Discovering the Hidden Users of Scratch

by

Samarth Mohan

Submitted to the Department of Electrical Engineering and Computer Science on May 22, 2018, in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering

Abstract

Usage statistics ("telemetry data") have become an essential tool for understanding how complex systems are used and how to improve them. However, many of these systems are deployed in areas with limited internet connectivity which hampers the ability to collect telemetry data. In this thesis, we describe a telemetry data collection system built for the Scratch programming language to collect usage data regarding how Scratch is being used in areas with poor internet connections. We develop the system to allow users to opt-in to sharing their usage and project data with the Scratch research team at the MIT Media Lab. The data is stored locally on the user's machine until it is ready to be transmitted. Once network conditions are appropriate, the packets are transmitted to a server which verifies the contents of the packet and stores it in a data storage cluster. We aggregate the data and build a visualization dashboard to examine usage patterns, geolocation statistics, and project content for Scratch users all around the world.

Thesis Supervisor: Andrew Sliwinski
Title: Research Scientist

Thesis Supervisor: Mitchel Resnick
Title: LEGO Papert Professor of Learning Research
Acknowledgments

I would like to thank Andrew Sliwinski and Mitch Resnick for giving me the opportunity to work on Scratch. I am especially thankful to Andrew for his patience and support throughout these past two years.

I also want to thank members of the Scratch research and development teams for helping me with multiple rounds of testing and giving feedback on the project. Special thanks to Chris Willis-Ford for always making time to help me during the development process.

Furthermore, I would like to thank the people of the Lifelong Kindergarten Group for being such great peers and mentors. It was a pleasure being around such incredible people who taught me so much. I could not have asked for a better group to be a part of for my Masters thesis.

Thank you to the Theta Xi fraternity - Delta chapter and the MIT Cricket Club for all the great memories and experiences over the past six years. Also, thank you to the cast and crew of World of Wires for making this past semester so much fun. Trout, trout, trout!

Finally, I would like to thank my family for all the support during this journey. It would not have been possible to get this far without them.
Contents

List of Tables 6

List of Figures 7

Listings 9

1 Introduction 10

1.1 Research Objectives 11

2 Background 13

2.1 Anatomy of a Scratch Project 13
2.2 Learning Analytics 15
2.3 Offline System Functionality 19
2.4 Privacy and Trust 21
2.5 Data storage and Deployment 23

3 System Architecture 25

3.1 Client-side components 26

3.1.1 Event Packets 27
3.1.2 Offline Queue 28
3.1.3 Network Detector 29
3.1.4 Packet Dispatcher 30
3.1.5 File System 30
3.1.6 Consent Dialog 30
List of Tables

3.1 The Event packet schema content and descriptions . . . . . . . . . . 28
List of Figures

2-1  An example Scratch project consisting of several blocks and the Scratch Cat as a Sprite .................................................. 14
2-2  The Learning Analytics pipeline ........................................................... 16
2-3  Broadband costs around the world .......................................................... 20
3-1  The Client-side Architecture .................................................................. 26
3-2  The Consent dialog .................................................................................... 31
3-3  The Server-side Architecture .................................................................... 32
4-1  Number of app::open events that occurred between 01/30/2018 - 04/29/2018 43
4-2  Highest language frequencies in "Project Save" event packets ......................... 44
4-3  Graph displaying the frequencies of all languages used in "Project Save" event packets ................................................................. 44
4-4  Number of active users on the online portal from Asia (blue), Europe (orange), and the Americas (green) ................................................. 45
4-5  Number of "Save Project" events that occurred over a 7-day interval .............. 46
4-6  Number of projects that were uploaded to the online Scratch community over a 7-day interval ................................................................. 46
4-7  Number of unique clients that uploaded projects to the online community over a 7-day interval ................................................................. 47
4-8  Aggregates for projects created by users who set German ("de") as their language ................................................................. 47
Aggregates for projects created by users who set Brazilian Portuguese ("pt-br") as their language.
Listings

2.1 Abstract Syntax Tree Representation ........................................... 14
3.1 Valid Event Packet Schema ......................................................... 34
3.2 Invalid Schema, Missing the "name" field ..................................... 34
Chapter 1

Introduction

Scratch is a drag-and-drop programming language and online community developed by the Lifelong Kindergarten Group at the MIT Media Lab. Scratch users can create and share interactive stories, games, and animations [1]. Through Scratch, children between ages eight and sixteen (often referred to as “Scratchers”) learn important strategies for reasoning systematically, thinking creatively, and working collaboratively - essential skills for today’s world.

