A Vectorised Packing Algorithm for Efficient Generation of Custom Traffic Matrices

Christoper W. F. Parsonson,[1,*] Joshua L. Benjamin,[1] and Georgios Zervas[1]

[1] *Optical Networks Group, Electronic & Electrical Engineering, UCL*

[*] *zciccwf@ucl.ac.uk*

Abstract: We propose a new algorithm for generating custom network traffic matrices which achieves 13×, 38×, and 70× faster generation times than prior work on networks with 64, 256, and 1024 nodes respectively.

1. Introduction

Data centres (DCs) have become critical tools for modern computational tasks. To meet the ever-increasing demands of DCs, recent years have seen a growth in the research and development of next-generation DC optical systems [1]. However, most researchers rely on simulations, which require the generation of synthetic traffic. In doing so, they often make overly simplistic assumptions about the characteristics of their generated traffic and develop systems which, in practice, perform poorly under real-world conditions [2]. Furthermore, many works omit open-accessing their synthetic traffic or even the methodology used to generate it, bringing problems with reproducibility, benchmarking, and cross-validation. The lack of a reproducible and high-fidelity synthetic traffic generation tool has been a long-standing problem in the DC research community.

Prior works [3,4] have released traffic generators, but these were either intended to be unrealistic, were for specific network topologies, required the cumbersome use of inflexible configuration files, or lacked a reproducibility guarantee. To address this, recent work presented TrafPy; an open source tool for generating reproducible DC traffic with custom distributions and characteristics [2]. However, the authors only demonstrated traffic generation for 64 network nodes; far smaller than the $O(1000)$ node DCs which are becoming increasingly common place.

In this work, we first show that the original flow source-destination assignment algorithm ('packing', see Section 2) used in the TrafPy generator is a major bottleneck because its time complexity scales poorly with the number of DC nodes $|N|$ for which $|F|$ flows are being generated. This prevents the generation of traffic for large networks. Next, we propose a novel vectorised packing algorithm which fits in with the rest of the authors' traffic generation framework. Finally, we demonstrate our vectorised packer achieving 13×, 38×, and 70× faster generation times than [2] on networks with 64, 256, and 1024 nodes respectively with up to ≈ 5M traffic flows, with the speed-up factor increasing with the network size. We expect this work to unlock a new realm of DC research at scale and to further facilitate the development of next-generation systems and common platforms for benchmarking networks. We note that while here we focus on generating traffic for optical DCs, the same traffic generation scheme and vectorised packing algorithm could be re-purposed and applied to any network system.

2. Custom Traffic Matrix Generation

Problem statement. DC traffic is made up of *flows*. A flow f is fully described by its *size* f^s (how much information to send), *arrival time* f^a (when the flow requests to be transported through the DC, thus giving rise to the *inter-arrival time* in a dynamic multi-flow setting), and *source-destination pair* f^p (which machines in the DC the flow is requesting to be sent between). In the framework of [2], traffic generation is split into two stages. In the first stage ('shaping and sampling'), custom flow size and inter-arrival time distributions are generated and sampled to attain a set of sizes $\mathbf{b}^s$ and arrival times $\mathbf{b}^a$ for $f \in F$ flows which match the target distributions within some Jensen-Shannon distance (JSD) threshold[1]. In the second stage ('packing'), given $\mathbf{b}^s$ and $\mathbf{b}^a$, the task is to assign each flow $f \in F$ to a source-destination pair such that some target node distribution (a.k.a. traffic matrix heat map) $\mathbb{P}^N$ with nodes $n \in N$ and corresponding source-destination node pairs $p \in P$ is realised as closely as possible without exceeding the load capacity limitations of any node. The authors formulate this task by extracting the fraction of the overall load requested by each pair $p \in P$ into an array, multiplying each element by the overall DC's target load rate to get the per-pair target load rate, and then again multiplying each element by the simulation duration (the time between the first and last flows' arrivals) to get the total amount of information to load onto each pair,

[1] The JSD $\in [0,1]$ is a measure of how similar two distributions are to one another (lower is more similar), and the JSD 'threshold' as defined by [2] is a constraint on how similar the generated traffic characteristics must be to the target distributions.

