Fair-share Rate limiting in BPF

Jonas Otten
Lorenz Bauer

Cloudflare

Outline

- Introduction
- Algorithm
- BPF Implementation
- Conclusion

Introduction - DNS & QUIC

DNS & QUIC services

- E.g. 1.1.1.1

→ UDP services

CLOUDFLARE

Introduction - Problem

DNS & QUIC services

- E.g. 1.1.1.1

What about Floods?

CLOUDFLARE

Introduction - Problem

DNS & QUIC services

- E.g. 1.1.1.1

→ Flood & legitimate traffic on same socket

→ Queues will overflow, traffic be dropped

CLOUDFLARE

Introduction - Problem

- TCP: SYN-cookies

→ no similar context for UDP

How can we protect UDP sockets?

Solution: Rate Limiting

Solution: Rate limiting

Idea

1. detect the traffic stream that’s flooding
2. rate limit

CLOUDFLARE

What is a “stream”?

- Need to group packets based on some criterion

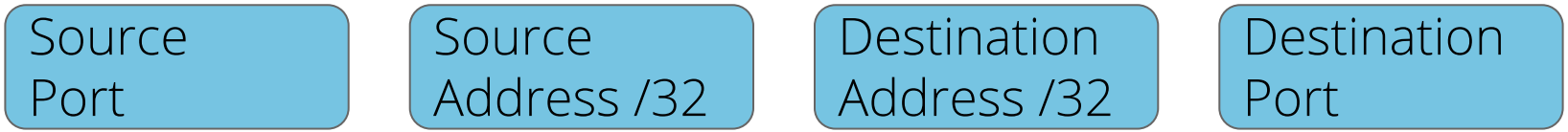
First Idea: match on part of 4-tuple

- Source Port
- Source Address
- Destination Port
- Destination Address

→ rate limit each of those

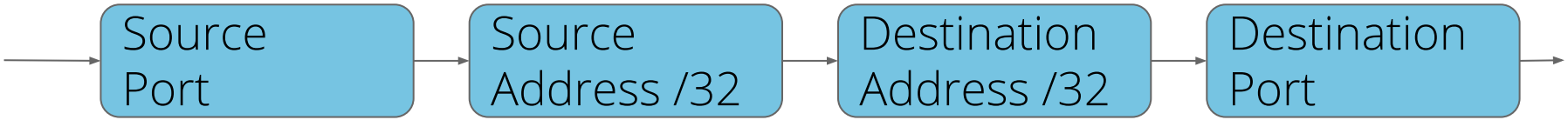
CLOUDFLARE

First Idea: match on part of 4-tuple



CLOUDFLARE

First Idea - Algorithm



CLOUDFLARE

First Idea: limitations

- Susceptible to DoS
  - E.g. Source Ports can be flooded easily

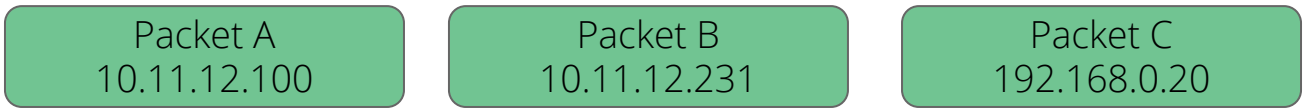
CLOUDFLARE

Idea: Evaluate Attributes simultaneously

- Based on *Hierarchical Heavy Hitters*
  - Simplifications due to rate limiting scenario
- Combine all attributes
  - → Lattice

