On Augmenting TCP/IP Stack via eBPF

Sepehr Abbasi Zadeh, Ali Munir, Mahmoud Mohamed Bahnasy, Shiva Ketabi, Yashar Ganjali
Outline:

- Congestion Control Challenges.
- Augmenting TCP/IP via eBPF.
- Augmenter Design.
- Augmenter Implementation.
- Augmenter Evaluation.
- Discussion and Future Work
- Conclusion.
Congestion Control Challenges:

- Next generation DCN/cloud requires **high-bandwidth** and **predictable ultra-low E2E latency**.
- Link capacity is growing rapidly (from 1/10Gb/s to 100/400Gb/s and expected 800Gb/s)
- Flows become smaller: 60-80% flows can finish within one RTT at 100Gb/s, expected that 89% will finish with 400Gbps link capacity.

![Graph showing percentage of flows finishing within one RTT at different link capacities.](image)

**Majority of data flows finish within one RTT and don’t get a chance to react to congestion signals.**
Congestion Control Challenges:

1. Adaptiveness should start from the first RTT:
   - Majority of flows can finish in one (or few) RTTs
     - Therefore, these short flows cannot sense any congestion
   - Existing caching mechanisms (e.g., caching ssthresh per destination) are passive
Congestion Control Challenges:

2. End-hosts treat each flow separately
   - CC signals are per flow
     - Packet loss, ECN, … etc.
   - Other solutions aggregate RTT and CC signals:
     - Aequitas: For admission control.

Sharing flows’ information within the same data path could lead to better CC decision.
Congestion Control Challenges:

3. Linux TCP/IP stack rigidity:
   ❑ Packet processing is predefined inside the host’s kernel.
     ➢ Can be configured with coarse grain configuration parameters; e.g., initial CWND for all flows.
   ❑ Changing Kernel function is inflexible:

<table>
<thead>
<tr>
<th>Traditional Method of changing the kernel</th>
<th>Native Support</th>
<th>Kernel Module</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Change kernel source code and convince the Linux kernel community that the change is required.</td>
<td>Write a kernel module</td>
</tr>
<tr>
<td></td>
<td>Wait several years for the new kernel version to become a commodity.</td>
<td>1. Fix it up regularly, as every kernel release may break it</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Risk corrupting your Linux kernel due to lack of security boundaries</td>
</tr>
</tbody>
</table>

Adapted from [https://ebpf.io/what-is-ebpf/](https://ebpf.io/what-is-ebpf/)

eBPF brings back programmability.
Question: Can we enrich flow decisions with prior experience from existing flows using eBPF programmability?

Augmenter!
Augmenting TCP/IP via eBPF:

- eBPF allows reprogramming the behavior of the Linux kernel without changing the kernel source code or writing kernel modules.

Prior flow experience is aggregated. + eBPF Stores/shares flow states (per data path view) + eBPF hooks invoke different TCP/IP reaction, while accessing shared flow information.
Augmenter Design:

- **Main Idea:**
  
  Augmenting TCP decision making process with prior flow experiences collected and shared among active flows using eBPF modules.

- **Components:**
  1. Kernel-level *state aggregation*.
  2. Flow state analyzer and TCP decision altering.
Augmenter Design:

   - Aggregating flow information per path:
     - E.g., when a set of flows are limited at a certain congestion point, all new flows that are going to use the same data path can use congestion signals (e.g., IW, ECN, packet loss) from other flows.
   - Sharing information between kernel layers using eBPF maps:
     - E.g., Congestion signal (ECN marking for example) experience by flows using this data path.
     - Last stable cwnd for flows sharing the same data path.
Augmenter Design:

2. Flow state analyzer and TCP decision altering.

- Data flow can read and analyze prior flow experience available by eBPF modules.
- TCP decision process is altered at a faster and more flexible control loop:
  - Floating-point calculation is allowed
  - Opportunity for sophisticated ML-based agents
Augmenter Implementation:

- Our code is being injected in the datapath of the Linux TCP/IP stack
- This induces very minimal overhead for packet processing, as if the kernel had to call another function through its path

```c
SEC("sockops")
int trp_sockops(struct bpf_sock_ops *skops)
{
    int *res, lw = 10;
    int op = (int) skops->op;
    u32 dst_ip;
    struct bpf_sock *sk = skops->sk;
    struct bpf_tcp_sock *tcp_sk;
    dst_ip = bpf_ntohl(skops->remote_ip4);
    ...
}
```
Augmenter Evaluation:

Experiment I (Testbed):

- Objective: Validate that aggregating average cwnd improve per-flow initial cwnd decision.
- One client generate twelve requests from two servers (6 per server) every 5ms.
- Each request generate a responded of 1MB in size.
- Evaluation metric: Flow Completion Time (FCT)
- 6000 flows were generated in total.

Augmenter significantly improves application performance in terms of FCT (50% reduction).

eBPF and Kernel Extensions Workshop, SIGCOMM 2023
Augmenter Evaluation:

Experiment II (NS-3 Simulation):
- Scenario: a simple incast scenario while varying the incast size.
- Evaluation metric: Flow Completion Time (FCT)
- DCTCP is used as the underlying congestion control.
- Each sender sends a flow of 500KB towards one receiver.

Augmenter automatically adapts to the scenario; it performs similar to line-rate at low loads and similar to IW-10 at high loads.
Discussion and Future Work:

Future Work:

- CC algorithms can benefit from reusing the congestions signals experience by other flows in the same data path rather.
- E.g., Augmenter can be used to aggregate other ECN signal for ECN-based CC (e.g., DCTCP, DCQCN and BBRv2).
- Augmenter aims to provide useful suggestions to different transport protocols (e.g., TCP, RDMA, etc.).
- The programmable eBPF allows us to change priority at a packet-level on the fly not per flow.

Challenges:

- Managing multiple paths require sophisticated state keeping to ensure information aggregation accuracy.
- Sharing packet-level information between flows might raises some security concerns.
Conclusion:

- We proposed a framework, Augmenter, that increases the network visibility of the TCP/IP stack.

- Leveraging eBPF, Augmenter gathers the state of ongoing flows and uses this information to manage other flows.

- We present one specific use case of setting the initial congestion window (IW) dynamically.

- Our initial tests, show that Augmenter can improve the application performance by up to 1.4x compared to the fixed initial window-based solutions.
Thank you.