Analyzing the Communication Clusters in Datacenters*

Klaus-Tycho Foerster
klaus-tycho.foerster@tu-dortmund.de
TU Dortmund
Dortmund, Germany

Thibault Marette
marette@kth.se
KTH
Stockholm, Sweden

Stefan Neumann
neum@kth.se
KTH
Stockholm, Sweden

Claudia Plant
claudia.plant@univie.ac.at
Faculty of Computer Science and ds:Univie, University of Vienna
Vienna, Austria

Ylli Sadikaj
ylli.sadikaj@univie.ac.at
Faculty of Computer Science and UniVie Doctoral School Computer Science, University of Vienna
Vienna, Austria

Stefan Schmid
stefan.schmid@tu-berlin.de
TU Berlin
Berlin, Germany

Yllka Velaj
yllka.velaj@univie.ac.at
Faculty of Computer Science, University of Vienna
Vienna, Austria

ABSTRACT

Datacenter networks have become a critical infrastructure of our digital society and over the last years, great efforts have been made to better understand the communication patterns inside datacenters. In particular, existing empirical studies showed that datacenter traffic typically features much temporal and spatial structure, and that at any given time, some communication pairs interact much more frequently than others. This paper generalizes this study to communication groups and analyzes how *clustered* the datacenter traffic is, and how stable these clusters are over time. To this end, we propose a methodology which revolves around a biclustering approach, allowing us to identify groups of racks and servers which communicate frequently over the network. In particular, we consider communication patterns occurring in three different Facebook datacenters: a Web cluster consisting of web servers serving web traffic, a Database cluster which mainly consists of MySQL servers, and a Hadoop cluster. Interestingly, we find that in all three clusters, small groups of racks and servers can produce a large fraction of the network traffic, and we can determine these groups even when considering short snapshots of network traffic. We also show empirically that these clusters are fairly stable across time. Our insights on the size and stability of communication clusters hence uncover an interesting potential for resource optimizations in datacenter infrastructures.

*Authors ordered alphabetically.

WWW '23, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04.
https://doi.org/10.1145/3543507.3583410

CCS CONCEPTS

• **Networks** → **Data center networks**; **Network dynamics**; • **Computing methodologies** → **Cluster analysis**.

KEYWORDS

Data Center, Clustering, Network Traffic.

ACM Reference Format:
Klaus-Tycho Foerster, Thibault Marette, Stefan Neumann, Claudia Plant, Ylli Sadikaj, Stefan Schmid, and Yllka Velaj. 2023. Analyzing the Communication Clusters in Datacenters. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583410

1 INTRODUCTION

With the popularity of data-centric and distributed applications, for example in the context of artificial intelligence, datacenter networks have become a critical infrastructure of our digital society. Indeed, these applications led to an explosive growth of communication traffic over the last years, especially inside datacenters, pushing datacenter networks to their capacity limits [32, 33].

Interestingly, however, datacenter traffic is not only growing quickly, but also features much structure. Studying packet traces collected from networking applications, researchers have found that datacenter traffic matrices are often sparse and skewed [1, 6], and exhibit locality [9], also over time [38, 39]. In other words, packet traces from real world applications are generally far from arbitrary or random, but are of fairly low entropy [2, 16, 20, 32].

The existence of such structure in communication traffic is attractive, and may be exploited for network provisioning and infrastructure optimizations [23]. Indeed, the networking community is currently putting great effort into designing protocols and algorithms to optimize different layers of the networking stack to leverage the traffic structure. These efforts include, e.g., learning-based traffic engineering [35] and video streaming [26], self-adjusting optical

networks [5, 15], or self-driving networks [21]. For instance, many network optimizations exploit the presence of elephant flows [3].

This paper aims at an understanding of the *clustered nature* of the communication traffic, going beyond the typically considered pairwise interactions [7, 16, 22, 32] and looking into communications among *groups*. In particular, we wonder whether the communication traffic matrices typically observed in empirical studies feature clusters of *dense* communication. We ask:

- How can communication clusters be efficiently found algorithmically?
- How large are communication clusters in datacenters, and how is their size distributed?
- How stable are these communication clusters over time?

The answers to these questions have important implications on the optimizability of resource allocations in datacenters [19]: dense communication clusters may be allocated locally in the datacenter (e.g., in the same rack or pod), which can significantly reduce communication overheads and improve throughput. Furthermore, stable clusters over time are attractive as frequent reoptimizations (i.e., reconfigurations such as, migrations or topological adaptions [17]) can be avoided.

The communication clusters we identify consist of groups of senders and receivers that transmit a lot of data between each other. As the groups of senders and receivers may be different, this introduces an *asymmetry* in the clustering that we will have to take into account in our methods. Additionally, it is highly likely that some nodes appear in *multiple* clusters, e.g., some nodes may be included in multiple receiver clusters because they require more information than other nodes in the network.

We propose a methodology based on a biclustering approach, allowing us to find the communication clusters described above and allowing us to efficiently identify groups of racks and servers which communicate frequently over the datacenter network. Our approach is optimized toward the standard precision and recall metrics, whose definitions we adapt to fit the network application scenario, allowing us to study the cluster similarity over time.

A merit of our approach is that it only requires very little data: we only require access to the Top $x\%$ of endpoint pairs that create the most traffic. In particular, we do not require the exact amount of traffic sent between these endpoints, which is difficult to obtain in practice [10]. Additionally, our method does not require any additional knowledge about the application running in the datacenter.

We evaluate our method in an extensive *case study*. Since it is known that the amount of structure available in a communication traffic depends on the application [4, 32], we consider three different traffic traces: one from a Web cluster, one from a Database cluster, and one from a Hadoop cluster. The Web cluster consists of web servers serving web traffic, the Database cluster mainly consists of MySQL servers, and the Hadoop cluster is used for batch processing. These traces have been made available to the research community by Facebook [37], and are currently the largest datacenter data sets available to the research community. We believe that our insights generalize to many other datacenters as well, since the applications we consider are widely deployed in practice.

We find that our approach can indeed efficiently identify high-quality traffic clusters, and we make several interesting observations. In particular, we find that *small* groups of racks and servers often produce large fractions of the network traffic, and our approach can determine these groups even when considering short snapshots of network traffic. We also show empirically that these clusters are very stable across many time steps.

In summary, ***our contributions*** are:

- We present a systematic and efficient approach to identify possible dense clusters in communication traffic.
- We propose a methodology for employing precision and recall metrics in the context of network traffic clustering. To the best of our knowledge, this is the first study of these metrics in this context.
- We report on an extensive evaluation based on actual datacenter packet traces (Web, Database, and Hadoop), showing that our approach is efficient and can find high-quality clusters.
- We uncover that small clusters are responsible for a significant amount of network traffic, and that these clusters are stable over time. This suggests that an optimized allocation of communication endpoints in datacenters may have a significant impact on the overall communication cost.
- Our methodology only requires a list of endpoint pairs which cause a lot of traffic. We show empirically that this gives good clusterings even when taking into account the absolute amounts of traffic.
- As a contribution to the research community, and to ensure reproducibility and facilitate follow-up work, our implementations are available as open-source software.[1] Our experimental datasets and artifacts are available upon request.

2 FINDING HI-QUALITY TRAFFIC CLUSTERS

Now we describe our methodology. We start with a high-level overview and then present the full details and parameter settings.