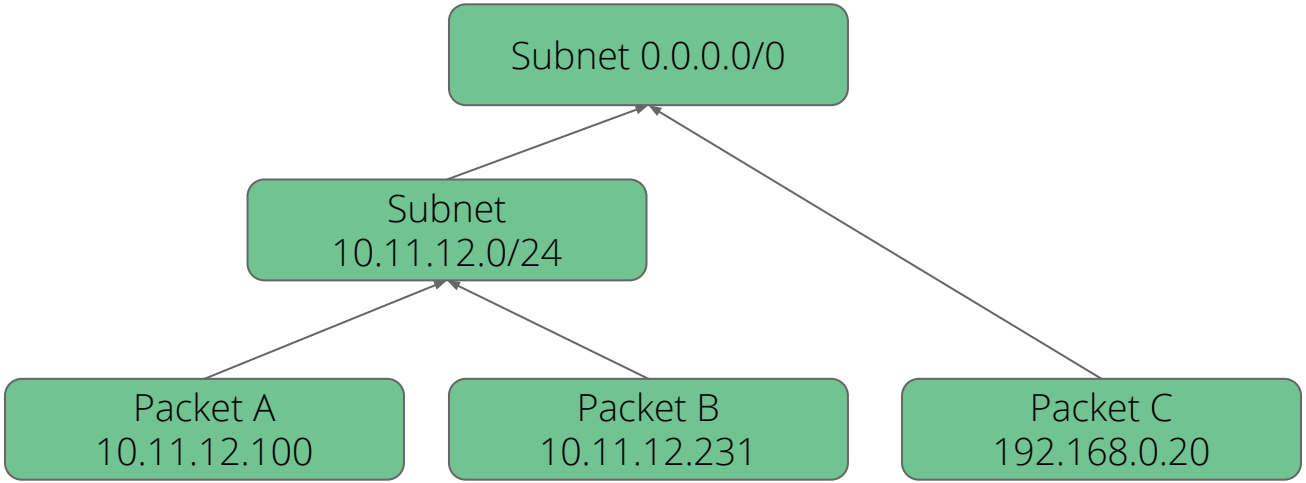
CLOUDFLARE

Grouping packets



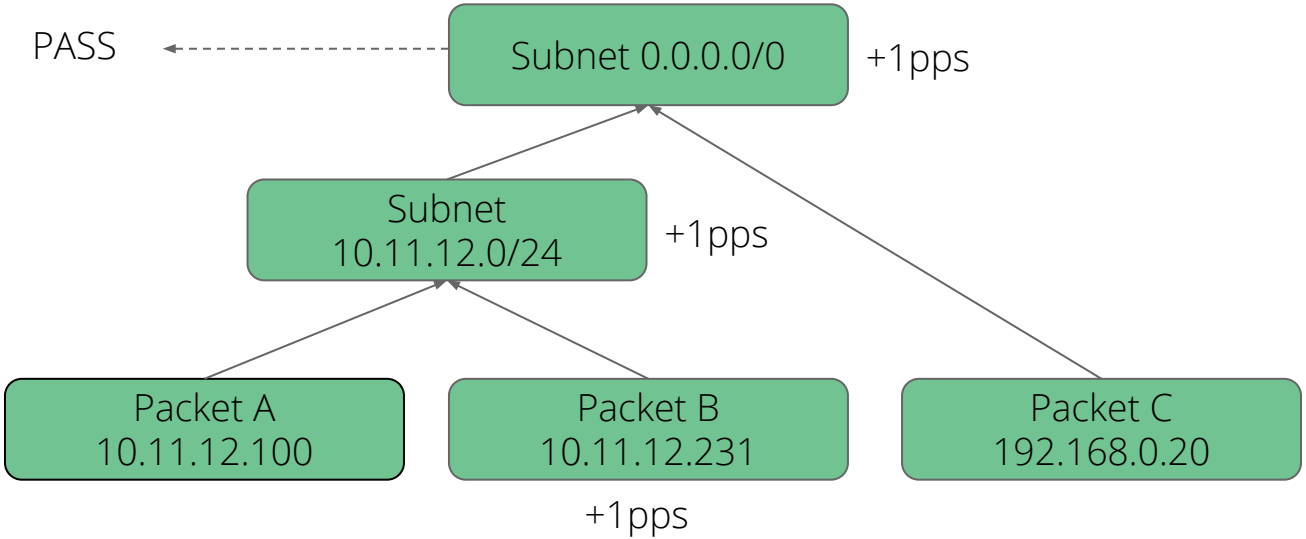
CLOUDFLARE

Grouping packets



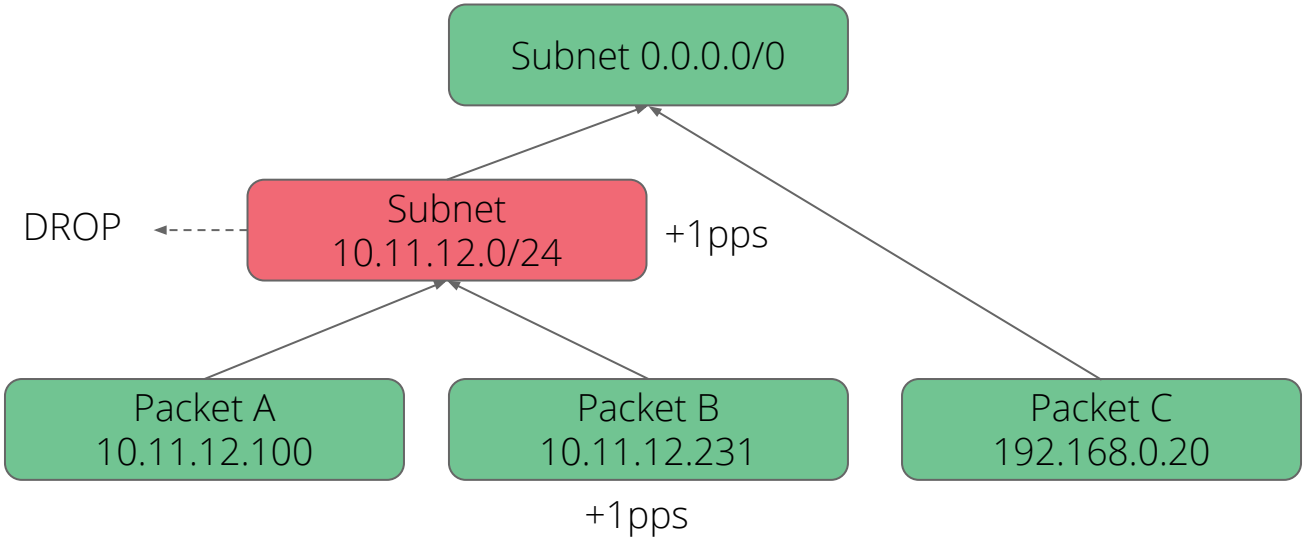
CLOUDFLARE

Incoming packet from 10.11.12.231

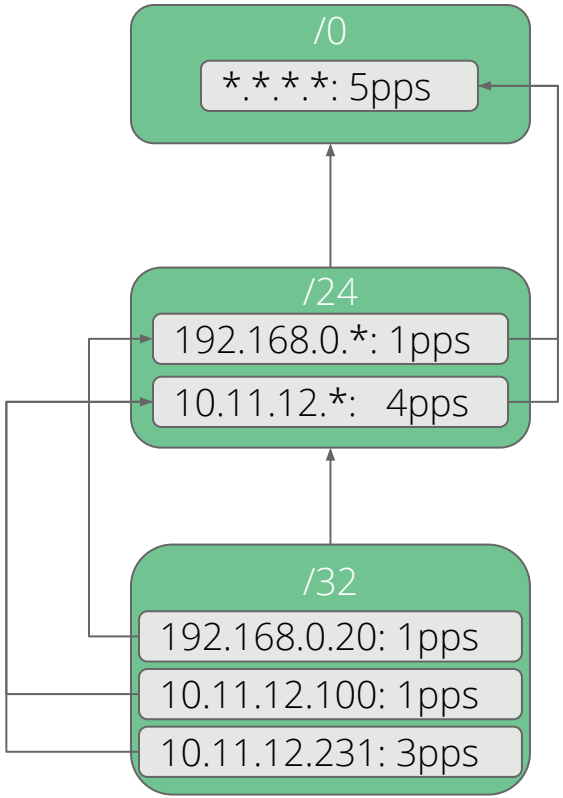


CLOUDFLARE

Incoming packet from 10.11.12.231

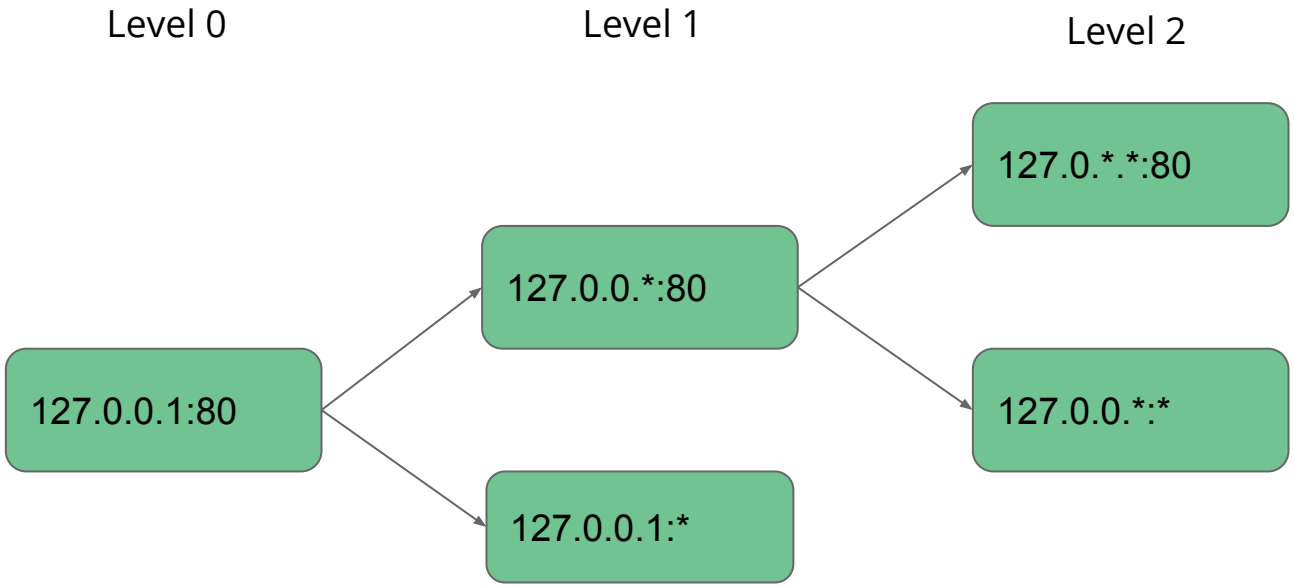


Hierarchy of a single attribute



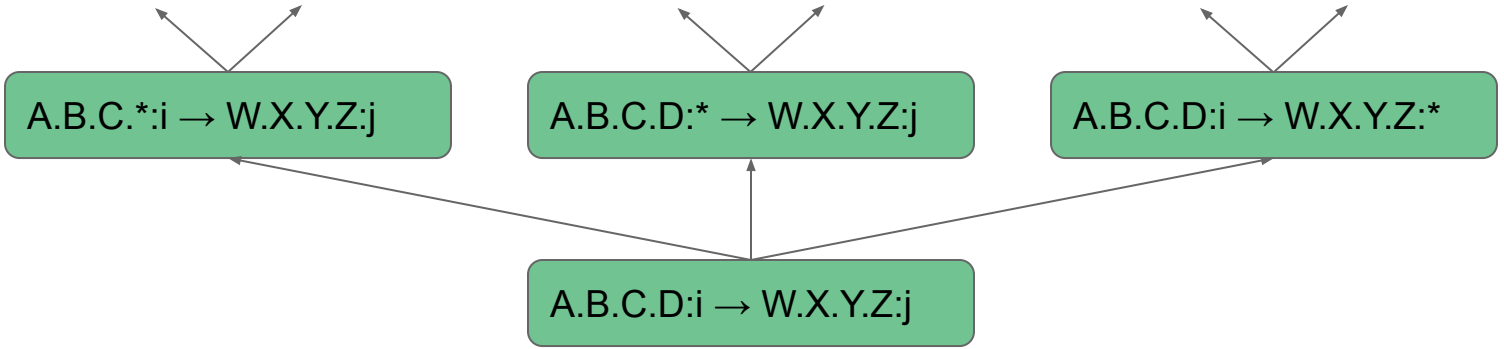
CLOUDFLARE