As Scratch has grown in popularity around the world, the programming interface (editor) has had to adapt to accommodate the diversity of languages and contexts in which it is used. As part of this effort, Scratch has created a Standalone Desktop Editor (SDE) that is especially useful for users that live in areas where internet connectivity is poor. The MIT Scratch Team is able to collect data about the ways that people use the online editor around the world, but currently there is no way to collect data from people who use the SDE. As an example, let’s consider statistics from April 2018. During this time, approximately 1M online Scratch accounts were registered and approximately 6M projects were shared. However, according to download metrics, the SDE was downloaded approximately 1M times. Unfortunately, for all these downloads, we have no information on how the SDE was being used. This is a massive gap in visibility and impact evaluation because we have no idea how millions of users are interacting with Scratch. Needless to say, this data is important.
for the developers and designers of Scratch, not only in understanding who is using the product and how they are using it, but also in helping to shape the development of Scratch towards learners around the world.

1.1 Research Objectives

The overall goal of this project will be to instrument the SDE such that it delivers summary statistics and usage information to the Lifelong Kindergarten Group at the MIT Media Lab. Using this system, we hope to gain a better understanding of the use of Scratch in areas with limited internet connectivity.

The collected data will consist of metrics that will help us gain insight on the types of projects that are being created in the SDE. With this, the research team will be made aware of projects that were previously hidden away unless they were shared with the online Scratch community. This will give us a more precise picture of the content being created through Scratch and who is creating it.

Given that Scratch is used globally, it is imperative that we develop a system infrastructure that runs smoothly on machines from all over the world with varying hardware and software capabilities. Additionally, the system needs to be scalable in order to accommodate the large amount of data that will be collected and stored.

The telemetry data collection system (TDCS) should be able to execute in the background without interfering with the user experience. The only interaction the Scratcher should have with the TDCS is to decide whether they want to share their data with the Scratch team or not.

Finally, Scratchers who are willing to share their data with us are doing so because they trust the Scratch team to keep their data safe and confidential. In order to maintain the trust between us and the Scratch community, the shared data should
be completely anonymous and stored securely.
Chapter 2

Background

In order to determine the telemetry data that we can collect about Scratch projects, it is important to understand what comprises a Scratch project and how information about the project’s content is stored on a user’s computer. Once we are aware of that, we can look at previous research in learning analytics, offline system design, privacy and trust best practices, data storage, and deployment to better understand how to build our telemetry data collection system.

2.1 Anatomy of a Scratch Project

Scratch is a blocks-based visual programming language that allows users to easily create interactive games, animations, and stories [1]. Each Scratch project consists of various blocks, sprites, costumes, and sounds. A sprite is any individual computer graphic whose behavior, appearance, and other characteristics can be manipulated through code. Blocks can be combined together to form a script. Blocks are "puzzle-piece shapes" that can be connected together to generate Scratch code [30]. There are several types of blocks that form the standard library of Scratch, which is similar to what is found in many other general purpose programming languages.

Scratch projects contain a hierarchical Abstract Syntax Tree (AST) representation that captures the structure and content of the project. The AST describes which
Figure 2-1: An example Scratch project consisting of several blocks and the Scratch Cat as a Sprite.

sprites were used and what scripts, variables, sounds, costumes, lists, etc were associated with each sprite. Below is a subset of the AST representation (Listing 2.1) for “Cat” i.e. the Scratch cat in the project seen above in Figure 2-1:

```json
...,
"objName": "Cat",
"scripts":
[
  {
    "x": 58,
    "y": 175,
    "opcode": "whenGreenFlag",
    "args": []
  },
  {
    "opcode": "doRepeat",
    "args": [10,
      {
        "opcode": "turnRight",
        "args": [15]
      },
      {
        "opcode": "forward",
        "args": []
      }
    ]
  }
]
```
2.2 Learning Analytics

An important asset while building any system is data as to how it’s users leverage the system. In a learning context, we refer to this data as “Learning Analytics”. Learning analytics help us refine the learning system based on the behaviors and tendencies of the users. Using ideas from previous research, we hope to design a system to collect useful data that can be used to understand how we can improve the experience for children around the world.

As defined in the journal article titled “Learning analytics: drivers, developments and challenges”, learning analytics refer to “the use of intelligent data, learner-produced data, and analysis models to discover information and social connections, and to predict and advise on learning” [2]. Learning analytics allow educators and designers of learning experiences to understand how students are learning and which learning methods are working better than others.
In 2012, the Department of Education released a brief [3] that outlined several questions regarding the learning process and how to improve it that can be answered using learning analytics. Some of these include:

(a) Understanding learning behavior and motivation based on usage patterns

(b) Analyzing trends in data to realize how user activity is evolving over time

(c) Effectiveness of current learning environment and other learning tools

(d) Future ideas for improvement to foster better learning

These are exactly the kinds of questions that we are hoping to answer with the data collected by the telemetry data collection system (TDCS). Therefore, even though the scope of this project does not involve building analysis models on the data that has been collected, in future studies, the data will be used for further research to enhance and refine the Scratch editor and online learning community. As a result, it is important to design our data collection scheme in a manner such that these studies can provide useful information about the experience of Scratchers and help answer the questions listed above.

