

Clustreams: Data Plane Clustering

Roy Friedman

Computer Science Department
Technion

Or Goaz

Computer Science Department
Technion

Ori Rottenstreich

C. Science and E. Engineering
Technion

ABSTRACT

Clustering is a basic machine learning task. In this task, a stream of input items needs to be grouped into clusters, such that all items classified into the same cluster are closer to each other than to items classified to other clusters. Each cluster is centered around a *centroid* point, which may either be given as a parameter, or must be learned during the process in the case of unsupervised online learning.

This work studies the ability to perform clustering, e.g., for classifying network traffic, in programmable switches. Conducting such classification by the switches through which the traffic flows is potentially the most efficient approach. To that end, we develop *Clustreams*, a novel in-network clustering system designed to handle clustering in the data path. At the core of *Clustreams* is a novel clustering algorithm that relies heavily on TCAM (Ternary Content Addressable Memory) match-action capabilities. This algorithm is realized for the Nvidia Spectrum-3 switch, and is limited to classification when the centroid points are known a-priori.

The work includes accuracy measurements for the algorithms, as well as run-time performance measurements and analysis of the clustering algorithm on a Spectrum-3 switch. As shown in the measurements, *Clustreams* obtains very high accuracy without any noticeable run-time impact on the switch's performance.

CCS CONCEPTS

• **Networks** → **Packet classification.**

KEYWORDS

Network Algorithms, Clustering, Programmable Networks

1 INTRODUCTION

Machine Learning (ML) models are algorithms capable of producing insights from a given dataset. Clustering [16, 22], a prominent ML task, refers to grouping objects into subsets according to their similarity. Each group has a *centroid* point, which represents the center of the group. For instance, clustering can be used to classify types of network traffic, for detecting attacks on the network, to perform user segmentation for ads targeting [27], etc. The search for in-network clustering is motivated by the fact that network packets need to be clustered for enhanced network monitoring and management [17, 19, 20]. Nevertheless, since network devices

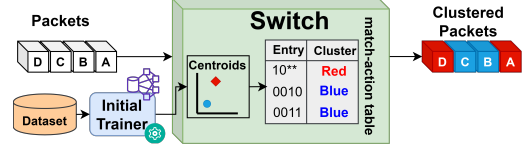


Figure 1: *Clustreams* trains the training set in an external server and uploads the centroids to the switch. Switch clusters each packet based on the match-action table.

should work at a high rate, clustering should leave a small footprint on the network's performance.

Programmable switches have become popular in the last decade, thanks to the emergence of the P4 language [5], which provides a specification for implementing a user-defined function inside the switch. However, the spec includes a bounded number of allowed instructions to reduce the impact on the switch performance.

Programmable switches are flexible and can be used as distributed caches [11], smart forwarding systems [24], for heavy hitters detection [25], etc. Self-driving networks [9, 13, 14] employ machine learning models to configure and maintain themselves autonomously. Another example is *pForest* [7], a recent effort to embed decision trees and random forest models using P4.

Our Contributions: In this work, we focus on clustering network traffic within programmable switches. Clustering network packets can be performed based on the packet header. The clustering attributes may include certain fields from the packets' headers or any custom header field that is predefined. To this end, we introduce *Clustreams*, a novel in-network clustering system that is capable of performing clustering of traffic within programmable switches. In fact, *Clustreams* supports an elemental clustering algorithm. Let us note that in *Clustreams*, one assumes that the centers of the clusters, also known as *centroids*, are given to the algorithm. We have realized this algorithm on the Nvidia Spectrum-3 switch.

For the purpose of clustering, define *workspace* to be a bounded area in the Euclidean space of dimension n where each axis belongs to the range $[0, k]$ for some k and the value of each attribute falls inside this range. We denote *object* as a subset of attributes in the packet's header intended for clustering. Our algorithm includes a combination of a quadtree [23], a data structure for encoding a two-dimensional workspace by recursively subdividing it into four quadrants, and a ternary match-action table (TCAM). *Clustreams* maps the workspace into designated areas. Each area is assigned to a specific cluster or remains unassigned, depending on whether the proximity of the area to a specific cluster is minimal or not. The appealing advantage of TCAM is the ability to search in parallel in all possible areas to find the one that contains the candidate object.

