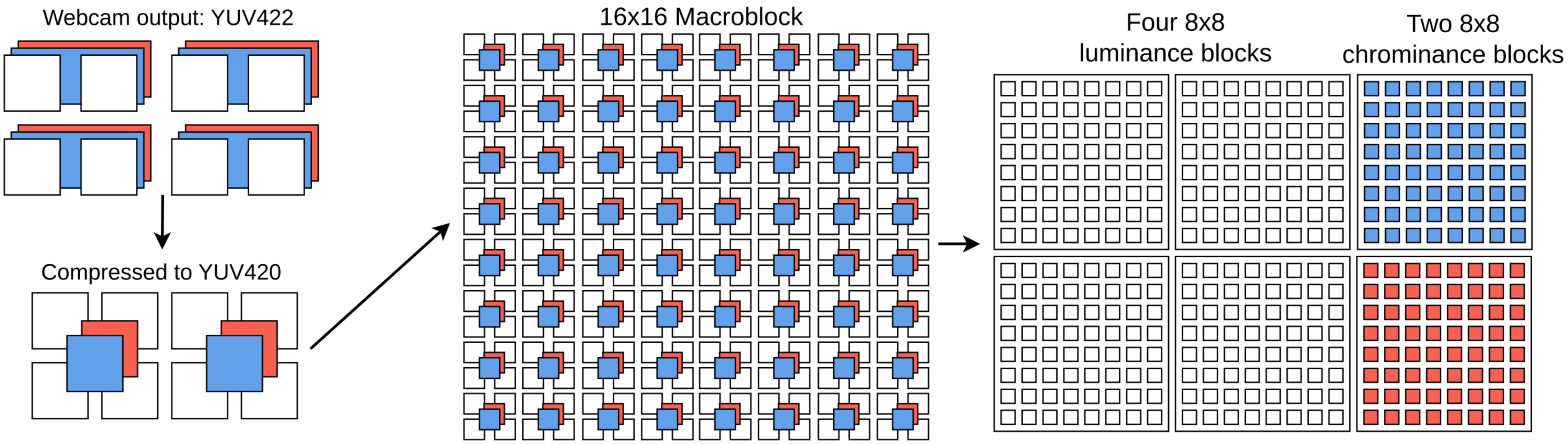
Read as: Circular buffer of 8192 packets of maximum size 1500 bytes each

Example specialization: RtpSlicePayloadReceiver::<u8, 1500, 8192>

Video Codec Deep Dive

We implement a JPEG-like scheme *from scratch*.

The human eye is more sensitive to light than color. We exploit this for compression.



Video Codec Deep Dive

Each block is DCT-encoded, quantized then reassembled into a quantized macroblock.

Quality factor

$$\frac{DCT_{i,j}}{f \cdot Q_{i,j}}$$

DCT Conversion

Original 8x8 Block

52	55	61	66	70	61	64	73
63	59	55	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

DCT of Block

-415	-30	-61	27	56	-20	-2	0
4	-22	-61	10	13	-7	-9	5
-47	7	77	-25	-29	10	5	-6
-49	12	34	-15	-10	6	2	2
12	-7	-13	-4	-2	2	-3	3
-8	3	2	-6	-2	1	4	2
-1	0	0	-2	-1	-3	4	-1
-0	0	-1	-4	-1	-0	1	2

Quantization Matrix

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

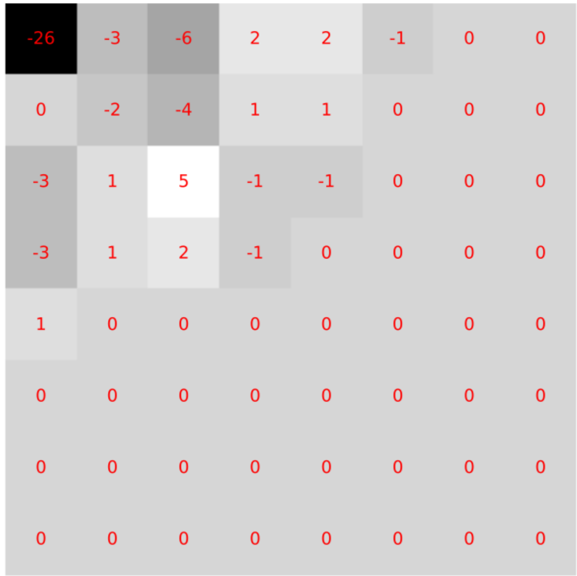
Quantized DCT Block

-26	-3	-6	2	2	-1	0	0
0	-2	-4	1	1	0	0	0
-3	1	5	-1	-1	0	0	0
-3	1	2	-1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