2.1 High-Level Overview

We start by giving an overview of our methodology and present an illustration of our approach in Figure 1. The details and formal definitions follow in Section 2.2.

We assume that our input is a *traffic trace*, i.e., we obtain a sequence of tuples (u, v, w, T) indicating that node u sent w bytes to receiver v at a timepoint T.

Based on the traffic trace, we create a sequence of contiguous and disjoint *time steps* $[T_i, T_{i+1})$. For each time step, we generate a *real-valued traffic matrix*, where each entry (u, v) corresponds to the total amount of traffic that was sent from node u to v in the entire timeframe $[T_i, T_{i+1})$.

For each real-valued traffic matrix, we create a *binary matrix* by thresholding, i.e., we set the Top $x\%$ largest entries in the traffic matrix to 1 and we set all other entries to 0. Note that this corresponds to having a list of sender–receiver pairs that cause the highest amounts of traffic during the time step.

Next, for each time step we apply a biclustering algorithm on the time step's binary matrix to obtain a *biclustering*. Informally (see below for a formal definition), the biclusterings reveal the groups

[1] https://github.com/tmarette/datacenterClusterAnalysis

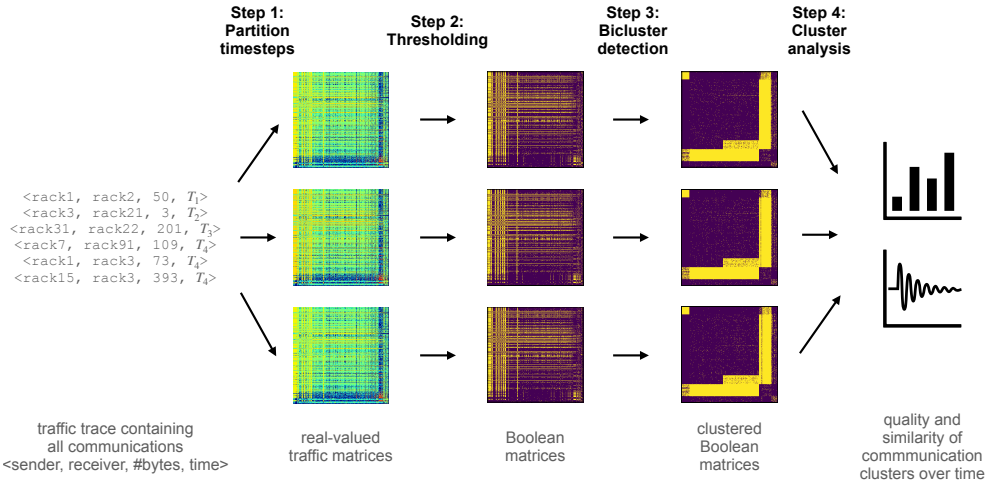


Figure 1: An overview of our approach. Given a traffic trace, we partition it into contiguous time steps for which we create real-valued traffic matrices, where entries (u, v) correspond to the amount of traffic sent from node u to v. For each traffic matrix, we create a binary matrix indicating the Top x% of sender–receiver pairs that created the most traffic. We apply our biclustering algorithms on each of these binary matrices. We subsequently evaluate the resulting clusterings over time, also taking into account the true amount of traffic that was sent.

of senders and receivers that communicate a lot during the time step and they correspond to the communication clusters that we are looking for. See Section 2.2 for a description of the biclustering algorithms that we used.

Clearly, the above approach has merits, as well as drawbacks. The main drawback is that we obtain our communication clusters based on the binary matrices and not on the original traffic matrices; indeed it is not clear whether the binary (rounded) traffic reveals anything about weighted traffic. Hence, in our evaluation we need to ensure that this approach allows us to draw conclusions about the patterns in the weighted traffic. However, if we can show that this is the case, this is also a merit of our method: in practice, it often creates significant overhead to store the entire traffic trace and therefore people only subsample it [10, 32]. Our approach, on the other hand, can be used even when we only obtain the binary matrices as input. In other words, for our approach it suffices to know the Top x% communication partners which can be obtained more efficiently, for instance, using heavy hitters data structures [12, 29, 36]. This might make our methodology more applicable in settings in which one aims to understand communication clusters without creating too much overhead for the network.

We introduce three different measures to study the quality of our clusters. First, we define versions of *precision* and *recall* to understand how well our communication clusters capture the information in the binary matrices. Second, we study the *traffic inside the biclustering* to understand how well our biclusters, that were derived on the binary matrices, represent the actual traffic from the real-valued traffic matrices. This measure compares the traffic sent between sender–receiver pairs from the biclusters with the total traffic. Our experiments show that we typically obtain high values for all three measures, indicating that we find high-quality clusters for the binary matrices *which also capture the real-valued traffic information.*

Furthermore, we introduce a similarity measure to study the similarity of different biclusterings. This allows us to compare the communication clusters that we obtain at different time steps. Our experiments will show that these communication clusters are typically highly stable over time.

2.2 Detailed Approach

Next, we describe the implementation of our approach in full detail. It will be convenient for us to consider unweighted/weighted graphs rather than binary/real-valued matrices, as this allows us to streamline our exposition. This is equivalent to using matrices by identifying graphs with their adjacency matrices.

From traffic traces to weighted graphs. We assume we are given a packet trace, or more specifically, a sequence of packet headers containing a timestamp and the amount of traffic that server/rack u sent to server/rack v at a certain time. To study how the traffic patterns develop over time, we partition the traffic into time intervals of a certain duration, and we will refer to each of these time intervals as *time steps*. For the following discussion we assume that the duration d of a time step is fixed. This simplifies our notation.

We represent the traffic during each time step t as a weighted bipartite *communication graph* $G_t = (U \cup V, E_t, w_t)$. Here, U is the set of all racks/servers that send data over the network during the *entire* time duration; similarly, V is the set of all racks/servers that receive data during the entire time. We stress that if a rack/server sends *and* receives data, it has two nodes in the graph: one in U, and one in V. Furthermore, E_t is the set of edges at time step t. The weight function $w_t : U \times V \to \mathbb{R}_+$ assigns to each pair (u, v) the total amount of traffic that rack/server u sent to rack/server v during time step t. We chose to represent the traffic as a bipartite graph as finding clusters in bipartite graphs is a well-studied problem [14, 28] and it allows us to find non-trivial patterns of datacenter traffic.

From weighted graphs to unweighted graphs. As some of our algorithms expect *un*weighted graphs, we also consider the following *thresholded* unweighted graphs $G_t^\tau = (U \cup V, E_t^\tau)$. Here, $\tau \in \mathbb{R}$ is a threshold such that G^τ only contains edges of weight at least τ, i.e., $E_t^\tau = \{(u,v) \in E_t : w(u,v) \geq \tau\}$. This can be interpreted as keeping all edges (u,v) for which u sent "a lot" of traffic to v.

Typically, we set the threshold τ such that a desired fraction of edges from G_t remain. For example, if we set τ to the 70%-percentile of the non-zero edge weights in G_t, then G_t^τ contains the largest 30% of non-zero weight edges from G_t. When setting τ to one of these percentiles, we write τ_p to denote the p-percentile for non-zero weight edges in G_t for $p \in (0,1)$. Note that if we pick large values for p (e.g., $p = 70\%$), this results in sparser graphs $G_t^{\tau_p}$, and smaller values for p (e.g., $p = 30\%$) result in denser graphs $G_t^{\tau_p}$. For $p = 0\%$, we keep all non-zero weight edges in $G_t^{\tau_p}$.