Clustreams may not cluster all objects since the defined areas can be related to two or more clusters. This is influenced by a parameter

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SOSR '21, September 20–21, 2021, Virtual Event, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-9084-2/21/09...\$15.00

<https://doi.org/10.1145/3482898.3483356>

controlling the maximum zoom level that the algorithm is allowed to explore. With finer grained zoom levels, the clustering rate inside the switch is increased at the expense of more TCAM entries. An object that is not contained in any predefined area can be routed to an auxiliary control plane clustering server for standard clustering on modern hardware with the capability to run any high-level language program. Our algorithm guarantees complete accuracy when using the quadtree solution and a high clustering rate.

We have measured the run-time performance of *Clustreams* on a Spectrum-3 switch to study its impact on the routing throughput of the switch. We also measured the throughput improvement and latency reduction compared to the standard solution in which packets are sent from the switch to a clustering server for classification there. In our measurements, the switch delivered nearly identical latency and throughput results with and without *Clustreams*. Further, compared to clustering by forwarding packets to a clustering server and then sending it back to the client, *Clustreams* was able to cluster packets about 14-24 times faster compared to using a clustering server alone.

In addition to our quadtree *Clustreams*, we tested a hybrid solution that involves a prior search on fewer “hot-zones”. Those “hot-zones” are mostly areas anticipated to contain a large portion of the candidates to be clustered, motivating checking them first before scanning the entire workspace with all the designated areas.

2 BACKGROUND

2.1 Clustering and K-Means

Clustering is a technique of grouping a collection of objects in such a way that objects that belong to the same group are more similar to each other than to those who belong to other groups. Clustering is a very popular unsupervised learning task and it has multiple implementations and applications. For instance, document categorization [4], crime locations identification [12], customer segmentation [27], etc. There are multiple clustering algorithms, and each one of them is chosen according to the nature of the data that needs to be clustered. We denote *centroid* of a cluster as the cluster center. *K*-Means [10, 21], a centroid based clustering algorithm, is a widely adopted algorithm with very simple logic.

2.2 Programmable Switches and P4

Programmable Switches are a new generation of switches that are capable of performing instructions in addition to ordinary packet forwarding. There are two main differences between traditional switches and programmable switches. First, the data plane functionality in a programmable switch is not fixed by design and it is configurable. The data plane has no built-in knowledge of existing network protocols and it is configured at the initialization step to perform the given program functionality. Second, the control plane communicates with the data plane through the same channels as in a fixed-function device. The compiler generates a dedicated API for the control plane and the data plane communication.

The prominent programming language for programmable switches is P4 [8]. P4 is a domain-specific language and it is also a target-specific language. Thus, the spec of the P4 language in every target can be different (decided by the switch manufacturer). P4 supports

conditions, match-action tables, stateful objects, and more. However, it does not support loops, recursion, and other complex operations to reduce the impact on the switch latency.

The match-action table is a fundamental data structure in P4. A match-action table contains lookup keys from packet fields or computed metadata and matched actions. When a match-action table is applied on a packet, a table lookup is performed. Then, the action of the selected key is executed.

Programmable switches can implement various switch architectures. A common architecture for describing P4 concepts is PISA [18] (Protocol Independent Switch Architecture), which is a single pipeline forwarding architecture. This protocol is based on the instruction pipelining architecture known for CPU designs [15].

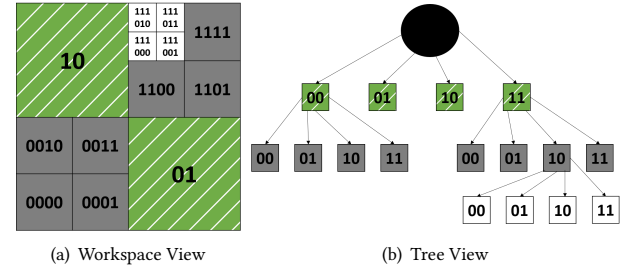


Figure 2: Quadtree data structure. Fig. 2(a) illustrates the split of a given workspace into quadrants and shows their encoding. Fig. 2(b) shows how this data structure is implemented as a tree.

2.3 Quadtree Data Structure

Quadtree [23, 26] is a tree data structure that encodes a 2 dimensional workspace. The entire workspace is divided into 4 squares while each square gets a specific 2 bits encoding. Each square can be further divided into 4 squares while the encoding of the children is a concatenation of the parent encoding and the 2 bits encoding of the square. This can be applied recursively. For instance, Figure 2 introduces a quadtree data structure with 3 levels. There are similar solutions for higher dimensional workspaces such as octree for 3 dimensions. A common use case of quadtree is geospatial search.