Multi-attribute Example



CLOUDFLARE

Multi-attribute: Extendable to more attributes



CLOUDFLARE

BPF Implementation

Unprivileged socketfilter - Why?

1. Minimise required privileges
2. Minimise blast radius
3. Only Service is aware of its appropriate rate limit

CLOUDFLARE

Ingredients

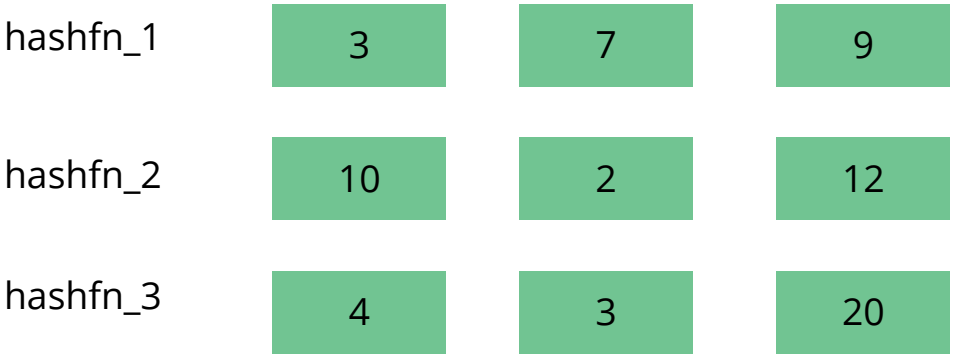
- Determine rates of any element
- We need a sliding window over packet rates

CountMin Sketch: estimate counts