We discuss the choices for τ that we use in our case study in Section 2.3.

Clustering bipartite graphs. To find traffic patterns inside our unweighted graphs, we propose a *biclustering* approach. More concretely, given an unweighted bipartite graph $G = (U \cup V, E)$, our algorithms compute a *biclustering* of G, i.e., they return sets of biclusters $(U_1, V_1), \ldots, (U_k, V_k)$ such that $U_i \subseteq U$ and $V_i \subseteq V$ for all $i = 1, \ldots, k$.

Our algorithms (more details follow) pick the biclusters in such a way that the induced subgraphs $G[U_i, V_i]$ are dense; this corresponds to sets of senders and receivers that communicate a lot of data in the network. In other words, the nodes in a cluster V_i receive "a lot" of traffic from vertices in U_i.

In our experiments, we compute a biclustering for each time step t. More concretely, we apply our biclustering algorithms on each graph G_t^τ for $t = 1, \ldots, T$, where T is the total number of time steps. This yields biclusterings $(U_1^t, V_1^t), \ldots, (U_k^t, V_k^t)$ for each time step $t = 1, \ldots, T$.

See Section 2.3 for the concrete algorithms that we use.

Measuring the quality of traffic clusters. Next, we introduce the measures that we use to obtain insights into our data and to assess the quality of the clusterings that we obtain.

Comparing graph snapshots. To compare two (unweighted) graph snapshots G_t^τ and G_{t+1}^τ, we consider three different measures:

(1) common= $|E_t \cap E_{t+1}| / |E_t \cup E_{t+1}| \cdot 100$,
(2) appear= $|E_{t+1} \setminus E_t| / |E_t \cup E_{t+1}| \cdot 100$,
(3) disappear= $|E_t \setminus E_{t+1}| / |E_t \cup E_{t+1}| \cdot 100$.

These measures represent the percentage of edges that appear in two consecutive graph snapshots, the percentage of edges that are new, and the percentage of edges that disappear, resp.

Evaluating the quality of biclusterings. To evaluate the quality of our biclusterings, we consider three different measures which, taken together, give us a differentiated picture on the quality of the traffic clusters. While biclustering approaches have been applied successfully in many applications in computer science, evaluating these approaches typically requires an external validation which is not available in our problem setting. An interesting measure for the validity of a biclustering under variations of the input data set is the stability index by Lee et al. [24], which is relevant for finding biclusters in data with numerical attributes. With the following definitions of *traffic inside the biclustering, recall and precision*, we introduce a methodology to measure the quality of traffic clusters that originate from thresholded communication graphs.

First, for a biclustering $(U_1, V_1), \ldots, (U_k, V_k)$ we say that its *induced bicliques* are given by the set

$$\mathrm{bicl}((U_1, V_1), \ldots, (U_k, V_k)) = \bigcup_{i=1}^{k} (U_i \times V_i), \tag{1}$$

i.e., $\mathrm{bicl}((U_1, V_1), \ldots, (U_k, V_k)) \subseteq U \times V$ contains all edges (u, v) with $u \in U_i$ and $v \in V_i$ for some i. Intuitively, the induced bicliques are "idealized" versions of the the original biclusters, in which all missing edges were added.

The *traffic inside the biclustering* $(U_1, V_1), \ldots, (V_k, V_k)$ is the fraction of traffic sent between vertices that are contained in one of the biclusters (U_i, V_i). More formally, it is given by

$$\frac{1}{W} \sum_{(u,v) \in \mathrm{bicl}((U_1,V_1),\ldots,(U_k,V_k))} w_t(u,v),$$

where $W = \sum_{(u,v) \in E_t} w_t(u,v)$ is the total sum of edge weights (corresponding to the total amount of network traffic at time step t). Note that here we are summing over the *weights* from the original weighted graph. Hence, if the traffic inside the biclustering is large, this indicates that the biclustering captures the traffic inside G well.

However, the traffic inside a biclustering has some limitations. For example, consider a biclustering (U_1, V_1) with $U_1 = U$ and $V_1 = V$. Then the traffic inside this biclustering is equal to the total sum of weights but the biclustering is not very informative, as it considers the whole graph as a single bicluster. Therefore, we also consider *recall* and *precision*, which are defined for unweighted graphs $G_t^\tau = (U \cup V, E_t^\tau)$.

The *recall* measures the fraction of edges (u, v) of G^τ that are "covered" by the induced bicliques of the biclustering $(U_1, V_1), \ldots, (V_k, V_k)$. More formally, the recall is given by

$$\mathrm{recall} = |E_t^\tau \cap \mathrm{bicl}((U_1, V_1), \ldots, (U_k, V_k))| / |E_t^\tau|.$$

On the other hand, the *precision* measures the fraction of edges in the induced bicliques which also have corresponding edges in G^τ. More formally, the precision is given by

$$\mathrm{precision} = \frac{|E_t^\tau \cap \mathrm{bicl}((U_1, V_1), \ldots, (U_k, V_k))|}{|\mathrm{bicl}((U_1, V_1), \ldots, (U_k, V_k))|}.$$

Intuitively, the precision is high if the biclusters are "not too large" and the recall is high if the biclusters "cover" the edges well. Hence, if we find biclusters that *simultaneously* have high recall, high precision and a lot of traffic inside the biclusters, this indicates that the biclusters capture the traffic structure inside our communication graph well.

Comparing biclusterings across time steps. Next, recall that the biclusterings we computed depended on the time step t. As we are interested in comparing biclusterings from different time steps, we need to measure their *similarity*.

For two time steps t_1 and t_2 with biclusterings $(U_1^{t_1}, V_1^{t_1}), \ldots, (U_{k_1}^{t_1}, V_{k_1}^{t_1})$ and $(U_1^{t_2}, V_1^{t_2}), \ldots, (U_{k_2}^{t_2}, V_{k_2}^{t_2})$, we say that their *similarity* $\mathrm{sim}(t_1, t_2)$ is given by the fraction of edges which are contained in both induced bicliques. More formally, let $\mathrm{bicl}_1 = \mathrm{bicl}((U_1^{t_1}, V_1^{t_1}), \ldots, (U_{k_1}^{t_1}, V_{k_1}^{t_1}))$ and $\mathrm{bicl}_2 = \mathrm{bicl}((U_1^{t_2}, V_1^{t_2}), \ldots, (U_{k_2}^{t_2}, V_{k_2}^{t_2}))$. Then

their similarity is given by

$$\mathrm{sim}(t_1, t_2) = \frac{|\mathrm{bicl}_1 \cap \mathrm{bicl}_2|}{|\mathrm{bicl}_1 \cup \mathrm{bicl}_2|}.$$

Intuitively, two biclusterings are similar if their idealized bicliques are similar. While this definition might look a bit artificial at first glance, it is quite handy because it allows us to deal with overlapping biclusters and also with biclusterings that have different numbers of biclusters.

To compare $T > 2$ different biclusterings, we turn to *similarity matrices* (see, e.g., Fig. 3(d)). Given multiple biclusterings $(U_1^t, V_1^t), \ldots, (U_k^t, V_k^t)$ for $t = 1, \ldots, T$, the *similarity matrix* $S \in [0,1]^{T \times T}$ has entries $S_{t_1,t_2} = \mathrm{sim}(t_1, t_2)$. Thus, S_{t_1,t_2} measures how similar the biclusterings t_1 and t_2 are. As S is symmetric, we only report the entries for $t_2 \geq t_1$.