3 RELATED WORK

TCAM based KNN: A fast similarity search algorithm for the *K* nearest neighbors problem using TCAM was presented in [6]. Their algorithm is based on a novel extension to BRGC (Binary Reflected Gray Code) [1] for encoding a multi-dimensional workspace. BRGC is a binary encoding of integers in a contiguous range such that the codes of any two consecutive numbers differ by a single bit. It provides a method for encoding constant ranges. Therefore, the paper presents a new way to generate custom ranges by adding a few extra bits to the encoding value. The algorithm suggested in [6] detects the *K* nearest neighbors of an object *O* by building cubes surrounding a specific area surrounding *O*. The cubes are used to fetch objects that exist in the same cube as well as the object *O*. Those objects are candidates for the nearest neighbors of object *O*.

Each cube contains a certain number of neighbors, and the scanning of the cubes can be run in parallel so finding all the K neighbors executes quickly. Still, the algorithm has some limitations: The main drawback is the cube edge, which is limited to the number of bits needed to define the entire workspace. When the edge is limited to a specific size, the cube cannot cover large areas. Another drawback is the distance error. Usually, Euclidean distance is calculated using the ℓ_2 norm. In TCAM, queries are based on the maximum norm ℓ_∞ , which can produce an error in distance calculation.

Do Switches Dream of Machine Learning?: Multiple implementations of a decision tree, SVM, Naive Bayes, and K -Means on programmable switches are described in [28]. The paper demonstrates the mapping of those four trained machine learning algorithms to a match-action pipeline. Software and hardware-based prototypes of a framework that automatically maps trained models to a match-action pipeline are presented.

pForest: pForest [7] is a pioneering solution for implementing in-network decision trees. pForest is capable of classifying packets in the switch or run more complex operations such as random forest. pForest encodes the decision trees in P4 registers and uses internal logic to advance in the decision tree inside the switch.

4 OVERVIEW OF OUR CLUSTERING SOLUTIONS

Since many existing programmable switches still do not support stateful objects (e.g., Spectrum-3), we focus on stateless algorithms.

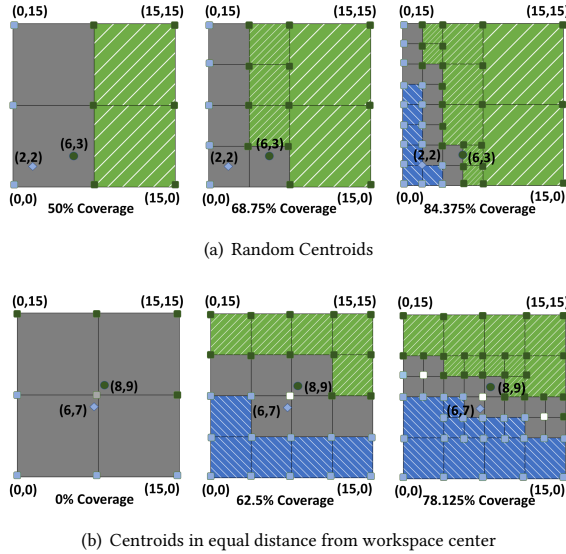


Figure 3: Different examples of the first three encoding steps. In each example we used 4 bits to store a dimension value, thus workspace size is $(2^4-1) \times (2^4-1) = 15 \times 15$. Both examples contain exactly two centroids.

4.1 Quadtree Clustreams

As mentioned above, a quadtree is a well-known method for geospatial search. In our case, we utilize quadtree to encode an area based on its affinity to one of the centroids.

The algorithm assigns a quad to a specific centroid if and only if the centroid is the closest centroid to all the quad's vertices. If not all vertices agree on the same centroid, the quad is further split into 4 smaller quads. The algorithm tries to assign the new quads to centroids in the same way, and the split process continues recursively until there are no more quads to split or the algorithm reaches the maximum number of allowed splits (a runtime parameter). Figure 3 shows two detailed examples of this process.