- Probabilistic data structure
- Estimate counts for any element
- Constant Memory Requirements

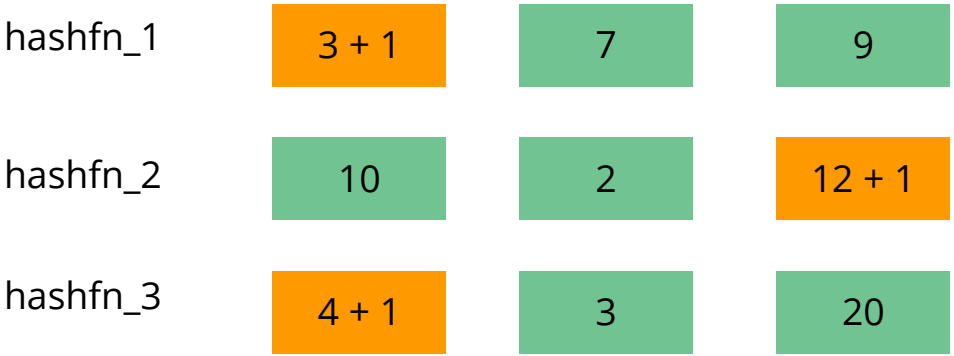
CLOUDFLARE

CountMin Sketch: estimate counts



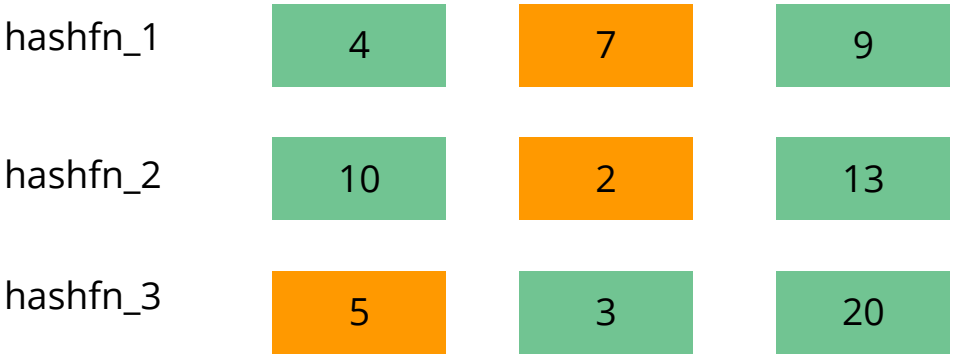
CLOUDFLARE

CountMin Sketch: Updates



CLOUDFLARE

CountMin Sketch: Queries



Estimate Count: 2

CLOUDFLARE

CountMin Sketch: estimate counts

- Probabilistic data structure
- Estimate counts for any element
- Constant Memory Requirements