Figure 2-2: The Learning Analytics pipeline

We can use previous work done by researchers at the MIT Media Lab in the field of learning analytics as a reference to determine what type of data is useful to collect. In the past, researchers have used learning analytics to assess computational thinking
[4] in Scratchers. In one study [4], they created a framework for assessing computational thinking that consisted of three different components: concepts, practices, and perspectives. Concepts consist of specific programming patterns (expressed by blocks) that users employ when designing their programs. The concepts described in the study included sequences, loops, parallelism, events, conditionals, operators, and data. These are ideas that are commonly used across Scratch projects and share similarities across a variety of programming languages. Practices focus on the user’s design strategies and thought processes. The goal is to understand the journey that the user went through from start to completion of a project. When interviewing children, the researchers discovered a common set of practices that Scratchers employed during development such as iterative and incremental design, testing and debugging, code reuse, and making the code modular. Additionally, they used a tool called Scrape that created visualizations to analyze the programming blocks used within a Scratch project. Based on the blocks used, they were able to determine which computational concepts were being applied by the Scratcher.

Other studies such as the one conducted by Fields, Giang, and Kafai [5] used analytics to examine patterns in Scratchers’ participation in collaborative spaces. In this study, the researchers collected data on participation behavior for a random sample of Scratchers who logged into Scratch in January 2012. Participation was defined by the following actions:

(a) Remixing a project

(b) Downloading a project

(c) Commenting on a project

(d) Marking a project as favorite

(e) Clicking "love-it" for a project

(f) Sending a friend request
By looking at the relations between each of these actions, the researchers were able to understand the participation tendencies of Scratchers. For example, they found that there is a strong correlation between commenting, favoriting, loving, or friending someone and downloading projects. In simpler terms, it was unlikely for a Scratcher to simply play around with a project without looking at the underlying code. Additionally, the study discovered that the trends in participation were independent of gender even though the Scratch community is skewed towards male participants. In other words, given a group of active Scratch users who were likely to participate in all of the above actions, the gender distribution was insignificant.

Apart from the Scratch-related research, learning analytics has proven useful in analyzing various learning systems and techniques to determine what works best for learners. Experimenting with various learning techniques and then using analytics to test their effectiveness can aid in figuring out optimal ways to teach students. The Department of Education [3] outlines work being done by companies such as Onsophic Inc. that use student behavior data to answer questions like does the learning system help improve learning or what kinds of interactions lead to learning. Additionally, the brief highlights a company that uses teachers’ data to “identify the pedagogical patterns of effective teachers, i.e., teachers whose students learn the most or are most engaged.” This can help teachers understand what behaviors they can adopt to become better educators.

It is clear that learning analytics can be used as a powerful tool for building better learning systems. However, quantitative analysis alone cannot be the solution. For example, in the study to measure computational thinking [4], the researchers used in-person interviews and design scenarios to collect information about how well a Scratcher understood a particular concept or practice. This is not possible to do simply by analyzing the project code because even if a Scratcher uses a block, it does not necessarily imply that they understand the underlying concept. Therefore, learning analytics have certain limitations which can be overcome by balancing and
supplementing such approaches with qualitative research methods. As a result, designers of learning systems must think deeply about their research objectives and balance the use and weight of learning analytics accordingly.

2.3 Offline System Functionality

The telemetry data collection system will need to function properly even if an internet connection is not available. Therefore, it is important to understand techniques used by software systems to handle network failure without disturbing the user experience.

A post titled "Designing Offline-First Web Apps" [6] examines various scenarios that can be encountered by users when an application goes offline and offers suggestions to make the application usable when this happens. The author suggests that apps should store local data pertaining to their most recent state so that if the server is unreachable, the app can still load content for the user, even if it cannot be modified.

Another similar issue that the author raises is treating offline mode as an error. When in a disconnected state, the app needs to filter views that contain no data and also make sure that error states are properly communicated. The author describes Instagram’s offline functionality as a potential approach to deal with this issue. When an internet connection is unavailable, Instagram treats it as a failure and ensures that any user actions performed during the offline state are completed once connectivity is reestablished. A more ideal approach, as is the case with Google Docs, is to allow the app to generate local data and then make sure that the data can be delivered at a later time when a connection is available, instead of preventing the user from interacting with the app. In essence, the goal of the app should be to function seamlessly so that the user does not feel a change in functionality even when it is in a disconnected mode.

A guide for Apple developers [7] also details some useful pointers for writing software for real-world networks. For example, the guide talks about how minimizing
bandwidth is important to reduce costs for users. Each network operation uses some bandwidth which comes with a cost and in some parts of the world, it is fairly expensive. For example, looking at Figure 2-3 [8], the average monthly cost of an internet connection in Burkina Faso is almost 200 times the cost in Iran. Therefore, the developer needs to be cognizant of these constraints and design the software accordingly so that users aren’t prevented from using the software.

![The Most And Least Expensive Countries For Broadband](image)

**Figure 2-3: Broadband costs around the world**

The guide advises to batch operations to make network usage more efficient. It also provides design ideas for dealing with network loss such as designing for:

(a) **Varying network connectivity**: Network connectivity can change at any time for various reasons such as moving outside of Wi-Fi range or into a dead spot with no cellular connectivity.