Visualization. Sometimes we visualize the biclustering we obtain. Given a bipartite graph $G = (U \cup V, E)$ with biadjacency matrix $B \in \{0,1\}^{|U| \times |V|}$ and a biclustering $(U_1, V_1), \ldots, (U_k, V_k)$, we reorder the rows and columns of B using the ADVISER algorithm [11]. In our plots (see, e.g., Fig. 2), yellow dots correspond to 1-entries in B and purple dots correspond to 0-entries in B. We follow the convention that vertices from U correspond to rows of B and vertices from V correspond to columns of B; in other words, the rows of B correspond to nodes that send data and the columns of B correspond to nodes that receive data.

2.3 Choice of Parameters and Algorithms

We conclude this section by justifying our choices for the parameter τ and the biclustering algorithms we use in the case study.

Choice of parameter τ. Recall that τ_p determines the sparsity of the graph G^{τ_p}, where larger (smaller) values for p correspond to sparser (denser) graphs. Furthermore, when running biclustering algorithms on sparser graphs, this typically results in smaller biclusters (i.e., the biclusters contain fewer vertices and the induced bicliques are smaller). Intuitively, if we pick p too small (corresponding to dense graphs and large biclusters), our biclusterings have high recall and high traffic inside the biclusters but small precision. Similarly, when p is too large (the biclusters are too small), we have high precision but small recall and small traffic inside the matrices.

As we argued before, we are interested in finding biclusterings which simultaneously have high recall, high precision and high traffic inside the biclustering. Therefore, in our experiments we pick the value of $p \in \{0\%, 30\%, 50\%, 70\%\}$ such that all of these three objectives are satisfied (if possible). If multiple values in $\{0\%, 30\%, 50\%, 70\%\}$ satisfy all three criteria, we report the largest possible value for p, since this corresponds to biclusters with fewer vertices which might be easier to optimize in application settings.

In preliminary experiments we also used other options with $p \notin \{0\%, 30\%, 50\%, 70\%\}$ but they did not reveal any new insights beyond what we report. Hence, we focus on $p \in \{0\%, 30\%, 50\%, 70\%\}$.

Biclustering algorithm. The first biclustering algorithm we use is the *pcv algorithm* [30]. This algorithm takes as input an *unweighted* bipartite graph G and a parameter k, and returns a biclustering $(U_1, V_1), \ldots, (U_k, V_k)$. The biclustering is such that the clusters $U_1, \ldots, U_k$ partition U, but the clusters $V_1, \ldots, V_k$ might overlap and their union does not have to equal V. In a nutshell, the algorithm computes the biadjacency matrix B of G and denoises B via a truncated rank k-SVD and then applies k-Means on the rows of the low-rank matrix to obtain the clusters $U_1, \ldots, U_k$. Then it finds the clusters V_i by looking at the submatrices $B[U_i, :]$ and setting V_i to the set of columns in $B[U_i, :]$ with "many" non-zero entries.

We used pcv because it processes graphs with tens of thousands of nodes highly efficiently and, even though it is a randomized algorithm, its outputs are very consistent over different runs (see App. A.2). Furthermore, it allows overlapping column-clusters which is necessary given the structure of our dataset. However, other methods that find overlapping clusters could have been used as well, for instance, algorithms for Boolean Matrix Factorization [27, 28]. In our experiments, we set $k = 7$ for pcv.

In addition to pcv, we also use *GraphScope* [34]. Unlike pcv which only considers a single time step, GraphScope is an adaptive mining scheme on time-evolving graphs, i.e., it takes as input all unweighted graphs $G_1^\tau, \ldots, G_T^\tau$. GraphScope requires no user-defined parameters, and it operates completely automatically, based on the MDL (Minimum Description Language) principle [31]. It simultaneously finds the communities and determines *change-points* when the cluster structure changes. In particular, given a bipartite graph, where one group of nodes represent the source nodes and the other the destination nodes, GraphScope treats source and destination nodes separately, and discovers separate source and destination partitions, i.e., it determines a partition of the sources into s groups and a partition of the destinations into d groups, where s and d can have different values. GraphScope starts by identifying the communities on the initial graph snapshot, and then compares the current structure to consecutive snapshots. If this structure does not change much over time, consecutive snapshots of the evolving graphs are grouped together into a *time-segment*. If a new graph snapshot cannot fit well into the old segment, GraphScope introduces a change-point, and starts a new segment at that time-stamp. Those change-points often detect drastic discontinuities in time. The advantage of applying GraphScope is that it requires no parameters, i.e., it determines the number of row and column groups automatically. However, it is less scalable than pcv; in particular, it is too inefficient to process rack–server or server–server communications in our case study, and does not allow for overlapping column clusters V_i.

We apply these two different algorithms on all unweighted graphs G_t^τ for rack–rack communication. We empirically show that they find similar clusters, indicating that the clusters we find are true positives.

3 A CASE STUDY

As a case study, we consider the traces obtained from three different clusters in the Altoona datacenter, which were shared with the research community by Facebook [32, 37]. The clusters correspond to a Web cluster, a Database cluster and a Hadoop cluster. We consider the traffic over a 150 minutes (2.5 hours) time frame, plus some additional time period in the beginning to avoid possible artifacts from the "warm-up phase" and to focus on the steady-state behavior. To understand how the traffic patterns develop over time, we partition the 150 minutes of traffic into disjoint time intervals of 1, 5, and 15 minutes, the *time steps*; there are hence 150, 30, and 15 time steps, resp.

3.1 Web Cluster

This cluster contains Web servers serving Web traffic.

Properties of datasets. To analyze how similar two consecutive graphs G_t and G_{t+1} are, we compute the percentages of edges that appear, disappear, and do not change between E_t and E_{t+1} in the rack–rack communication over a 15 minutes interval for τ_0, $\tau_{0.3}$, $\tau_{0.5}$, and $\tau_{0.7}$. The graphs exhibit a strong similarity over time, and the percentage of edges that E_t and E_{t+1} share is very high, i.e., greater than 70%. The results for the 5 and 1 minutes interval are similar.

For the rack–server communication, we notice that the results are similar to those for the rack–rack communication. The only exception is represented by the sparsest graphs, i.e., $p = 70\%$, for the 15 minutes time interval where the percentage of common edges is lower than the percentage of appearing and disappearing edges. For the server–rack communication the similarity over time is analogous to that of the rack–rack communication for 15 and 5 minutes time interval, i.e., the percentage of common edges is higher than the percentage of appearing and disappearing edges, except for $p = 70\%$ in the 5 minute time interval. For the 1 minute time interval with $\tau_{0.5}$ and $\tau_{0.7}$, instead, the similarity changes and we notice that 40% of the edges are appearing or disappearing and only a small number of edges remain in two consecutive graph snapshots. (We present additional Figures in the appendix.)

Discussion of results. We start by discussing our results that we obtained using pcv. In Fig. 2 we present the visualization of the rack–rack communication over a typical 1 minute interval. We used $\tau_{0.7}$, i.e., we kept the 30% largest non-zero weight edges in G^{τ_p}. The plot clearly allows us to identify the cache servers, the Web servers and the multifeed servers.