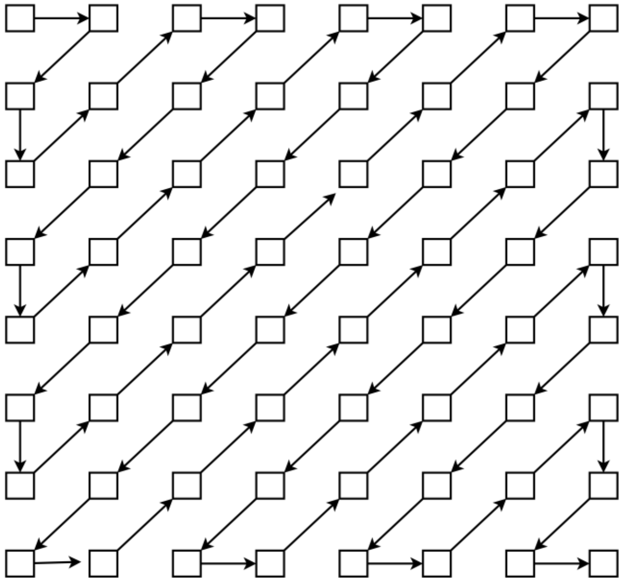
Video Codec Deep Dive

Quantized macroblocks are run-length-encoded, inserted into RTP packets and then sent over.

Quantized DCT Block



Zig-zag 2D to 1D transformation

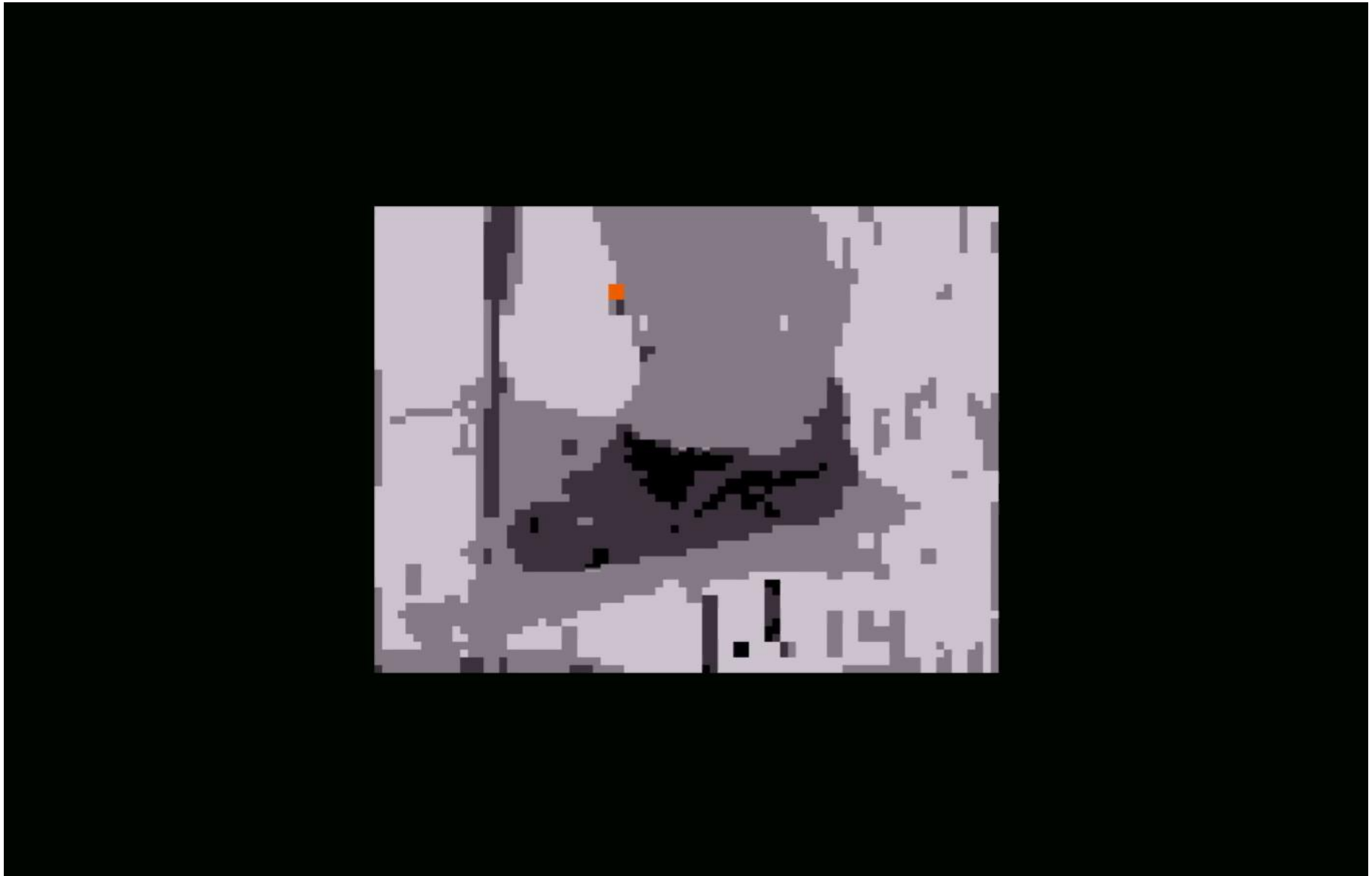


Run-length encoding

(-26, 1), (-3, 1), ..., (0, 31)