(b) **Varying network speed**: The network becomes busier and therefore each user
has access to less bandwidth

(c) **High Latency:** Slow speed of connection (e.g. 2G speed) can lead to a reduction in performance

Lastly, the guide also lists some ideas for testing such as testing for various network conditions like "reduced bandwidth, high latency, DNS delays, packet loss etc." It is important to ensure that the software is functional in all of these situations.

## 2.4 Privacy and Trust

Any system that asks users to share its data must ensure that the data is secure and protected. It is of the utmost importance that users trust Scratch to keep their data safe. This becomes especially important given events such as Cambridge Analytica getting access to private information of over 50 million Facebook users that transpired in April 2018 [9]. These breaches of user privacy and trust underscore the importance of data protection, informed consent, and anonymization.

Informed consent is a useful tool for establishing trust. As defined in the European Union General Data Protection Regulation (GDPR) [10, 25] that will go into effect in May 2018, consent means "any freely given, specific, informed and unambiguous indication of the data subject’s wishes by which he or she, by statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her." Additionally, the GDPR states that the user whose data is being collected must have the right to withdraw consent at any time and the process to withdraw consent should be as simple as it was to give consent in the first place. Furthermore, the GDPR also requires that when processing sensitive data, the consent must be explicit. In other words, the user must be made aware of the nature of the data that is being collected, the purpose of collecting the data, and any risks involved.

There is also the question of opting into sharing your data vs. opting out. Given the
rules on explicit consent, GDPR endorses the "opt-in" approach. Previous regulations assumed implicit consent and if a user wanted to stop sharing their data, they would have to "opt-out" on their own. The GDPR sets a good example for us to follow when implementing our method of getting the user’s consent.

Anonymization of data is another approach to protect data. Even if anonymous data is leaked, it is not possible to identify the user to which the data belongs and therefore, the user’s privacy is still protected. For example, legislation like the Health Insurance Portability and Accountability Act of 1996 (HIPAA), which outlines provisions to protect medical information in the United States, don’t apply to data that has been completely anonymized [11]. However, to be fully anonymous, the data must not contain any "direct and indirect subject identifiers" as stated in the MIT Committee on the Use of Humans as Experimental Subjects (COUHES) guidelines [11]. The GDPR also states that the rules on data protection do not apply to anonymous data [10].

A journal article titled "Ethical and privacy principles for learning analytics" [12] in the British Journal Of Educational Technology provides some useful guidelines to follow in order to establish trust with the user. Some tips include letting the user know exactly what type of data is being tracked. Additionally, making sure that the data being transferred is properly protected also influences the amount of data users are willing to share. Users will be more likely to disclose their data if they are confident that the data they are disclosing will not be stolen or misused by third parties.

The authors also outline the following principles of learning analytics systems: Transparency, Student control over data, Security and Right of Access, Assessment, and Accountability [12]. For each principle, the authors give advice on what the developers of the application should do in order to ensure that users feel comfortable using the application and—above all—trust the authors of the application. Regarding transparency, students, tutors, and teachers should be made aware of how the analytics
process is carried out and what type of information is collected. Additionally, they should be made aware of how long the data will be kept in the system and what processes are used to analyze the data and produce results.

Data Security and Right of Access is another key issue. In order to protect sensitive data from being accessed by unauthorized parties, a comprehensive right-of-access policy must be defined. This policy should identify the types of access operations allowed on the data, use a permissions model to decide which researchers and moderators have access to which parts of the system, and make the user aware of what information they are manipulating.

2.5 Data storage and Deployment

For storage of data that has been collected, we need to set up a data storage cluster that will consolidate the data being received from all users. In addition, the collection system must be deployed in a manner such that it can be efficiently reached from any location around the world.

Our choice of database system must satisfy the following requirements:

(a) Efficient and scalable read operations

(b) Efficient and scalable write operations

(c) A query syntax and a system that allows for complex data analysis

(d) Fault tolerance

There are several systems that satisfy these requirements. For brevity, we will consider only two of them: ElasticSearch (non-relational) and PostgreSQL (relational). Given that our data will consist of flat time-based log events, a non-relational model
like ElasticSearch fits our needs [13]. ElasticSearch is a versatile search and analytics engine that has several capabilities such as basic document search and efficient aggregation of large datasets. It has been used in practice by various companies to answer many types of data-related questions. For example, the Guardian, a British daily newspaper, uses ElasticSearch to determine metrics such as which headlines / articles are generating the most traffic, how many hits each article is getting, what article links should be displayed to the user etc [14]. This real-time insight into user behavior leads to improvement in user experience because the website displays content that the user wants to see, which in turn leads to an increase in the number of page views [14]. We hope to use the collected data in a similar way so that the user experience of Scratchers around the world can be improved.