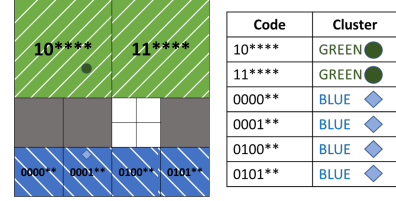


Figure 4: Quadtree Clustreams while scanning depth 3

At the end of the build process, a quad is either assigned (the colored zones) to a specific centroid or is left undecided (the gray zones). This assignment process guarantees complete accuracy as only quads that have a single closest centroid for all edges are added. This encoding method generates only LPM (Longest Prefix Match) entries. LPM entries contain prefix keys. In the lookup process, a value matches the entry if it starts with the key value. Entries are stored in the order of their depth in ascending order. An example of the workspace segmentation and the related keys are described in Figure 4. In this example, there are two centroids, blue and green. On depth 1 the upper quads were assigned to the green centroid because it is closest to all their corners. The rest of the quads remains gray since the algorithm cannot assign them to a specific centroid and they need to be split. In the second depth scan, the 4 lower quads were assigned to the blue centroid for the same reason, and the rest remains gray for the next level. On depth 3, each unassigned gray quad is split into another 4 quads for further assignments. In some P4 targets, e.g., Tofino [3], the entries of the match-action table can be changed in runtime. We can update the entries occasionally based on newly collected data without interfering with the production flow.

Since values are stored in the packet header with a fixed number of bits, we define the size of every dimension as $[0, 2^b - 1]$ where b is the number of bits that are required to store each dimension value. For safety, when a point is encoded into a quadtree representation before it is sent to the switch, if the point falls exactly on one of the quadtree separator lines, the encoder uses the quadrant with the smaller bit value. For example, if the point $(7.5, 3)$ is encoded for the workspace that is presented in Figure 3(b), the expected encoding value of the first depth is the lower-left quad, which is 00.

Quadtree utilizes the switch functionality to reduce the latency of a clustering request. When a clustering request is sent to the clustering server, the packet potentially can be clustered on the switch, and immediately the cluster label is sent back to the client.

when quadtree is not able to cluster the packet inside the switch, the packet continues to the clustering server and is sent back after clustering. As the number of clustered packets in the switch increases, the clustering server requires fewer resources and the average response time keeps decreasing.

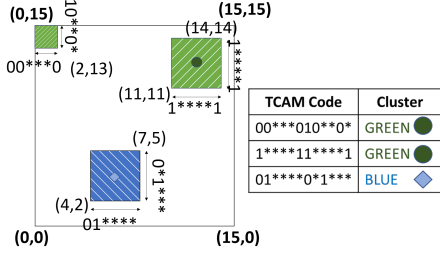


Figure 5: hybrid *Clustreams*. “Hot-zones” are pre-defined around the centroids and/or in promoted areas.

4.2 Hybrid *Clustreams*

Some workloads exhibit “hot-zones”, which contain a large portion of the objects that should be clustered. Those “hot-zones” can be chosen using heuristics, e.g., in a map, main roads will probably have more points than rural roads, etc. The clustering algorithm can utilize such “hot-zones” by focusing on them and disregarding the remaining areas at the beginning. Such a hybrid solution can benefit from both worlds. That is, we can divide the search process into two stages in the data plane.

The first stage is used to track objects inside “hot-zones”, and objects that exist in one of the “hot-zones” are clustered and skip the second step. The benefit is that “hot-zones” are likely to translate into a small number of areas, which means the number of entries in the match action table is small. Even though the search for entry match in the match action table can be run in parallel, running fewer operations in parallel can reduce the power usage, which is usually very high to enable the device to run this search in parallel.

Figure 5 explains the “hot-zones” definition. As shown, each centroid is rounded by a quad to define. Each quad is encoded with the cube based approach, while as one of the TCAM constraints, range length must not be bigger than the number of bits used. In the example, we use 6 bits to encode each range, while the ranges are in size of 4 or 2. The 6 bits encoding includes 4 bits for the standard Gray code and 2 extra bits for custom ranges. The range of each axis is encoded, and the final TCAM code represents a concatenation of all the encodings. Unlike the quadtree solution, the location of the “hot-zones” is dynamic. Therefore, we can exploit it to reduce the number of entries in the first stage.

The second stage is the same algorithm we described for the quadtree *Clustreams*. Thus, all the objects outside the “hot-zones” can be clustered on the second stage. Since PISA enables dividing the code into atomic units and run them in the pipelining model, two stages can be run in parallel for different packets.

S	The dimension size of all dimensions for centroids and points
D	Number of dimensions in workspace
K	Number of centroids
Ω	Depth backoff - number of depth levels to stop before max depth

Table 1: Notations: K , S and D - the maximal accepted values.

5 IMPLEMENTATION

Clustreams is realized to run on a programmable switch with TCAM capabilities. Notations are defined in Table 1.

Clustreams compilation and interface are simple. To produce the P4 code, a Python script generator needs to be invoked along with the *Clustreams* configuration parameters. After the code is created, it is deployed to a P4 target switch through the control plane.