→ works great with BPF array

→ fasthash to hash elements

CLOUDFLARE

CountMin Sketch: estimate counts

```
struct countmin {
    struct cm_value values[HASHFN_N][COLUMNS];
} __attribute__((packed));

struct bpf_map_def SEC("maps") countmin = {
    .type        = BPF_MAP_TYPE_ARRAY,
    .key_size    = sizeof(__u32),
    .value_size  = sizeof(struct countmin),
    .max_entries = NODES,
};
```

CLOUDFLARE

CountMin Sketch: estimate counts

```
// add element and determine count
static __u64 FORCE_INLINE add_to_cm(__u64 ts, struct countmin *cm, struct packet_element *element)
{
    fpoint min = -1;

    #pragma clang loop unroll(full)
    for (int i = 0; i < HASHFN_N; i++) {
        __u32 target_idx       = fasthash64(element, sizeof(struct packet_element), i) & (COLUMNS - 1);

        struct cm_value *value = &cm->values[i][target_idx];
        value->value           = estimate_avg_rate(value->value, ts - value->ts);
        value->ts = ts;

        if (value->value < min) {
            min = value->value;
        }
    }
    return min;
}
```

CLOUDFLARE

Determining Rates

Naive:

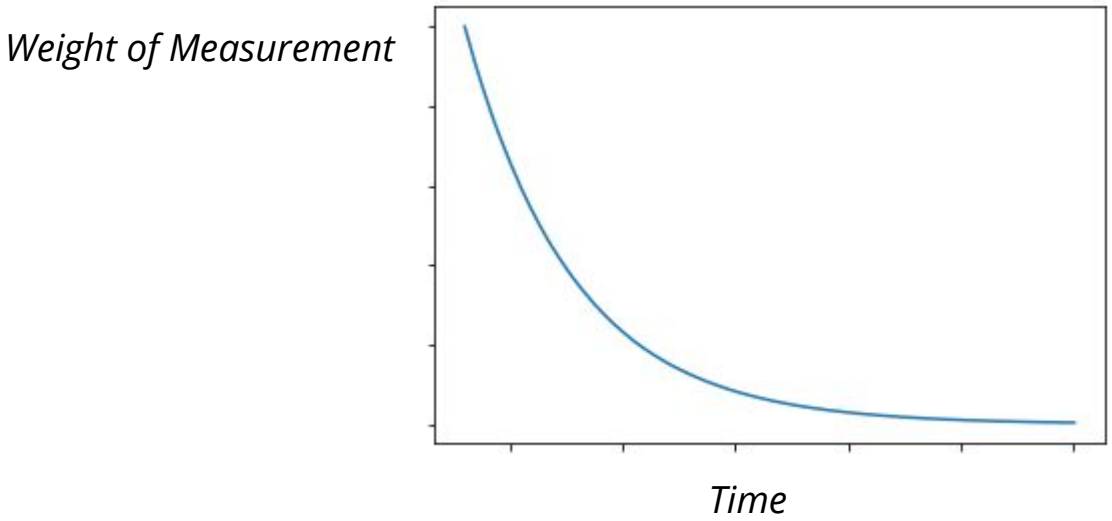
rate = 1/duration since last packet

→ Loose all context before

→ Instead: Exponentially Weighted Moving Average (EWMA)

CLOUDFLARE

EWMA: discounting rates over time



CLOUDFLARE

EWMA: discounting rates over time

Previous rate, duration since last packet

→ instant_rate = rate since last packet

```
estimate_rate_pps(instant_rate, old_rate):
    return (1-a) * instant_rate + a * old_rate
```

CLOUDFLARE

EWMA: discounting rates over time

→ 0 <= a <= 1

→ use fixed points for discounting

```
estimate_rate_pps(instant_rate, old_rate):
    return (1-a) * instant_rate + a * old_rate
```

CLOUDFLARE

EWMA: discounting rates over time

```
fpoint a = to_fixed_point(dur) / WINDOW;

fpoint new_rate = old_rate;
if (old_rate > to_fixed_point(rate_current)) {
    new_rate -= a * to_int(old_rate - to_fixed_point(rate_current));
} else {
    new_rate += a * to_int(to_fixed_point(rate_current) - old_rate);
}
return new_rate;
```