To reach Scratchers all around the world, we can use a Content Delivery Network (CDN) to deploy our collection server. A CDN is "a large, geographically distributed network of specialized [edge] servers" [15] that allow content to be delivered to users more quickly by reducing transit time between the end user and the origin server. Instead of going directly from the end user to an origin server, a user request first travels to a nearby edge server and the edge server then delivers the request to the origin server. The bandwidth between the two servers is generally much faster than the end user’s bandwidth which allows the user request to be resolved more quickly.
Chapter 3

System Architecture

The telemetry data collection system (TDCS) consists of several client and server-side components that work together to collect and deliver data. The components are described in further detail below.
3.1 Client-side components

![Client side System Diagram]

Figure 3-1: The Client-side Architecture
Within the TDCS, the goal of the client-side code is to generate data packets and send them to the server in a secure and reliable fashion. In order to accomplish this goal, several subcomponents were added to the existing Standalone Desktop Editor (SDE). These include:

(a) A Packet (3.1.1) class to parse and generate data from a Scratch project’s abstract syntax tree (AST)

(b) An Offline Queue (3.1.2) to store the packets

(c) A Network Detector (3.1.3) to ensure packets are sent only if the network is available and the server is reachable

(d) A Packet Dispatcher (3.1.4) to handle sending packets to the server

(e) A File System (3.1.5) to persist and retrieve data to and from disk

Additionally, we want to keep our users informed about the data collection system and give them the choice as to whether they want to share their personal data. The Consent Dialog (3.1.6) addresses this by asking the user to opt into the system when they open the SDE for the first time. The role of each subcomponent and how it interacts with the overall system is described in further detail below.

### 3.1.1 Event Packets

Whenever the user performs certain actions within the SDE, the TDCS generates an Event Packet. These data are generated for the following events:

(a) The editor is opened (**app::open**)

(b) The editor is closed (**app::close**)

(c) A Scratch project is loaded into the editor (**project::load**)

(d) The current Scratch project is saved (**project::save**)

(e) A new Scratch project is created (**project::create**)

27
(f) The Scratch project is uploaded to the user’s online profile (project::upload)

The schema for each packet is as follows:

<table>
<thead>
<tr>
<th>Entry name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>Unique identifier for a given packet</td>
</tr>
<tr>
<td>timestamp</td>
<td>Integer</td>
<td>UNIX timestamp for when the packet was generated</td>
</tr>
<tr>
<td>clientID</td>
<td>String</td>
<td>The unique identifier for each new computer that downloads the Offline Editor</td>
</tr>
<tr>
<td>projectName</td>
<td>String</td>
<td>Name of the Scratch project</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Name of event for which packet was generated (e.g ‘app::open’, ‘project::save’)</td>
</tr>
<tr>
<td>userTimezone</td>
<td>Integer</td>
<td>The difference in minutes between UTC time and computer’s local time</td>
</tr>
<tr>
<td>language</td>
<td>String</td>
<td>The language of the editor as set by the user</td>
</tr>
<tr>
<td>scriptCount</td>
<td>Integer</td>
<td>The total number of scripts in the project</td>
</tr>
<tr>
<td>spriteCount</td>
<td>Integer</td>
<td>The total number of sprites in the project</td>
</tr>
<tr>
<td>variablesCount</td>
<td>Integer</td>
<td>The total number of variables used</td>
</tr>
<tr>
<td>blocksCount</td>
<td>Integer</td>
<td>The total number of blocks used</td>
</tr>
<tr>
<td>costumesCount</td>
<td>Integer</td>
<td>The total number of costumes used</td>
</tr>
<tr>
<td>listsCount</td>
<td>Integer</td>
<td>The total number of lists used</td>
</tr>
<tr>
<td>soundsCount</td>
<td>Integer</td>
<td>The total number of sounds used</td>
</tr>
</tbody>
</table>

Table 3.1: The Event packet schema content and descriptions

3.1.2 Offline Queue

Once a packet is generated, it is stored in the Offline Queue until it is removed by the Packet Dispatcher (3.1.4). The Offline Queue is represented by a "First In First Out" (FIFO) Array-based Queue [16] containing the last N entries and is stored on
disk using the File System API (3.1.5). Each entry object contains the packet itself and a **retry_limit**. This limit is the maximum number of times a packet can be re-added to the queue. A packet is re-added if it was dispatched from the client but was not successfully delivered to the server.

There are three possible actions provided by the Offline Queue object: **enqueue**, **dequeue**, and **flush**. If the queue is at capacity when a new packet is added, the oldest packet is removed from the queue to create space before adding a new one.

### 3.1.3 Network Detector

The Network Detector is responsible for tracking whether a network connection is available. This is particularly important since the SDE is extensively used by millions of users in areas where internet connectivity is limited, as highlighted in Chapter 1. Therefore, the TDCS must always be aware whether the user’s computer is connected to the internet. Anytime there is a change in the connectivity status, the Network Detector is immediately notified and updates its state. As a result, the TDCS can rely upon the Network Detector to know at all times whether the user’s computer is connected to the internet or not.