Algorithm 1 *Clustreams* Quadtree

```

1: procedure ENCODE( $dim\_size, d, depth\_backoff, centers$ )
2:    $entries \leftarrow \{\}$ 
3:    $prefixes \leftarrow \{\}$  ▷ Prefixes to skip
4:    $max\_depth \leftarrow dim\_size$ 
5:    $depth \leftarrow dim\_size - depth\_backoff$ 
6:   while  $depth > 0$  do
7:      $codes \leftarrow generate\_all\_codes(d, depth, prefixes)$ 
8:     for  $c \in codes$  do
9:        $centroid\_id \leftarrow find\_closest\_centroid(x, C)$  ▷ returns
       NULL in case there are more than 1 centroid
10:      if  $centroid\_id \neq null$  then
11:         $entries \leftarrow entries \cup (code, centroid\_id)$ 
12:         $prefixes \leftarrow prefixes \cup code$ 

```

5.1 Quadtree Solution

The quadtree solution includes a training phase and a clustering phase. First, *Clustreams* trains a given dataset and computes the centroids. Then, the algorithm splits the workspace, based on the number of dimensions D , into quads/cubes/etc. and encodes each of them according to the quadtree data structure based on the centroids as described in Algorithm 1. Algorithm 1 generates all the quadtree codes for every zoom level. If a prefix of a quadrant exists in the *prefixes_to_skip* list, the quadrant’s code is not calculated. Then, for each of the codes, the algorithm checks if there is a closest single centroid for all the vertices of the quadrant. Each quadrant is either assigned to a centroid or is split again. Those assignments are converted into a match action table format with corresponding LPM keys. Recall Figure 4, an example of quad assignments and their relevant LPM keys. Entries of these keys are imported and injected into the P4 code that handles clustering.

Figure 6 shows a slim example of the generated P4 code. The “const entries” block contains all the encoding and related cluster ids. To enable our algorithm to cope with varying numbers of dimensions and dimension sizes, we add the following:

Depth Backoff: In some cases, the split process can continue until a quad reaches an extremely small edge size. Each splitting of a quad into four smaller identical quads increases the number of entries by 3. Hence, a large number of splits in high dimensional and large dimension sizes may result in tens or hundreds of thousands of

```

control ingress(header hdr) {
  action ADD_CLUSTER(bit<4> cluster_id){
    hdr.p4clustering.label = cluster_id;
    send_back();
  }
}

table clustreams {
  key = {hdr.p4clustering.code: ternary;}
  const entries = {
    10*****: ADD_CLUSTER(1);
    11*****: ADD_CLUSTER(1);
    0000****: ADD_CLUSTER(2);
    0001****: ADD_CLUSTER(2);
    0100****: ADD_CLUSTER(2);
    0101****: ADD_CLUSTER(2);
    *****: MOVE_FORWARD();
  }
}

header p4clustering_h {
  bit<32> id;
  bit<8> code;
  bit<32> object;
  bit<4> label;
}

```

Figure 6: Clustreams P4 code example. Red area contains the header definition. Green area is the key and lookup type.

entries. For example, in case the dimension size is 2^8 and there are 4 dimensions, the maximum number of entries reaches $S^D = (2^8)^4 = 4, 294, 967, 296$ (depending on K and the centroids locations), which is an infeasible number of entries to scan.

An alternate solution is to limit the depth our algorithm is willing to reach. We denote M the maximum depth, i.e., the depth at which the size of sides for all quads is 1 (or any other defined size). Instead of reaching maximum depth M , we add a new parameter named Ω . Ω defines how many depth levels the algorithm should stop before the maximum depth. That is, the new maximum level is defined by $M - \Omega$. For example, a 64×64 grid can be split up to 6 times before it reaches the minimal size of 1. Therefore, if $\Omega = 2$, the new maximal number of splits becomes 4.

Of course, we expect to see a reduction in the number of classified entries. But, on the other hand, the number of entries can be significantly reduced on each level we do not scan. Furthermore, using the depth backoff parameter reduces the entries size, yielding faster searches and reducing the training time.

Split Limit: Sometimes even the depth backoff parameter cannot prevent a generation of huge number of entries. Split limit is a hard limit on the number of entries for the entire algorithm or for a specific level. In case this number is reached, the algorithm stops.