Next, we need to ensure that the nice visual clusters from Fig. 2 indeed correspond to actual traffic patterns. We analyze this in Fig. 3. First, we need to ensure that the biclusterings returned by pcv have high recall, precision, and traffic inside the biclusters. Fig.s 3(a)–3(c) show that indeed all of these measures are very high, for time steps of lengths 15, 5, and 1 minutes, resp. For each time step, the clusters contained more than 80% of the total traffic, while having recall and precision values of at least 90%. Hence, the biclusters seem to correspond to real traffic patterns. Second, we try to understand how stable the biclustering results are over time. Fig.s 3(d)–3(f) provide the similarity matrices for time steps of lengths 15, 5, and 1 minutes, resp. The plot shows that the clusterings of the different time steps have very high similarity and thus the obtained cluster structures are very stable over time, with similarity values over 85%.

Table 1: Our findings for the Web cluster and Database using GraphScope for rack-rack traces. (S - segments, s - #clusters for sources, d - #clusters for destinations)

Clusters	Web cluster			Database		
	15′ S, s, d	**5′ S, s, d**	**1′ S, s, d**	**15′ S, s, d**	**5′ S, s, d**	**1′ S, s, d**
p=0%	1, 7, 7	1, 8, 9	1, 7, 6	1, 8, 8	2, 8, 8	43, 11, 11
p=30%	3, 7, 12	1, 7, 7	1, 6, 5	2, 10, 9	3, 12, 9	46, 1, 9
p=50%	4, 10, 9	1, 6, 7	1, 5, 6	3, 8, 7	7, 8, 8	45, 8, 8
p=70%	1, 7, 13	1, 7, 7	12, 12, 16	2, 9, 6	9, 8, 8	40, 8, 10

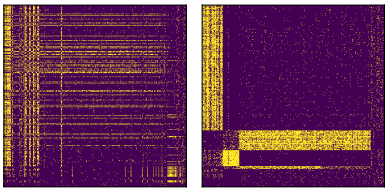


Figure 2: Visualization of our approach for rack–rack communication over a 1 minute interval from the Web cluster for $p = 70\%$. On the left, we present the unordered binary matrix. On the right, we present the binary matrix after reordering based on the bicluster structure. The cluster structure reveals the cache servers (vertical block), the Web servers (horizontal block) and the multifeed servers (small square).

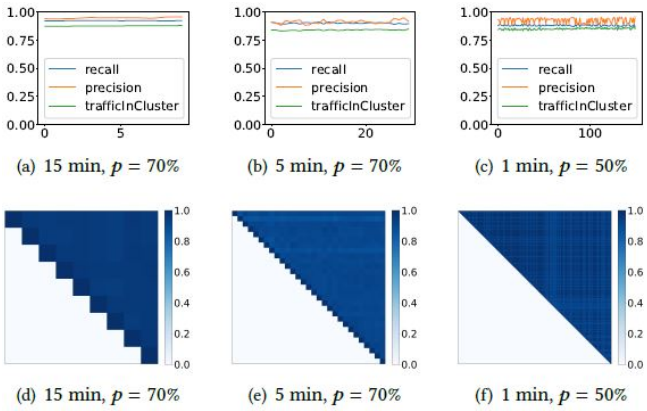


Figure 3: Web cluster and rack–rack communication results, with $p = 70\%$ and time step lengths varying from 15 minutes to 1 minute. Fig.s 3(a)–3(c) present recall, precision and traffic inside the clusters, and Fig.s 3(d)–3(f) present the corresponding similarity matrices.

In Table 1 we report the results of GraphScope for the Web cluster. In the Web cluster the graphs are more similar and few change-points are identified by GraphScope, i.e., the number of segments is equal to 1 in most of the cases, for all the time intervals. The results of GraphScope, for some aspects, are in line with our analysis of the graph similarity in terms of common, appearing, and disappearing edges. The Web cluster is characterized by a well defined structure that remains persistent over time.

It is important to underline that the results obtained with GraphScope differ from those obtained with the pcv in terms of number of clusters and segments detected. This difference is due to the different solutions provided by the algorithm: GraphScope does not identify overlapping communities and, unlike pcv, the nodes are partitioned and belong to only one cluster.

Next, we consider rack–server communication over 5 minute time frames and $p = 0\%$. In Fig. 9(a) we visualize a typical biclustering. We see that the resulting biclusters are extremely dense, which is why our biclusterings have a precision of almost 100%. The recall is around 85% and the traffic inside the biclustering is about 90%.

This suggests that we found high-quality traffic biclusters. Furthermore, the biclusterings are very stable over time and all entries in the similarity matrix are over 94%. For shorter time frames of length 1 minute, the data becomes sparser and pcv does not find the full clusters anymore, which is why the traffic inside the clusters drops to around 60% in this case. For server–rack communication the situation is almost identical.

For the server–server communication, we consider 15 minute time frames and $p = 0\%$. These are the densest graphs that we obtain among all clusters, containing around 10% of all possible edges; for shorter time frames or larger values for p, the graphs become much sparser (e.g., for 5 minute intervals and $p = 0.5\%$, they contain less than 2.1% of all possible edges). The typical biclustering we find clearly reveal the structure of the cache and the Web servers (see Fig. 6 in the appendix). The similarities of our biclusterings are very high (typically above 99%); their recall is constantly above 95%, and their precision and traffic inside the clusters is around 50%. We explain the low fraction of traffic by the fact that pcv seems to miss some servers in the clustering (see also Fig. 6).

3.2 Database Cluster

The Database cluster mainly consists of MySQL servers which store user data and serve SQL queries.

Properties of datasets. We evaluate the similarity of graphs in the Database cluster and we notice that for the rack–rack communication, and the rack–server communication, the percentage of common edges is high, i.e., greater than 70% for the 15 and 5 minutes interval, for any p. The sparser graphs with $p = 70\%$ exhibit a less strong similarity over time and the percentage of common edges decreases to 50% for the 1 minute interval. In the server–rack communication, the graphs characterized by a lower similarity are those obtained with $\tau_{0.7}$ for the 15 and 5 minutes time interval. While for the 1 minute time interval the results are similar to those in the other communication cases with an exception for the sparsest graphs with $p = 70\%$ where the number of common edges, about 20%, is lower that that of the new edges, about 40%. (We present additional figures in the appendix.)

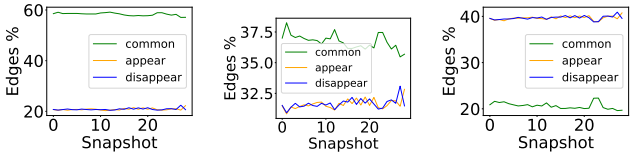
Discussion of results. We present our findings that we obtained from pcv and 1-minute time intervals in Fig. 4. We present a typical plot for the rack–rack communication over a 1-minute interval with $p = 0\%$ in Fig. 4(a). In Fig. 4(b), we present recall, precision and traffic inside the biclusters. Recall and precision are very high (almost 100%) but the precision is lower (around 80%). Indeed, when increasing p to 30% and 50%, the precision drops quite significantly to 70% and 60%, resp., but for both the recall and the traffic inside the clusters stay above 90%. However, when setting $p = 70\%$, the traffic inside the clusters drops quite significantly to around 70% and precision and recall drop to 50% and 80%, resp. In Fig. 4(c) we see that the clusterings have very high similarities of at least 85%.