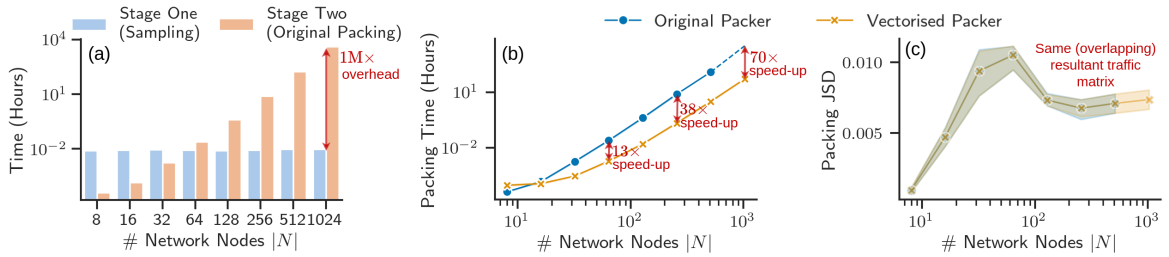

Fig. 1: i) (a) The time for stages one (shaping and sampling) and two (packing) when generating flows with the original packing algorithm. ii) The packing (b) time and (c) Jensen-Shannon distance between the target and the generated node distributions for the original and vectorised packing algorithms when generating traffic for networks with different numbers of nodes. (a) shows that the original packing algorithm is the major traffic generation bottleneck of [2]. (b) shows that as the number of network nodes is increased, the vectorised packer's speed-up factor over the original algorithm increases. (c) shows that both algorithms achieve the exact same resultant node distribution. Note that the original algorithm's time results for $|N| = 1024$ are extrapolations since it would have taken ≈ 200 days to run the packer.

$\mathbf{b}^{p,I}_{target}$, needed in order to achieve the desired target node distribution $\mathbb{P}^N$. The packing task is therefore reduced to finding the source-destination node pair assignments for each flow $f \in F$ such that the difference between the actual and the target per-pair total information loads, $\mathbf{b}^{p,I}_{target} - \mathbf{b}^{p,I}_{actual}$, is 0 *or*, where this is not possible given any incompatibility between the target node distribution $\mathbb{P}^N$ and the overall DC load rate, to match $\mathbb{P}^N$ as closely as possible (see [2] for further details).

As shown in Figure 1a, as $|N|$ is increased, stage two (packing) becomes a major bottleneck, taking $\approx$1 000 000 times longer than stage one for $|N| =$1024. We therefore focus on optimising stage two.

Original packing algorithm. The original packing algorithm of [2] works by sequentially iterating through the set of flows and, for each flow, conducting two passes through the candidate source-destination pairs. In the first pass, the packer attempts to match the target node distribution by looping through all pairs, sorted in descending order of the total size of flow information previously assigned, to find a pair which has not yet met its target information load given the target node distribution and total flow arrival duration provided. Failing to find such a node pair, the packer moves to the second pass, whereby it again loops through each sorted pair but now in search of a source-destination combination which, if allocated the flow in question, would not exceed either the source's or the destination's maximum load capacity given any prior flow allocations.

Vectorised packing algorithm. We negate the need for separate first and second passes and for nested pair `for` loops by using vector array operations. We begin by initialising the per-pair remaining capacity vector as the maximum port capacity (half the per-node capacity, since it is split between the source and destination ports) $\mathbf{b}^{p,c}$. We then sequentially iterate through $f \in F$ and, for each flow f, we generate a boolean vector pairs mask $\mathbf{b}^{p,m}$ which masks out any pair indices $i \in [0, ..., |P|]$ which would exceed their load capacity were they to be allocated the flow in question:

Algorithm 1 Vectorised packing algorithm pseudocode.