5.2 Hybrid Solution

The hybrid solution involves two phases - the cube based technique of [6] and the quadtree based solutions. The main P4 code is constant with 2 stages, each of which contains a single match action table with one ternary key. If the cluster is found in the first stage, the second step is skipped. Similar to the initial step in the quadtree algorithm, as described in Section 5.1, a Python module trains and sets the centroids. Then, the centroids are covered by the maximal quad / cube size that does not overlap with any other centroid's shape. Predefined areas can be covered too by the same rule. "Hot-zones" are encoded to ternary entries in the first match action table of the P4 code and injected as in the constant entries section. The second step is the exact implementation from Section 5.1, which is invoked in case the first step does not detect the cluster.

6 EVALUATION

Our evaluation process includes running the clustering P4 program on Spectrum-3 switch. Clustreams generation code and evaluation methods [2] are written in Python.

6.1 Metrics

Switch Clustering Rate: Since our algorithm does not guarantee complete clustering on the switch because of the "gray zones" (Section 4.1), we measure the portion of objects in the dataset that was clustered inside the switch by the quadtree-based approach.

Error Rate: As mentioned, the quadtree based solution guarantees complete accuracy. Thus, this measure is only relevant to the hybrid solution, which may sometimes lead to a clustering error because of the cube based algorithm that reflects distances in infinity norm as described in Section 3. In this metric, we measure how many objects in the dataset were clustered to the wrong cluster.

Clustering Time: This metric includes the total time of clustering a stream of packets with and without *Clustreams*. This metric measures the end to end time of all packets since the first packet was sent from the client and until the last clustered packet was received back to the client. Further, we would like to measure if the throughput or the latency of a packet are affected by *Clustreams*.

6.2 Datasets

Random Generated Traces: Varying synthetic traces that are created by a Python script that picks up K centroids within a workspace of D dimensions with dimension size S . Our K values vary between 2 to 5 centroids, D is between 2 to 5 and $S = \{2^4, 2^6, 2^8\}$.

We used the uniformly distributed traces mostly to measure the robustness and the cost of *Clustreams* on big and uniform distributed datasets. Each dataset was split into a 25% training set and 75% testing set. The training set was used to create the initial centroids and the testing set was used to measure accuracy.

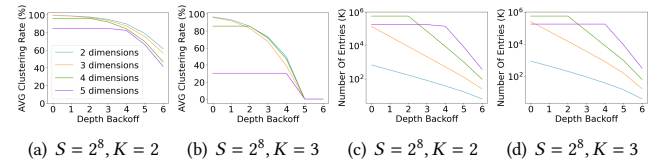


Figure 7: The average clustering rate and the relevant average number of entries based on the Ω value for different parameters. $K = \{2, 3\}$ and $S = \{2^8\}$. Each line refers to a different number of dimensions D .

6.3 Experimental results

6.3.1 Clustering: First, we would like to measure the switch clustering rate for different parameters. The first parameter we aim to check is Ω , since tuning the depth backoff can drastically affect the number of entries in the match action table. Figure 7 illustrates the switch clustering rate as a function of the Ω value. We limited the Ω value to $(S - 2)$ to get at least 2 levels of quadtree. As expected, the clustering rate has decreased when Ω increases since a high Ω reduces the granularity of the encoding process. For a low D value, we got better results since there are fewer vertices in every quadrant. As the number of vertices increases, an agreement of all vertices on the same cluster is less reasonable. To better understand the tradeoff between the clustering rate and the number of entries

the algorithm needs to create in the match action table, we measure the average number of entries for different Ω values.

In this test, we can see that the number of entries is drastically decreased when the Ω value gets bigger. This is expected since as we limit the zoom level of the algorithm, fewer areas are stored in the match action table. As the number of allowed entries is bounded by the manufacturer and the memory usage, our goal is to find the balance between high accuracy and memory usage on the switch.

In particular, when the depth backoff is 0 or 1, we span more entries in the 4 dimensions case that with 5 dimensions as shown in Figure 7. Yet, with these specific set of parameters, we also obtain a significantly better clustering rate in the 4 dimensions case than with 5 dimensions, as illustrated by Figure 7.

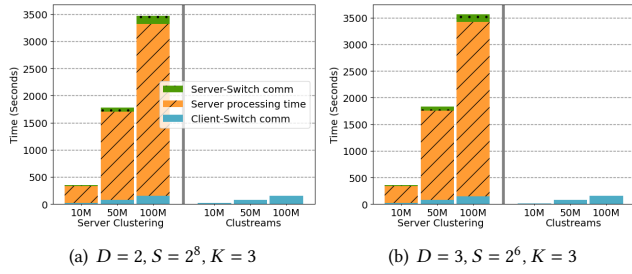


Figure 8: End to end clustering time between server clustering to *Clustreams*. Each bar title represents the number of sent packets from the client (10M / 50M / 100M packets).