We note that when considering longer time steps, consisting of 5 or 15 minutes, the precision for larger values of p increases. For example, when considering 15 minute intervals, the precision for $p = 30\%$ is above 80% and for $p = 50\%$ it is above 75%.

Applying GraphScope to the Database cluster we notice that the similarity of the graphs decreases, and the algorithm detects more change-points and segments, in particular for the 1 minute time interval, as shown in Table 1.

(a) ordered plot (b) measures (c) similarity

Figure 4: Our results for the Database cluster and rack–rack communication over 1-minute time steps and $p = 0\%$. Fig. 4(a) shows the graph after reordering according to the biclustering. Fig. 4(b) presents recall, precision and traffic inside the biclusters. Fig. 4(c) shows the similarity matrix.



(a) measures: 5 min, $p = 30\%$, Hadoop cluster rack–rack com. (b) measures: 5 min, $p = 50\%$, Hadoop cluster rack–rack com. (c) measures: 5 min, $p = 70\%$, Hadoop cluster rack–rack com.

Figure 5: Similarity of consecutive graph snapshots for the Hadoop clusters. Notice that lines may overlap.

As the Web cluster, also the Database cluster is characterized by a well-defined structure that remains persistent over time. The reason for the high number of change-points in the 1 minute time interval in Table 1 is related to the size of the graphs, these are the smallest in terms of number of edges compared to the other graphs.

Next, we consider rack–server and the server–rack communications over 15 minute time frames and for $p = 0\%$. We visualize typical communication patterns in Fig.s 9(b) and 9(c), resp. We considered the 10,000 servers that received/sent the most data over the whole 150 minutes. In both cases, there exists one bicluster that is relatively dense, even though sparser than what we have seen for the Web cluster; the rest of the graph is extremely sparse. For both, rack–server and server–rack communication, the precision and recall are around 50% across all time steps; we explain this by the sparsity of the obtained biclusters and the long "unclusterable" tail of servers that do not receive/send a lot of data. The bicluster similarity is very high (over 80%) in both cases. Intriguingly, for rack–server communication the biclusters contain 85% of the traffic, while for server–rack it contains just 15% of the traffic.

For server–server communication, we considered the 10,000 servers which sent and received the most data. We did not find any meaningful biclusters; even for 15 minute time frames and $p = 0\%$, the graphs are extremely sparse and have less than 0.12% of all possible edges.

3.3 Hadoop Cluster

This cluster is for batch processing and runs Hadoop applications.

Properties of datasets. In the Hadoop cluster, we notice a different structure compared to the previous two clusters. In particular for the rack–rack communication, in the 15 and 5 minutes time interval for $\tau_{0.7}$, the percentage of common edges is low while there is a high number of edges that appear and disappear. We show the percentages in Fig. 5(a), Fig. 5(b), and 5(c), The same happens for the rack–server communication for $\tau_{0.7}$ and 5 minutes time interval. This low similarity over time is exhibited also for the 1 minute time interval with $\tau_{0.5}$, $\tau_{0.7}$ in the rack–rack communication, and for the 5 minute time interval with $\tau_{0.7}$ in the rack–server communication. In the server–rack communication, the percentage of common edges is low while there is a high number of edges that appear and disappear, the only graphs exhibiting a strong similarity over time are those obtained with τ_0 for the 15 minutes time interval.

Discussion of results. We again start by discussing our results from pcv. We present the biclustering from rack–rack communication over 5-minute intervals and $p = 30\%$ in Fig. 8 in the appendix. Fig. 8(a) shows a typical plot of a graph $G_t^{\tau_p}$; we see that this graph is very dense, containing almost 70% of all *possible* edges. As the biclustering essentially forms one single large cluster for the graph, the precision is also around 70%, and the recall and traffic inside the cluster are very high (Fig. 8(b)). Not surprisingly, the similarities across different time steps are extremely high (Fig. 8(c)).

In Table 2 in the appendix we report the results obtained with GraphScope for the Hadoop cluster. Comparing the results for the Hadoop cluster with the other clusters, it is easy to see that in this Hadoop cluster the similarity of the graphs decreases, and the algorithm detects many more change-points and segments, in particular for the 1 minute time interval. The Hadoop Cluster is the one with the lowest percentage of common edges and highest percentage of edges that appear and disappear for the 1 minute time interval and, as shown in Table 2, GraphScope detects a high number of change-points. Moreover, it is the cluster with the highest average number of edges.

Next, we consider rack–server communication over 15 minute time intervals and $p = 30\%$. We visualize a typical biclustering in Fig. 9(d), where we considered the 10,000 servers that received the most data over the whole 150 minutes. When considering $p = 0\%$, the only change in the plots is that the dense areas become even more dense. Fig. 9(d) shows that the racks send a lot of data to a very large fraction of the servers. Our biclusterings had recall and traffic inside the biclusters almost 100%; the precision is around 50%, due to the sparsity of the graph. The smallest entry in our similarity matrices is over 85%, suggesting that the set of active servers is very stable over the entire time frame. For server–rack communication results are essentially identical.

For the server–server communication of the 10,000 most active sending and receiving servers, the matrices are relatively sparse (even over 15 minutes and for $p = 0\%$, they typically contain less than 5% of all possible edges). It seems like the data does not contain clear bicluster structures, as the recall and the traffic inside the biclusterings are almost 0%.

4 ADDITIONAL RELATED WORK

The study of communication traffic and its patterns is a topic of high relevance in the networking community, and has received much attention in the literature, at least from the 1990s when it was observed that enterprise and Internet traffic can significantly differ from other communication traffic, such as phone calls [25]. There also already exist several important studies on the traffic in datacenter networks specifically. Empirical studies found that much datacenter traffic is rack-local [6, 13], that there is typically only a small fraction of large flows [2], and that demand is generally bursty [6–8, 13] and of low entropy [4] and features ON/OFF behavior [6]. Some of these properties have recently been revisited in the context of a large empirical study in Facebook's datacenters [32], which showed that the specific properties also depend on the datacenter type. However, these works do not go in-depth for our research question on communication clustering, even though many further aspects are considered: e.g., the seminal work by Benson et al. [6] studies traffic from the viewpoints of application type, flow- and packet-arrival times, link utilization, and extra-rack vs. intra-rack traffic, but not the communication cluster structure itself. Similarly, Roy et al. [32] consider, e.g., rack locality, demand-distribution, and communication between co-located server clusters–but do not study clustering in the observed traffic. Lastly, while Ghobadi et al. [16] showed that small groups of servers can be talkative, they also did not further investigate communication clusters.

Hence, to the best of our knowledge, our work is the first to propose a systematic approach to determine communication clusters in datacenter traffic, based on biclustering, showing that a significant amount of traffic lies in small and stable clusters. Despite the practical relevance of biclustering in many applications, e.g., in text mining and bioinformatics, only few publications focus on the evaluation of biclustering results. Existing approaches based on external validation [18] require information on an ideal biclustering solution and are not applicable to our problem setting. Lee et al. [24] introduced a stability index to measure the validity of a biclustering under variations of the input data set. This measure focuses on finding biclusters in data with numerical attributes, as it is the case for gene expression.