Input: F, P, $\mathbf{b}^{p,I}_{target}$
Output: $\mathbf{b}^{p,I}_{actual}$
Initialise: $\mathbf{b}^{p,I}_{actual} = 0(|P|)$, $\mathbf{b}^{p,c} = \frac{\text{node capacity}}{2}(|P|)$
for f in F **do**
  $\mathbf{b}^{p,m} =$ `where`$(\mathbf{b}^{p,c} - f^s < 0, 0, 1)$ `// Generate boolean mask`
  $\mathbf{b}^{p,I,m}_{target} = \mathbf{b}^{p,I}_{target}[\mathbf{b}^{p,m}], \mathbf{b}^{p,I,m}_{actual} = \mathbf{b}^{p,I}_{actual}[\mathbf{b}^{p,m}]$ `// Mask invalid pairs`
  $\mathbf{p}^{max} =$ `argmax`$(2 \cdot \mathbf{b}^{p,I,m}_{target} - \mathbf{b}^{p,I,m}_{actual})$ `// Get furthest pairs`
  $p^{chosen} =$ `random_choice`$(\mathbf{p}^{max})$ `// Randomly choose pair`
  `update_trackers`(f, p^{chosen}) `// Assign flow to pair`
end for

$$\mathbf{b}^{p,m}_i = \begin{cases} 0 & \text{if } \mathbf{b}^{p,c}_i - f^s < 0 \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

We then apply this pairs mask to filter out any invalid pairs, thus ensuring that any pair chosen from here on would meet the requirements of the second pass of [2] and also reducing the time complexity of the `argmax` operation below in Equation 2 (since the number of candidate pairs is now reduced). Next, we take the masked candidate pairs' current distances from the target information loads, $\mathbf{b}^{p,I,m}_{target} - \mathbf{b}^{p,I,m}_{actual}$, shift them by $\mathbf{b}^{p,I,m}_{target}$ in order to retain any skewness in $\mathbb{P}^N$ for as long as possible given the overall DC load specified, and find the pairs in this masked subset which are furthest from their target information loads, $\mathbf{p}^{max}$:

$$\mathbf{p}^{max} = \text{argmax}\left(2 \cdot \mathbf{b}^{p,I,m}_{target} - \mathbf{b}^{p,I,m}_{actual}\right) \tag{2}$$

In order to avoid any bias towards smaller pair indices and create the fade phenomenon in the resultant traffic heat map [2], we randomly choose a pair $p_{chosen} \in \mathbf{p}^{max}$ to which to allocate the flow f, thus meeting the requirements of the first pass of [2]. Finally, we update the current total information vector's element for the chosen pair, $\mathbf{b}^{p_{chosen},I}_{actual}$, and the remaining capacity vector elements $\mathbf{b}^{p,c}$ for any pairs $p \in P$ which share either a source or a destination with p_{chosen}. The pseudocode for this vectorised packer is summarised in Algorithm 1.