Next, we would like to validate that quadtree *Clustreams* has a small footprint on the network latency and the switch throughput. To that end, we first measure the maximal throughput of the switch without *Clustreams*. We attached a 10GbE NIC (Network Interface Controller) to our server. Then we connected the server to the Spectrum-3 switch and configured the port as well. In our performance tests, we sent 60M packets with a size of 1KB. Without *Clustreams*, we were able to send and receive 1,156,173 pps (packets per second) (about 9691Mbps). When we turned on *Clustreams* using a P4 program with 10K ternary entries, we obtained a throughput of 1,158,610 pps (about 9713Mbps) were sent and received. Since performance results are almost the same, we deduce that *Clustreams* code in the switch has a hardly noticeable impact on the switch throughput and latency.

In Figure 8, we measure the end to end clustering time over a variety of workspaces and clusters. In this experiment, after we proved there is no footprint on the network latency and the switch throughput, we used a smaller packet size of 48 bytes only for this test. In the compiled P4 program there are about 1K entries at Fig. 8(a), and 11K entries at Fig. 8(b). The results exhibit 14-24 times improvement in the clustering time. Since *Clustreams* classifies the vast majority of the packets inside the switch, the processing time in the clustering server when using *Clustreams* is relatively insignificant. Further, the network latency in *Clustreams* is reduced by half, since the route of a packet includes only one hop in the network instead of two.

For the hybrid solution, we measure what is the error rate compared to the clustering rate. Figure 9 shows that the portion of

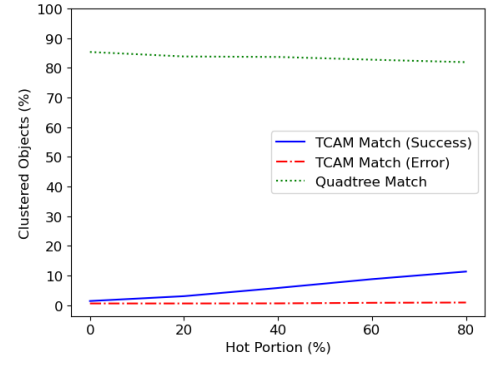


Figure 9: Rate of successful and unsuccessful clustered objects on stage 1 (cube) with the rate of successful clustered objects on 2 stage (quadtree). Results are averaged for various D , K , and S . The "hot-zones" are defined by the biggest possible non-overlapping shape around the centroids.

successful objects clustered at the first stage increases linearly when the hot-zones size increases relatively. The reason why we get less than 20% of objects clustered is overlapping surrounding "hot-zones", which cannot be clustered at the first stage and the ℓ_∞ norm that can result in some objects that the located in "hot-zone" areas to be considered as outside of them.

The portion of unsuccessful objects at the first stage is very small. It increases slightly when the hot portion size gets bigger, but is usually below 1%. The portion of successfully clustered objects at the second stage is usually above 80%, but we see an insignificant decrease when the hot portion gets larger since the first stage catches more objects. The error rate is not measured for the second stage since it was already proved to be immune to accuracy errors.

7 CONCLUSIONS

In this paper, we presented *Clustreams*, our novel in-network clustering system for programmable switches. We have shown that *Clustreams* provides accurate and fast clustering for network devices. *Clustreams* can run in-network clustering for incoming packets. Our quadtree method achieved maximal accuracy and good performance results. The hybrid solution also shows promising results, and it can be used to reduce power consumption of the switch in case it is required. Further, we presented how *Clustreams* can be implemented in P4 and be operational in the switch.

Our results indicate that running clustering process on the switch can drastically reduce latency and network traffic. These concepts are not limited to clustering only. In line with previous work on P4, a large number of obstacles can be overcome by using the ideas we discussed in the paper. Exploring how to implement additional ML models on programmable switches is left for future work.

8 ACKNOWLEDGMENT

This work was partially supported by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel National Cyber Directorate as well as by the Israel Council for Higher Education. It was also supported by the Taub family foundation.