CLOUDFLARE

It works!

```
jonas@nebelhorn:~/git/rakelimit$ ls -lh src/rakelimit_bpfel.o
-rw-rw-r-- 1 jonas jonas 875K Aug 20 16:07 src/rakelimit_bpfel.o
jonas@nebelhorn:~/git/rakelimit$
```

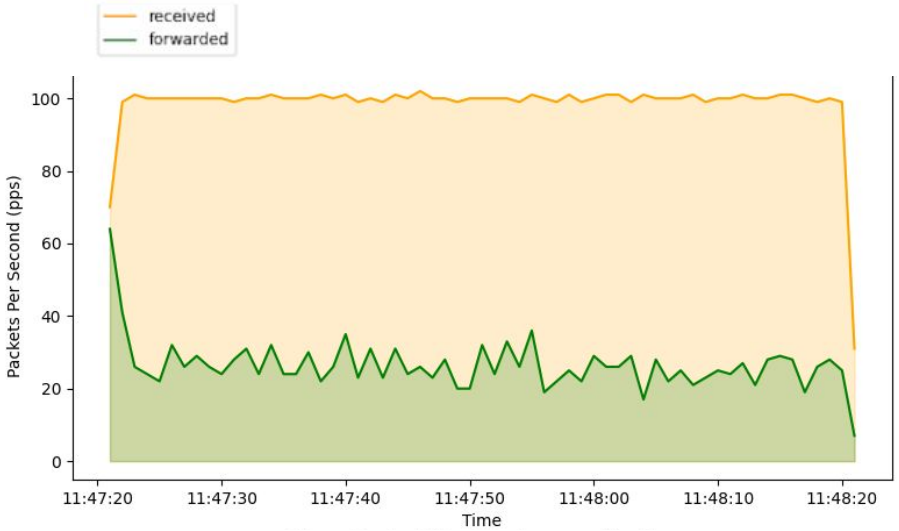
CLOUDFLARE

Testing

- Go Implementation as reference
  - Validate the algorithm
  - Validate Results of BPF Implementation
- Created flood scenarios
  - Single address/port flood
  - /24 {single,random} port flood
  - Reflection attack