5 FUTURE RESEARCH

We understand our work as a first step, and believe that it opens several interesting avenues for future research. In particular, it would be interesting to improve the scalability of our algorithms further, in order to be able to conduct even more fine-grained analyses. Another interesting direction is to find patterns of flow patterns (involving a sequence of senders and receivers), rather than patterns of sender–receiver that we considered. Next, the dataset we considered provides traffic information over a single day (24 hours) and is widely studied by the research community. As a consequence, we are limited in the traffic window, and hence studied a 150 minutes sample. Future studies could consider larger datasets to study shorter and larger time frames: seconds, days, weeks and months. More generally, while our work shows that small, dense, and stable clusters exist in the communication traffic, it remains to investigate the resulting possible opportunities for improving the resource efficiency in datacenters. In particular, it would be interesting to study under which circumstances a collocation of such clusters can be meaningful, possibly even dynamically over time, using cost-efficient migrations or reconfigurations.

ACKNOWLEDGMENTS

Research supported by the European Research Council (ERC), grant agreement No. 864228 (AdjustNet), Horizon 2020, 2020-2025. Stefan Neumann and Thibault Marette are supported by the ERC Advanced Grant REBOUND (834862), and the EC H2020 RIA project SoBigData++ (871042). Some of the computations were enabled by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and Swedish National Infrastructure for Computing (SNIC) partially funded by the Swedish Research Council through grant agreements no. 2022-06725 and 2018-05973.

REFERENCES

[1] 2016. ProjecToR Dataset. www.microsoft.com/en-us/research/project/projector-agile-reconfigurable-data-center-interconnect.
[2] Mohammad Alizadeh, Albert G. Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *SIGCOMM*. ACM, 63–74.
[3] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: minimal near-optimal datacenter transport. In *SIGCOMM*. ACM, 435–446.
[4] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. 2020. On the Complexity of Traffic Traces and Implications. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 1 (2020), 20:1–20:29.
[5] Chen Avin and Stefan Schmid. 2018. Toward demand-aware networking: a theory for self-adjusting networks. *Comput. Commun. Rev.* 48, 5 (2018), 31–40.
[6] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Internet Measurement Conference*. ACM, 267–280.
[7] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2010. Understanding data center traffic characteristics. *Comput. Commun. Rev.* 40, 1 (2010), 92–99.
[8] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: fine grained traffic engineering for data centers. In *CoNEXT*. ACM, 8.
[9] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. 2014. OSA: An Optical Switching Architecture for Data Center Networks With Unprecedented Flexibility. *IEEE/ACM Trans. Netw.* 22, 2 (2014), 498–511.
[10] Kimberly C. Claffy, George C. Polyzos, and Hans-Werner Braun. 1993. Application of Sampling Methodologies to Network Traffic Characterization. In *SIGCOMM*. 194–203.
[11] Alessandro Colantonio, Roberto Di Pietro, Alberto Ocello, and Nino Vincenzo Verde. 2012. Visual Role Mining: A Picture Is Worth a Thousand Roles. *IEEE Trans. Knowl. Data Eng.* 24, 6 (2012), 1120–1133.
[12] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms* 55, 1 (2005), 58–75.
[13] Christina Delimitrou, Sriram Sankar, Aman Kansal, and Christos Kozyrakis. 2012. ECHO: Recreating network traffic maps for datacenters with tens of thousands of servers. In *IISWC*. IEEE, 14–24.
[14] Inderjit S. Dhillon. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *SIGKDD*. 269–274.
[15] Klaus-Tycho Foerster and Stefan Schmid. 2019. Survey of Reconfigurable Data Center Networks: Enablers, Algorithms, Complexity. *SIGACT News* 50, 2 (2019), 62–79.
[16] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil R. Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel C. Kilper. 2016. ProjecToR: Agile Reconfigurable Data Center Interconnect. In *SIGCOMM*. ACM, 216–229.
[17] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. 2021. Cerberus: The Power of Choices in Datacenter Topology Design - A Throughput Perspective. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 3 (2021), 38:1–38:33.
[18] Blaise Hanczar and Mohamed Nadif. 2013. Precision-recall space to correct external indices for biclustering. In *ICML (2) (JMLR Workshop and Conference Proceedings, Vol. 28)*. JMLR.org, 136–144.
[19] Monika Henzinger, Stefan Neumann, and Stefan Schmid. 2019. Efficient distributed workload (re-) embedding. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 1 (2019), 1–38.
[20] Glenn Judd. 2015. Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter. In *NSDI*. USENIX Association, 145–157.
[21] Patrick Kalmbach, Johannes Zerwas, Péter Babarczi, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid. 2018. Empowering Self-Driving Networks. In *SelfDN@SIGCOMM*. ACM, 8–14.
[22] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. 2009. The nature of data center traffic: measurements & analysis. In *Proc. 9th ACM SIGCOMM Conference on Internet Measurement (IMC)*. ACM, 202–208.
[23] Wolfgang Kellerer, Patrick Kalmbach, Andreas Blenk, Arsany Basta, Martin Reisslein, and Stefan Schmid. 2019. Adaptable and data-driven softwarized networks: Review, opportunities, and challenges. *Proc. IEEE* 107, 4 (2019), 711–731.
[24] Youngrok Lee, Jeonghwa Lee, and Chi-Hyuck Jun. 2011. Stability-based validation of bicluster solutions. *Pattern Recognit.* 44, 2 (2011), 252–264.
[25] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. 1994. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Trans. Netw.* 2, 1 (1994), 1–15.
[26] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *SIGCOMM*. ACM, 197–210.
[27] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. 2008. The Discrete Basis Problem. *IEEE Trans. Knowl. Data Eng.* 20, 10 (2008), 1348–1362.
[28] Pauli Miettinen and Stefan Neumann. 2020. Recent Developments in Boolean Matrix Factorization. In *IJCAI*, Christian Bessiere (Ed.). 4922–4928.
[29] Jayadev Misra and David Gries. 1982. Finding Repeated Elements. *Sci. Comput. Program.* 2, 2 (1982), 143–152.
[30] Stefan Neumann. 2018. Bipartite Stochastic Block Models with Tiny Clusters. In *NeurIPS*. 3871–3881.
[31] Jorma Rissanen. 1978. Modeling by shortest data description. *Autom.* 14, 5 (1978), 465–471.
[32] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *SIGCOMM*. ACM, 123–137.
[33] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Hong Liu, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2016. Jupiter rising: a decade of clos topologies and centralized control in Google's datacenter network. *Commun. ACM* 59, 9 (2016), 88–97.
[34] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. 2007. GraphScope: parameter-free mining of large time-evolving graphs. In *KDD*. ACM, 687–696.
[35] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. 2017. Learning to Route. In *HotNets*. ACM, 185–191.
[36] David P. Woodruff. 2016. New Algorithms for Heavy Hitters in Data Streams (Invited Talk). In *ICDT (LIPIcs, Vol. 48)*. 4:1–4:12.
[37] James Hongyi Zeng. 2017. Data Sharing on traffic pattern inside Facebook's data center network. https://research.facebook.com/blog/2017/01/data-sharing-on-traffic-pattern-inside-facebooks-datacenter-network/. Accessed: 2022-05-10.
[38] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution measurement of data center microbursts. In *Internet Measurement Conference*. ACM, 78–85.
[39] Shihong Zou, Xitao Wen, Kai Chen, Shan Huang, Yan Chen, Yongqiang Liu, Yong Xia, and Chengchen Hu. 2014. VirtualKnotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter. *Comput. Networks* 67 (2014), 141–153.

A APPENDIX

A.1 Additional Plots and Tables

In this section, we present additional plots of Fig.s 6, 8 and 9, as well as Table 2.