It is also possible that the server (3.2) cannot be reached for various reasons (e.g. power failure, upstream network provider failure, etc.). This situation is treated in the same way as a loss of connectivity. However, there is an important difference. Unlike the case where connectivity is lost, the Network Detector does not always know that the server cannot be connected to. To deal with this issue, the Network Detector pings the server at a regular interval of five minutes to check whether a connection can be successfully established. If it is unable to do so, the TDCS will not attempt to send any data even if the computer is connected to the internet.

The TDCS uses the Network Detector to determine whether the appropriate conditions are met in order for the Packet Dispatcher (3.1.4) to send packets to the
3.1.4 Packet Dispatcher

The Packet Dispatcher is responsible for sending event packets from the client to the server. There are two main functions of the Packet Dispatcher: start and stop. Every minute, assuming there are no network and server connectivity issues as determined by the Network Detector (3.1.3), start dequeues an event packet from the Offline Queue (3.1.2) and attempts to send it to the server. Note that the event packets are sent in a synchronous fashion to the server. Therefore, if a packet is in the process of being sent, the system waits for the response from the server before resending the same packet or sending the next packet. (see section 3.2.3 for details)

If an event packet is not successfully delivered, it is requeued and its retry_limit (3.1.2) is decremented by 1. Once the retry_limit (3.1.2) reaches 0, the event packet is no longer requeued.

Note that the Packet Dispatcher will NOT attempt to send any packets for a user that did not opt into the system (see section 3.1.6 for details).

3.1.5 File System

In order to store user specific data such as the clientID, data privacy setting (3.1.6), and the Offline Queue (3.1.2), the system persists data to disk. The File System class provides an API to access and update the data stored on disk.

3.1.6 Consent Dialog

Maintaining trust with our users by keeping their data private is of the utmost importance. Therefore, before sharing any of the user’s data, we ask for the user’s explicit consent through the Consent Dialog (Figure 3-3). The consent dialog is displayed to
Figure 3-2: The Consent dialog

The user the first time they open the SDE.

The user’s choice is stored on disk using the File System API (3.1.5). If the user chooses not to share their data, the system will still generate event packets (3.1.1) and store them in the Offline Queue (3.1.2). However it will never execute the Packet Dispatcher (3.1.4) and therefore, no packets will ever be delivered to the server.
3.2 Server-side components

Figure 3-3: The Server-side Architecture
Once the event packet is dispatched from the client, it reaches the server in the form of an HTTP request where it runs through a series of steps that culminate in its storage. The event packet first passes through a security middleware layer that ensures that the packet satisfies certain conditions. Afterwards, its contents are verified to make sure it follows the desired schema. Finally, it is added to the storage layer and the client is informed that the transaction was successful.

3.2.1 Security layer

Before the server can begin the process of verifying the packet contents, it must ensure certain security properties. Firstly, the size of the HTTP request body must not exceed 400 bytes. This is to prevent malicious clients from sending excessively large packets and using up server resources to execute a DoS attack [17]. Secondly, the server uses token bucket rate limiting [29] to throttle HTTP requests to make sure that a single IP address cannot clog server resources. Token bucket rate limiting defines an upper bound on how many packets ("tokens") can be sent by a given IP address ("bucket") for a given time interval. Therefore, even if a malicious user tries to cause a DoS attack by sending a large number of packets over a given time interval, the attack will be prevented because the server will not process any packets sent by the malicious user once the upper bound is reached.

Once both these conditions are satisfied, the HTTP request is logged by the server and the transaction can continue.

3.2.2 Validation / Integrity Layer

If the HTTP request passes all the security checks and is logged, the contents of the event packet must be verified. This entails verifying the schema of the event packet. If any violation of the packet schema occurs, the server responds with a BadRequest message and the packet does not proceed to the Storage layer (3.2.3). An example of a valid (Listing 3.1) vs. invalid schema (Listing 3.2) is shown below in which the
invalid schema is missing the name of the event packet.

```json
{
    "id": "230d95d1-56b76096-4a6e859817cefdc5",
    "timestamp": 1520363743553,
    "clientID": "0a276d4d-338fdc2c-445b527df2eca147",
    "projectName": "test.sb2",
    "name": "project::load",
    "userTimezone": -60,
    "language": "es",
    "metadata": null
}
```

**Listing 3.1: Valid Event Packet Schema**

```json
{
    "id": "230d95d1-56b76096-4a6e859817cefdc5",
    "timestamp": 1520363743553,
    "clientID": "0a276d4d-338fdc2c-445b527df2eca147",
    "projectName": "test.sb2",
    "userTimezone": -60,
    "language": "es",
    "metadata": null
}
```

**Listing 3.2: Invalid Schema. Missing the "name" field**

### 3.2.3 Storage Layer

Once the contents of the packet have been verified, the packet is added to the ElasticSearch storage cluster. If the packet is successfully added to the storage cluster, the server sends a **200 OK** message back to the client so that the client can send the next packet (3.1.4). However, if there was an error while trying to add the packet to
the cluster, the server responds with a 500 \textit{Internal Server Error} message and the client either queues the packet or sends the next packet (3.1.2, 3.1.4). This is the end of the TDCS pipeline.