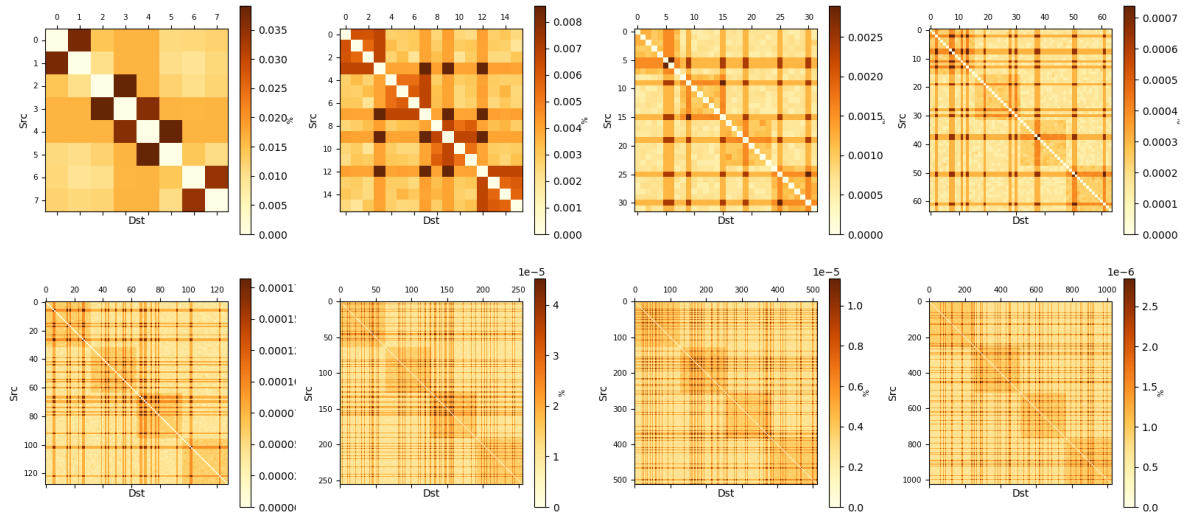

Fig. 2: Custom traffic matrix distributions generated with 8, 16, 32, 64, 128, 256, 512, and 1024 nodes, where the colour of each source-destination pair corresponds to the fraction of the overall network load it requests.

3. Simulation Setup

TrafPy enables the production of custom traffic distributions through the control of a handful of parameters. These include the flow size and inter-arrival time distribution parameters, the node distribution's inter- vs. intra-rack and skew node fractions, and the overall network load. To measure the packing times for the original and vectorised packing algorithms, we generated an assortment of custom traffic patterns typical for a 'university' DC[2] as detailed by [2] for networks with 4 racks and $|N| = \{8, 16, 32, 64, 128, 256, 512, 1024\}$ nodes (see Figure 2). We assumed an optical DC network with with an overall network load rate of 50%. For each traffic matrix, we generated $|F| = 5 \cdot |N|^2$ flows to ensure non-sparse packing. Each packing algorithm was ran on a shared cluster with an Intel Xeon ES-2660 CPU across 4 seeds to ensure reliable packing times given the variance in use of the shared cluster, with the 95% confidence interval bands plotted for any metrics recorded.

4. Results & Discussion

Figure 1b shows the packing times taken by the original and the vectorised packers when generating the distributions shown in Figure 2. The vectorised packer achieved a $\approx 38\times$ speed-up over the original packer on the $|N| = 264$ traffic matrix and $\approx 70\times$ on the $|N| = 1024$ matrix. Although the vectorised algorithm was slightly slower than the original packer on the smallest $|N| = 8$ network due to performing a `where` vector operation on all pairs, the absolute generation time was still $O(s)$ and this additional overhead quickly becomes negligible across $|N| > 8$ networks.

To verify that our proposed vectorised packing algorithm was generating the same node distributions as the original packer used by [2], we measured the JSD between the target and the generated node distributions for each algorithm (see Figure 1c). As expected, both packers deterministically reach the same solution, but the Jensen-Shannon distance will not be exactly 0 for either due to the incompatability between the 50% network load and the skewed target distribution (see [2] for more details).

In conclusion, we have proposed a flow source-destination pair assignment algorithm which makes novel use of vector array operations to achieve orders of magnitude faster traffic generation times than the original algorithm used by [2] when generating custom traffic matrices. This work significantly improves the utility of an open source traffic generation framework in order to aid the production of high-fidelity traffic patterns and to test and develop network systems at scale.

Acknowledgements: EPSRC Distributed Quantum Computing and Applications EP/W032643/1; the Innovate UK Project on Quantum Data Centres and the Future 10004793; OptoCloud EP/T026081/1; TRANSNET EP/R035342/1; the Engineering and Physical Sciences Research Council EP/R041792/1 and EP/L015455/1; the Alan Turing Institute; and Horizon Europe Dynamos.

References

1. M. Khani et al., "Sip-ml: High-bandwidth optical network interconnects for machine learning training," SIGCOMM (2021).
2. C. Parsonson et al., "Traffic generation for benchmarking data centre networks," Opt. Switch. Netw. (2022).
3. M. Alizadeh et al., "pfabric: Minimal near-optimal datacenter transport," SIGCOMM (2013).
4. W. Bai et al., "Enabling ecn in multi-service multi-queue data centers," NSDI (2016).

[2]University DCs service applications such as database backups, distributed file system hosting, and multicast video streaming, with $\approx 70\%$ of traffic being inter-rack and $\approx 20\%$ of nodes requesting $\approx 55\%$ of the traffic load.