A.2 Stability of pcv Solutions

Since pcv is a randomized algorithm and requires the number of clusters k as input, we also study how stable its biclustering solutions are over different runs and across different choices of k.

More concretely, given an unweighted bipartite graph G, we created five initial biclusterings of G for $k = 7$ (as in our experiments above). Then we vary $k = 5, \ldots, 10$ and each time we create another five biclusterings of G. For each new biclustering and and each initial biclustering, we compute the normalized mutual information (NMI) of their induced bicliques (see Eq. (1)); we report the averages and standard deviations of these NMIs. Note that we use the induced bicliques for computing the NMI because they relate to the edges "covered" by the biclustering and they are independent of k and can handle overlapping clusters (as produced by pcv).

We report our results in Fig. 7. The NMI values we obtain are always above 75% and the standard deviations are low. Indeed, here we report some of the "worse" results; on many other datasets the NMIs are well above 90% and have extremely small standard deviations.

A.3 Graph Similarity over Time

To analyze how similar two consecutive graphs G_t and G_{t+1} are, we compute the percentages of edges that appear, disappear, and do not change between E_t and E_{t+1}. We present additional plots for the Web cluster in Fig.s 10 and 11, and the Database cluster in Fig.s 12 and 13.

A.4 Graph Statistics

In this section we report the size of the clusters in Table 3.

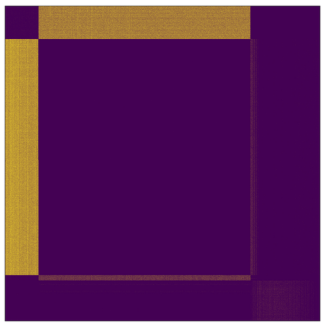


Figure 6: Visualization of the biclustering of server–server communication over a 15 minute interval from the Web cluster for $p = 0\%$. We considered the 10,000 servers that sent and the 10,000 server that received the most data. pcv detects the two blocks on the left and at the top as biclusters, but does not include the slightly more dense areas towards the bottom and towards the right.

Table 2: Our findings for the Hadoop cluster using GraphScope for rack-rack traces (S - segments, s - #clusters for sources, d - #clusters for destinations).

	15' S, s, d	5' S, s, d	1' S, s, d
p=0%	3, 5, 6	14, 9, 10	93, 9, 9
p=30%	7, 9, 8	21, 8, 10	81, 8, 10
p=50%	6, 9, 8	17, 8, 10	73, 8, 9
p=70%	7, 9, 9	14, 7, 9	68, 8, 8

(a) 5 min, $p = 70\%$, Web cluster rack–rack com.

(b) 1 min, $p = 0\%$, Database cluster rack–rack com.

Figure 7: Stability of the pcv biclusterings for rack–rack communication of the Web cluster (left) and the Database cluster (right). We plot average NMI values against the parameter k; errorbars correspond to standard deviations.

(a) ordered plot (b) measures (c) similarity

Figure 8: Our results for the Hadoop cluster and rack–rack communication over 5-minute time steps and $p = 30\%$. Fig. 8(a) shows the graph after reordering according to the biclustering. Fig. 8(b) presents recall, precision and traffic inside the biclusters. Fig. 8(c) shows the similarity matrix.

Table 3: Densities of the thresholded graphs $G_t^\tau = (V_t, E_t^\tau)$ for rack–rack communication. Here, the densities are the averages over all values $|E_t|/(|U| \cdot |V|)$, where we average over t. We present the results for different values for p and different time step lengths.

Clusters	Web cluster			Database			Hadoop		
t	**15'**	**5'**	**1'**	**15'**	**5'**	**1'**	**15'**	**5'**	**1'**
p=0%	0.83	0.74	0.48	0.47	0.46	0.4	0.98	0.96	0.7
p=30%	0.58	0.52	0.34	0.33	0.32	0.28	0.69	0.68	0.49
p=50%	0.42	0.37	0.24	0.23	0.22	0.2	0.49	0.48	0.35
p=70%	0.25	0.22	0.14	0.14	0.14	0.12	0.29	0.29	0.21

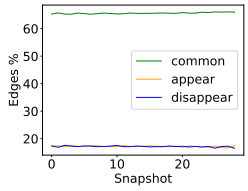
(a) ordered plot: Web cluster, rack–server, 5 min, $p = 0\%$

(b) ordered plot: Database cluster, rack–server, 15 min, $p = 0\%$

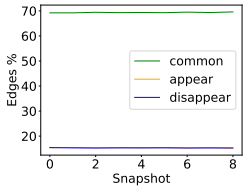
(c) ordered plot: Database cluster, server–rack, 15 min, $p = 0\%$

(d) ordered plot: Hadoop cluster, rack–server, 15 min, $p = 0.3\%$

Figure 9: Visualizations of the communications between racks and servers. In all plots, the sending racks are in the rows and the columns correspond to receiving servers, except for Fig. 9(c) where the rows are receiving racks and the columns are sending servers. For Fig.s 9(b) and 9(c) we have removed some sparse parts on the right side of the plot for better readability.

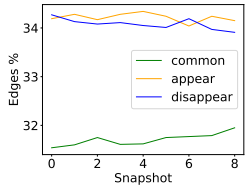


(a) measures: 5 min, $p = 30\%$, Web cluster rack–rack com.

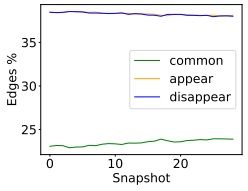


(b) measures: 15 min, $p = 30\%$, Web cluster rack–rack com.

Figure 10: Similarity of consecutive graph snapshots for the Web cluster, rack-rack communication. Notice that the percentage of edges that appear, disappear, or are in common may be the same and two different lines may overlap.

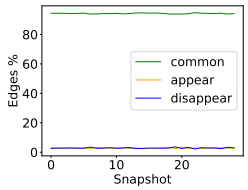


(a) measures: 15 min, $p = 70\%$, Web cluster rack–server com.

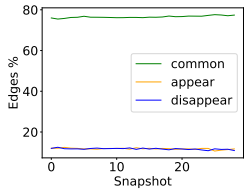


(b) measures: 5 min, $p = 70\%$, Web cluster server–rack com.

Figure 11: Similarity of consecutive graph snapshots for the Web cluster. Notice that the percentage of edges that appear, disappear, or are in common may be the same and two different lines may overlap.

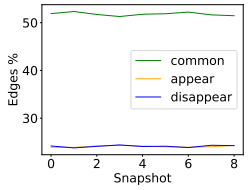


(a) measures: 5 min, $p = 30\%$, Database cluster rack–rack com.

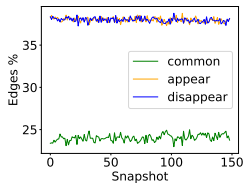


(b) measures: 5 min, $p = 30\%$, Database cluster rack–server com.

Figure 12: Similarity of consecutive graph snapshots for the Database cluster. Notice that the percentage of edges that appear, disappear, or are in common may be the same and two different lines may overlap.



(a) measures: 15 min, $p = 70\%$, Database cluster server–rack com.



(b) measures: 1 min, $p = 70\%$, Database cluster server–rack com.

Figure 13: Similarity of consecutive graph snapshots for the Database cluster, server–rack communication. Notice that the percentage of edges that appear, disappear, or are in common may be the same and two different lines may overlap.