### 3.3 Performance and Scalability

Given the large user base of the Standalone Desktop Editor, it is important that the TDCS is scalable. To do so, we need to account for varying network conditions (see section 3.5.1 and 3.5.2) around the world and make the server-side process efficient. Even though it may seem that the server has a non-negligible amount of work to do, based on the performance metrics collected as of April 2018, it takes 15 ms on average for the server-side process to complete. This demonstrates the server’s readiness to handle more traffic as more and more users start sharing their data.

### 3.4 Deployment

The server is created using Node.js, a popular Javascript runtime often used for server-side development. The server is deployed using Amazon Web Services's (AWS) Elastic Beanstalk service. Two separate environments were deployed for staging and production. It is possible for a single server instance to fail which would prevent any data from reaching the storage cluster. To accommodate server failures and ensure availability, multiple instances of the server were deployed. If any server fails, any HTTP requests to that server will be resolved by one of the other instances. The load-balancer will automatically adjust to account for the failure. We also use auto-scaling to automatically add and remove server instances based on changing levels of server load [18].

For storage of the data, we use an ElasticSearch cluster. The access policy of the cluster was modified such that only server instances could add data to the cluster. This ensures that the data being added to the cluster has gone through the neces-
sary security and verification steps (3.2.1, 3.2.2). The data is fully replicated across multiple ElasticSearch data nodes. Therefore, even if one data node fails, data is still accessible through the other nodes [31]. Note that any failed nodes will automatically be detected and replaced [31].

In order to increase availability and response times of the server around the world, we used a Content Delivery Network (CDN) [15, 20] provided by Fastly. This is especially important since our users are spread all around the world with varying levels of network reliability. With the CDN, we can easily reach more users and as a result, collect more data. Fastly also provides us with distributed denial of service attack (DDoS) [17] protection to make our system more resilient against malicious users with control over multiple compromised machines (e.g. botnets).

3.5 Design Decisions

3.5.1 Packet Retransmission

As stated in Section 3.1.2, each entry in the Offline Queue contains the packet and a \texttt{retry\_limit}. Given that network connectivity is a primary constraint on our system, we cannot assume that if the client sends a packet then it will definitely reach the server. Even though we use the Network Detector to ensure that all the necessary conditions are met before sending a packet (3.1.4), it is possible that there is network failure while the packet is in transit which can lead to the packet not being delivered to the server successfully. Therefore, in order to handle unseen network failures, we retransmit packets that were not successfully delivered until the \texttt{retry\_limit} is reached. The idea of using packet retransmission was inspired by several other systems such as TCP [19] that use retransmission as one of the basic ways to ensure reliable communication.

We chose to include the \texttt{retry\_limit} so that if one packet cannot be delivered, that
does not prevent other packets from being transmitted. While this results in loss of data, we discuss in Section 3.5.4 why this does not have a major negative impact on the applicability of collected data for our research.

### 3.5.2 Packet Schema

Restricting bandwidth usage is important to us because our users may have limited bandwidth that has to be shared by several processes on the user’s computer (e.g., web browsers). Additionally, internet connections can be quite expensive in certain parts of the world [8]. Therefore, it is imperative that the telemetry data collection system only increase the bandwidth usage minimally. As a result, we wanted our packets to be as small as possible. However, we still need to collect meaningful data in order to gain insights about our users and the work they are doing on Scratch. Thus, we have to handle the tradeoff between minimizing packet size to reduce the bandwidth usage and ensuring the data being collected can be used in future studies to understand how Scratch is used around the world.

We believe the current schema handles this tradeoff well. By using metrics like user timezone and language, we can gain insight about the location of our users without personally identifying them. Various counts for blocks, sprites, variables, and sounds can give us information about the projects that are being generated without exceeding 400 bytes per payload (3.2.1). Therefore, we are able to satisfy our bandwidth constraint while at the same time collecting meaningful data that allows us to make useful observations about the users of the Standalone Desktop Editor.

### 3.5.3 Choosing ElasticSearch as the data store

While thinking about what service to use for data storage, we wanted something that was simple to configure and allowed easy updates to the schema because we anticipated that our schema would be updated throughout the course of the project. Additionally, we wanted a service that provided us with efficient read and query op-
We considered several options such as ElasticSearch, PostgreSQL, and Amazon Redshift. Ultimately, we decided to go with ElasticSearch. ElasticSearch provided us with several benefits. Firstly, at the beginning of this research, we were not sure what the packet schema would look like at the end. As a result, we needed a service that made it as easy as possible to update the schema without losing any data and we found that ElasticSearch provided this sort of flexibility.