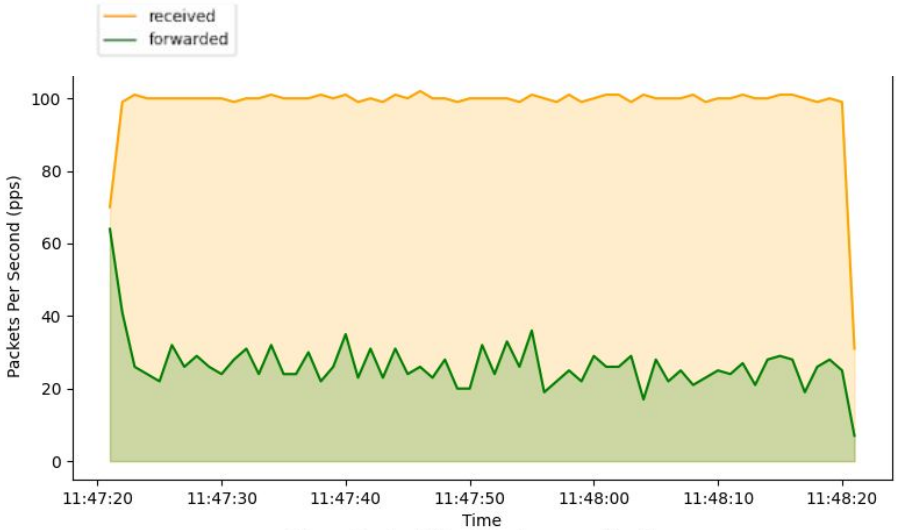
CLOUDFLARE

Scenarios - Single address & port

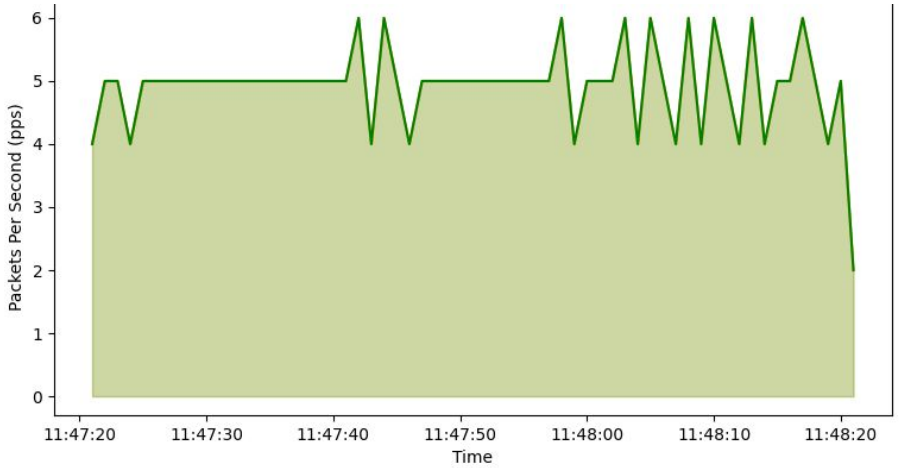


CLOUDFLARE

Scenarios - Single address & port



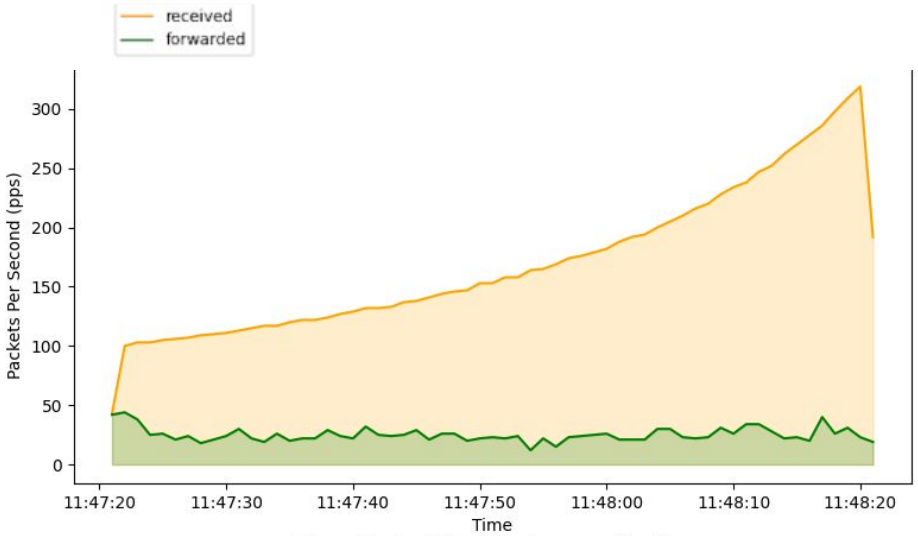
10.11.12.13:89



10.11.12.15:89

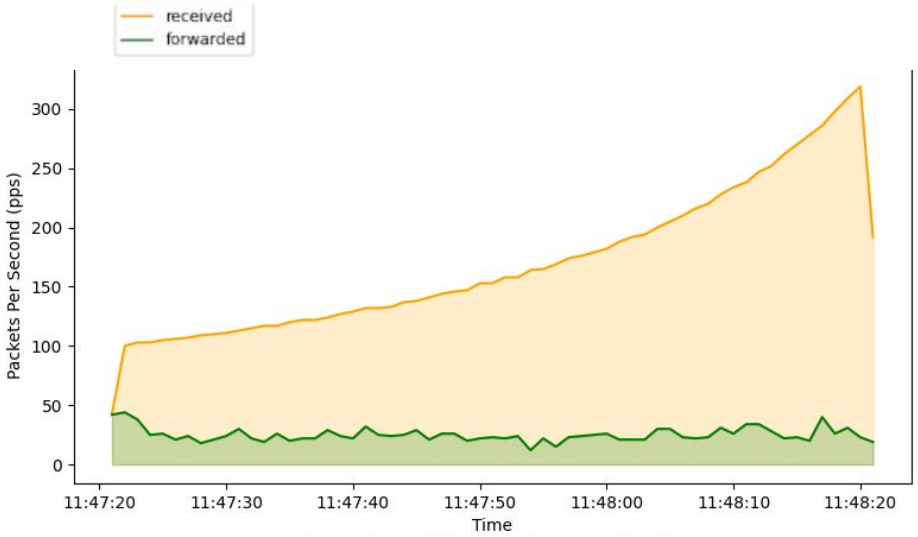
CLOUDFLARE

Scenarios - /24 address & single port

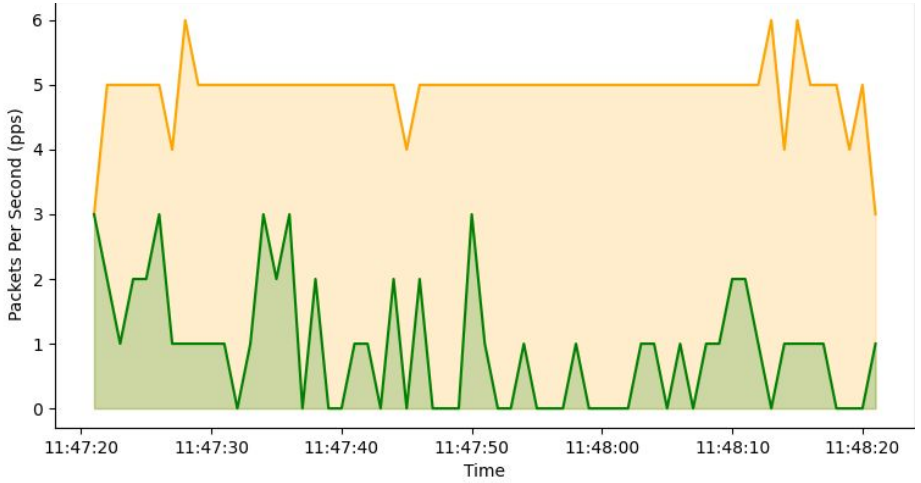


CLOUDFLARE

Scenarios - /24 address & single port



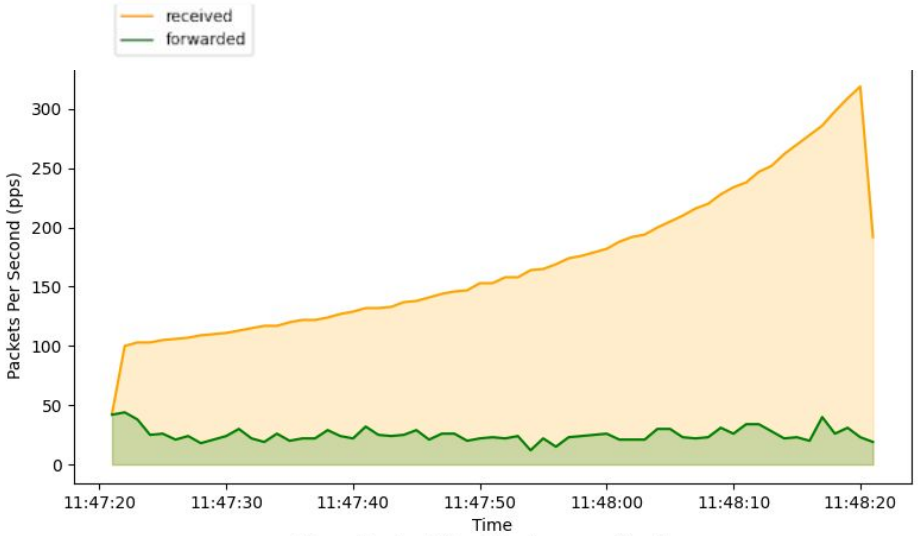
10.11.12.*:89



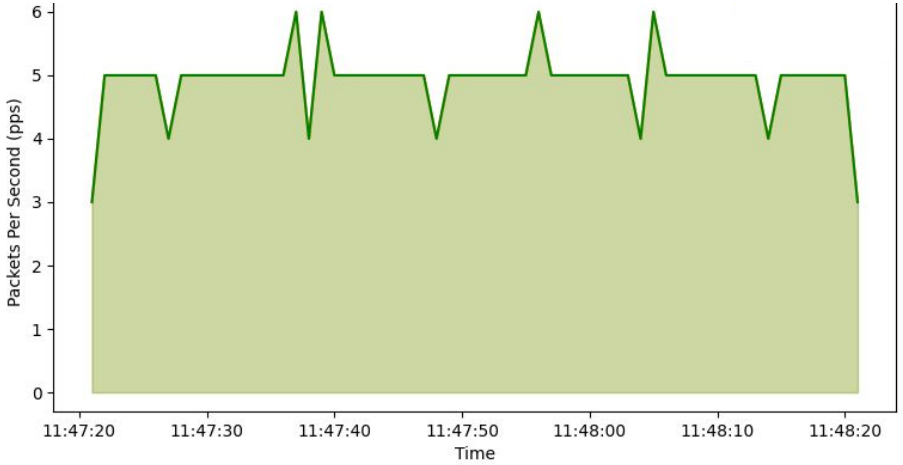
10.11.12.15:89

CLOUDFLARE

Scenarios - /24 address & single port



10.11.12.*:89



10.11.12.15:90

CLOUDFLARE

Limitations of unprivileged Socketfilters

Limitations unprivileged socketfilter

- Missing context fields

```
// does not work
*port = skb->remote_port;
// instead
*port = (load_byte(skb, 0) << 8) | load_byte(skb, 1);
```

CLOUDFLARE

Limitations unprivileged socketfilter

- No bpf to bpf calls

```
#define FORCE_INLINE inline __attribute__((__always_inline__))

static FORCE_INLINE int process_packet(__u64 ts, struct __sk_buff *skb)
{
```

CLOUDFLARE

Limitations unprivileged socketfilter

- No bpf to bpf calls

```
#define FORCE_INLINE inline __attribute__((__always_inline__))

static FORCE_INLINE int process_packet(__u64 ts, struct __sk_buff *skb)
{
```

  - Forced inlines: 875k
  - No inlines: 81k

CLOUDFLARE

Limitations unprivileged socketfilter

- No floating points (BPF wide limitation)

```
static __u64 FORCE_INLINE to_fixed_point(__u32 n)
{
    return ((__u64)n) << FRACTION_BITS;
}
```

CLOUDFLARE

Limitations unprivileged socketfilter

- No floating points (BPF wide limitation)
  - Instead: Fixed-points
  - Warning: had to spend lots of time to avoid precision problems

Conclusion

- Rate Limiter impacts minimum set of streams
- Rates can be estimated in BPF
  - EWMA
  - Fixed Points
- Limitations in unprivileged BPF
  - Can be worked around

CLOUDFLARE

Future

Continue optimisation within Cloudflare

Benchmarking

Publish Blog post

CLOUDFLARE

Thanks!

jonas@cloudflare.com
lmb@cloudflare.com