REFERENCES

- [1] 1953. F. Gray. Pulse code communication. US Patent 2,632,058.
- [2] 2021. Clustreams generation and analysis code. <https://github.com/orgoaz/Clustreams>.
- [3] Barefoot. 2017. Barefoot Tofino. <https://www.barefootnetworks.com/technology/#tofino>.
- [4] Daniel Boley, Maria L. Gini, Robert Gross, Eui-Hong Han, Kyle Hastings, George Karypis, Vipin Kumar, Bamshad Mobasher, and Jerome Moore. 1999. Partitioning-based clustering for Web document categorization. *Decis. Support Syst.* 27, 3 (1999), 329–341.
- [5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: programming protocol-independent packet processors. *Comput. Commun. Rev.* 44, 3 (2014), 87–95.
- [6] Anat Bremner-Barr, Yotam Harchol, David Hay, and Yacov Hel-Or. 2015. Ultra-Fast Similarity Search Using Ternary Content Addressable Memory. In *International Workshop on Data Management*.
- [7] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. 2019. pForest: In-Network Inference with Random Forests. *CoRR* (2019).
- [8] The P4 Language Consortium. 2020. The P4 Language Consortium. The P4 Language Specification, Version 1.2.0 - Release Candidate, October 2020. <https://p4.org/p4-spec/docs/P4-16-v1.2.0.html>.
- [9] Nick Feamster and Jennifer Rexford. 2018. Why (and How) Networks Should Run Themselves. In *ACM Applied Networking Research Workshop (ANRW)*.
- [10] J. A. Hartigan and M. A. Wong. 1979. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28, 1 (1979), 100–108.
- [11] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *ACM Symposium on Operating Systems*.
- [12] Anant Joshi, A. Sai Sabitha, and Tanupriya Choudhury. 2017. Crime Analysis Using K-Means Clustering. In *International Conference on Computational Intelligence and Networks (CINE)*.
- [13] Patrick Kalmbach, Johannes Zerwas, Péter Babarczi, Andreas Blenk, Wolfgang Kellerer, and Stefan Schmid. 2018. Empowering Self-Driving Networks. In *ACM Workshop on Self-Driving Networks, SelfDN@SIGCOMM*.
- [14] Diego Kreutz, Fernando M. V. Ramos, Paulo Jorge Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2015. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* 103, 1 (2015), 14–76.
- [15] Jerome M. Kurtzberg and Raymond D. Villani. 1973. A Balanced Pipelining Approach to Multiprocessing on an Instruction Stream Level. *IEEE Trans. Computers* 22, 2 (1973), 143–148.
- [16] T. Soni Madhulatha. 2012. An Overview on Clustering Methods. *CoRR* (2012).
- [17] Anthony McGregor, Mark A. Hall, Perry Lorier, and James Brunskill. 2004. Flow Clustering Using Machine Learning Techniques. In *International Workshop on Passive and Active Network Measurement (PAM)*.
- [18] N. McKeown. 2015. PISA: Protocol Independent Switch Architecture. In *P4 Workshop*.
- [19] Andrew W. Moore and Denis Zuev. 2005. Internet traffic classification using bayesian analysis techniques. In *ACM International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*.
- [20] Thuy T. T. Nguyen and Grenville J. Armitage. 2008. A survey of techniques for internet traffic classification using machine learning. *IEEE Commun. Surv. Tutorials* 10, 1–4 (2008), 56–76.
- [21] Sasikumar Periyasamy, Sibaram Khara, and Shankar Thangavelu. 2016. Balanced Cluster Head Selection Based on Modified k -Means in a Distributed Wireless Sensor Network. *Int. J. Distributed Sens. Networks* 12 (2016), 5040475:1–5040475:11.
- [22] P. Rai and S. Singh. 2010. A survey of clustering techniques. *International Journal of Computer Applications* 7, 0975–8887 (2010), 1–5.
- [23] Hanan Samet. 1984. The Quadtree and Related Hierarchical Data Structures. *ACM Comput. Surv.* 16, 2 (1984), 187–260.
- [24] Anirudh Sivaraman, Changhoon Kim, Ramkumar Krishnamoorthy, Advait Dixit, and Mihai Budiu. 2015. DC.p4: Programming the forwarding plane of a data-center switch. In *ACM SIGCOMM*.
- [25] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *ACM SOSR*.
- [26] Jamel Tayeb, Özgür Ulusoy, and Ouri Wolfson. 1998. A Quadtree-Based Dynamic Attribute Indexing Method. *Comput. J.* 41, 3 (1998), 185–200.
- [27] Rong-Shiunn Wu and Po-Hsuan Chou. 2011. Customer segmentation of multiple category data in e-commerce using a soft-clustering approach. *Electron. Commer. Res. Appl.* 10, 3 (2011), 331–341.
- [28] Zhaoqi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning?: Toward In-Network Classification. In *ACM Workshop on Hot Topics in Networks (HotNets)*.