Packet Loss Demo

Packet Loss + Lossy Compression Demo



Retroactive: Increasing Performance

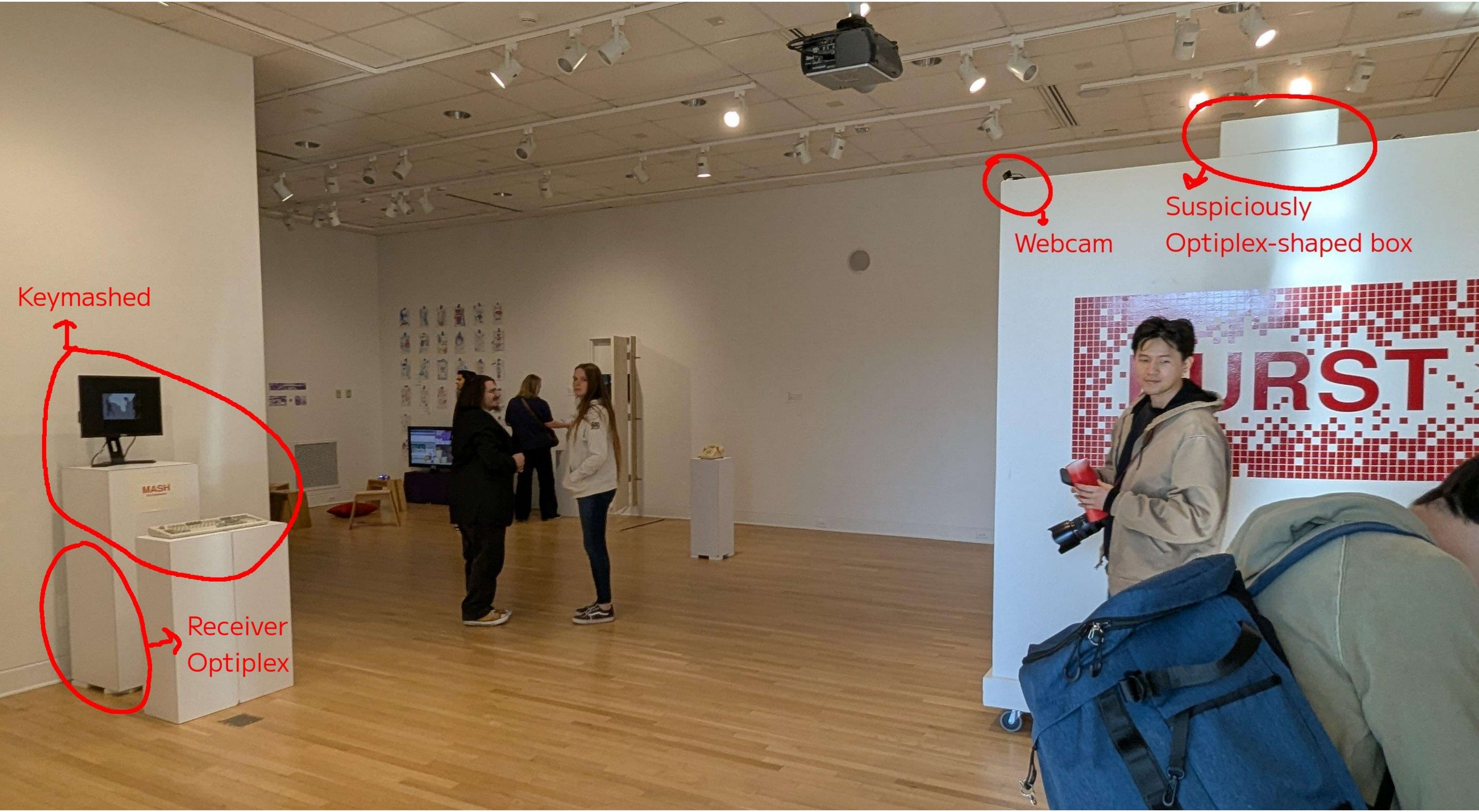
- We achieve ~6ms for sending/receiving frames.

- Encoding/decoding is by far the most significant cost. Assembly dump shows SIMD instructions, so further optimization might require algorithmic changes.

- Unblocking parallelism through reducing mutex usage.
(Currently, encoding threads acquire a mutex to insert quantized macroblock in the RTP packet).

Keymashed repository has more information!

github.com/kartva/keymashed


Keymashed
Receiver
Optiplex
Webcam
Suspiciously
Optiplex-shaped box
MASH
URST