Report: communcation jitter analysis, periodic peaks

Davide Rovelli, Michele Dalle Rive

04/2024

I Objective

We analyse the latency disitribution in the communication between 2 bare-metal nodes in a datacenter. The objective of our experiments is to make packet processing latency of Linux-based hosts with modern NICs as *deterministic* as possible. This consists in reducing communication jitter to a minimum, where $jitter = \max(latency) - \min(latency)$.

Contrarily to usual low tail-latency experiments/systems which aim to reduce the common-case latency up to some percentile (e.g.$99.5\%, 99.9\%$) we want to minimise the 100% latency, even at the cost of (non-substantially) increasing the common-case.

II Methodology

II.1 Benchmark

We run ping-pong tests across 2 hosts **A** and **B** by sending packets at a fixed sending rate. Each packet carries a packet ID and 4 timestamps:

- ts_1: timestamp just before sending the PING message in host **A**
- ts_2: timestamp as soon as the process in host **B** receives the PING packet.
- ts_3: timestamp just before sending the PONG message in host **B**
- ts_4: timestamp as soon as the process in host **A** receives the PONG packet.

We then calculate the difference between timestamps of successive packets to get the variations in latency. For example, $ts_1(33) - ts_1(32)$ where 33 and 32 are the packet IDs, represents the time interval between the PING send timestamp of packet 32 and the PING send timestamp of packet 33. If there's no jitter, it should be equal to the sending rate. The difference between $ts_1(33) - ts_1(32)$ and the sending rate represents the relative jitter for that pair.

II.2 Hardware setup

Cluster A 2x CloudLab xl170 nodes

- CPU: Intel Xeon E5-2640 v4 at 2.40GHz, 10 cores, 64GB RAM
- OS/kernel: Ubuntu 22.04.4 LTS / 6.6.19-060619-generic x86_64
- NIC/driver: Mellanox Connect-X 4 / mlx5

Nodes are connected over ethernet, via a TOR switch with zero or minimal network load in order to only observe the endhost processing overhead.

II.3 Software setup

We use different *kernel bypass* methods in order to minimize the well-known overhead introduced by the classical network stack in Linux machines. We test the following:

- RoCE, specifically double-sided SEND/RECV RDMA with the Unreliable Datagram (UD) transport type
- eBPF XDP in two different modalities:
  - XDP-offload: packet is handled by an eBPF XDP hook and executed as early as possible in the network stack
  - AF_XDP: XDP socket type which handles TX and RX buffer management to the application
- Standard network stack over UDP

Packet size: *1024 bytes*

III Analysis of periodic outliers

Here we show some of the results which include interestingly periodic latency outliers, which we refer to as "peaks". We observe two types of periodic peaks of which we have not yet identified the root cause. We refer to them as ts_1**-offset** and ts_2**-arrow**.

Figure 1 shows a typical benchmark containing a thick baseline around the sending rate ($996\mu s$), which means that most packets are subject to a minimal jitter apart from the the outlier types that we analyze below.

ts_1**-offset** The top-left plot in Figure 1, representing the difference between 2 successive send events of the PING packet on host A, shows ts_1**-offset**: a small number of outliers delayed from baseline by a similar offset in the y-axis ($\approx 150\mu s$). Successive outliers evenly spaced, meaning there is a fixed number of packet (i.e. time interval) between them.

ts_2**-arrow** The top-right plot in Figure 1, representing the difference between 2 successive receive events of the PING packet on host B, shows ts_2**-arrow**: a series of outliers with decreasing amplitude. Every outlier on top of the baseline (delayed packet) is immediately followed by a symmetric point below the baseline which shows that the successive packet was not delayed, hence the difference between its timestamp and the one of the previous delayed packet is less than then sending rate. Successive outliers forming the arrow are evenly spaced, meaning there is a fixed number of packet (i.e. time interval) between them.

Bottom left and bottom right plot show no periodic outliers, meaning that the PONG send and receive events do not seem to be subject to periodic peaks in our experiments. ts_1**-offset** and ts_2**-arrow** have the following characteristics:

- Independent from the software technology used, they manifiest with all bypass techniques listed in subsection II.3 as well as the normal network stack.

- Throughput dependent: changing the sending rate changes some characteristics as we cover in the next section

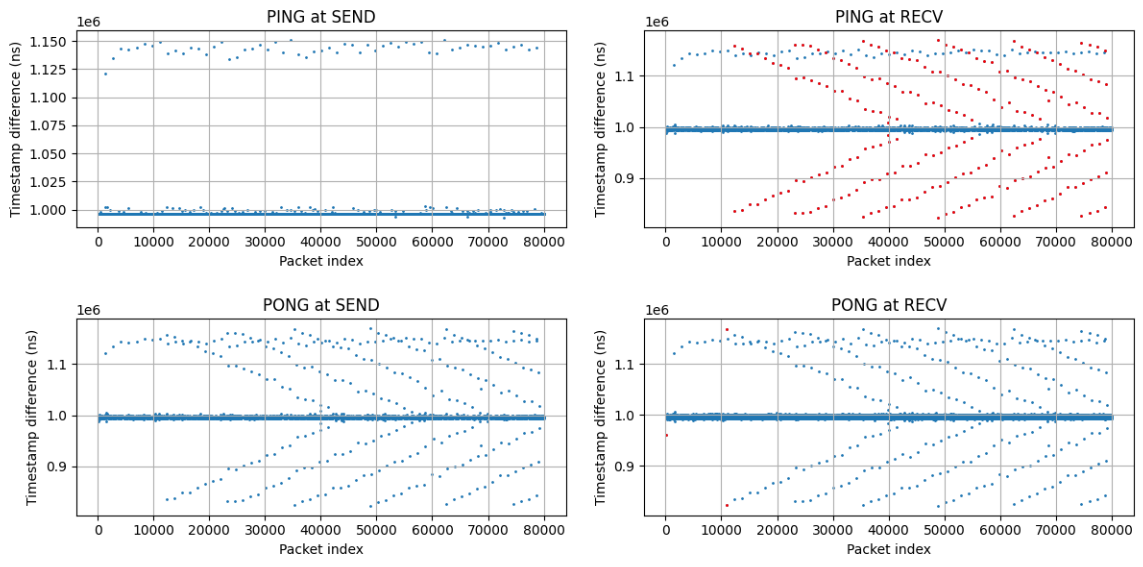

Figure 1: Pingpong benchmark plot - sending interval $996 \mu s$. Each dot corresponds to a packet ID (x-axis) and the difference between its timestamp and the timestamp of the packet with the previous ID (y-axis). The sending interval is fixed and packets are sent in order. See subsection II.1 for a more detailed explanation of the benchmark

- Partially reproducible: 2 runs with the same settings often leads to the same result but not always, especially ts_2**-arrow** seems to be the most variable.
-