ElasticSearch is primarily designed for efficient information retrieval and serves our research needs well. Once a packet is pushed to data storage, it is never updated. However, we want to access its contents efficiently to perform data analysis and build visualizations. In other words, our system needs fast read performance and can compromise on write performance since we only write a packet once. ElasticSearch uses the process of denormalization [21] which improves read performance by maintaining redundant copies of the data in every document, thereby eliminating any use of expensive join queries. This does make keeping data consistent across documents difficult. But as stated previously, we only write once per packet when adding the packet to the ElasticSearch cluster. Therefore, we cannot generate inconsistencies in the data and do not have to worry about the effect on performance. ElasticSearch also provides built-in support for sharding to accommodate increasing demand which helps with scalability [22]. Shards are independent copies of the stored collection of documents known as the index. Sharding helps improve performance because operations can be distributed and parallelized across different shards which leads to more efficient usage of available resources.

One added benefit of using ElasticSearch is that we are also able to use Kibana, an open source data visualization plugin for ElasticSearch. By combining the ease of data aggregation provided by ElasticSearch with Kibana, we were able to quickly generate a data visualizer to generate various charts and metrics to get basic usage
statistics without writing any additional code. (Section 4)

3.5.4 Packet Loss

Despite having the Network Detector, there is no guarantee that a packet will always be successfully delivered to the server even after multiple retransmissions (3.5.1). As a result, packet loss can occur. However we do restrict the number of times this occurs by using retransmissions and the Network Detector. Additionally, even if packet loss does occur, it does not greatly impact our system because packets are frequently generated by the user. For example, while making incremental changes to a project, a user will produce multiple save packets and it is unlikely that none of them will reach the server. Therefore, we can still get a basic idea of what the project will look like despite losing some packets.

3.5.5 Privacy and Trust

As described in Section 3.1.6, to maintain our users’ trust, we must keep their data private. The user must explicitly choose to share their data with the Scratch team. The Consent Dialog informs the user about the purpose of the telemetry data collection system and the user can decide whether they want to share their data or not.

We chose to make the system opt-in instead of opt-out [23]. We wanted our users to be fully aware that they were choosing to share their data with us. With an opt-out flow, the user could be sharing their data without their knowledge because the system collects the user’s data by default without explicit notification. In such a system, the onus is on the user to figure out themselves that their data is being shared. In other words, the underlying assumption here is that the user will seek out their data-sharing setting and opt-out of the system. The danger with this is that this process requires awareness and proactiveness from the user, which cannot be expected. In fact, given that the telemetry data collection system runs in the background without any interference with the UI, most users wouldn’t realize that their data is being shared if we...
were to take such an approach. This is problematic because the user is not informed and has not consented to sharing their data [24, 25]. However, with an opt-in flow, we ensure that the user has been informed of the telemetry data collection system and they are comfortable with sharing their data. The European Union General Data Protection Regulation (GDPR) [10, 25] endorses "opt-in" over "opt-out". The GDPR requires that users explicitly opt-in to having their information collected. While some users may choose not to opt-in, we feel that giving the user the power over their data is more important even if that means we collect less data.

We also choose to make our data anonymous so that if any data is leaked or any malicious user is able to get access to the data, there is no way to determine which user generated the packet. The data is sent over a secure connection that uses TLS [26] to encrypt the data end-to-end.

There is one piece of information, the client ID (3.1.1), that can potentially identify which packet belongs to which user. The client ID is a randomly generated universally unique identifier (UUID). When a user uploads their project to the online Scratch community, one can find all the packets with the same project name and determine at least one client ID for that user. However, this is not a violation of user privacy because in order to upload to the online Scratch community, the user has to make a Scratch account which involves agreeing to the Scratch website’s privacy policy and terms of service. Additionally, only members of the approved research group have access to the data.

Finally, even if a user chooses not to share their data, event packets are still generated and stored in the Offline Queue (3.1.2). However, when the Packet Dispatcher (3.1.4) runs, it notices that the user has chosen not to share their data and therefore no packets are sent. Additionally, the Offline Queue is flushed. Therefore, packets are only temporarily stored on the user’s disk and are never dispatched to the server.
Chapter 4

Evaluation

As stated in the Research Objectives (1.1), the primary goal of the telemetry data collection system (TDCS) is to deliver summary statistics and usage information to the Scratch research team at the MIT Media Lab. Data collected by the TDCS can aid in research in the following areas:

1. Overall user engagement
2. Language / geolocation distribution
3. Usage patterns

Each of these categories contain different metrics that provide insights into how often Scratch is used, where it is being used, and what types of projects are being created.

Note that the data is only collected from users who choose to opt into the TDCS. This could introduce bias in the data because users in areas with expensive internet connections may be less likely to opt into the system. Additionally, users with no internet connectivity will not have any data packets delivered to the server (3.2). Therefore, the data is not necessarily a fully accurate representation of the usage patterns of Scratchers around the world, but nonetheless provides valuable insight.
4.1 Overall user engagement

These metrics give us an overview of how frequently Scratchers are interacting with the Standalone Desktop Editor (SDE) and how often are they sharing their work with the online community. Metrics in this category include the following (for a given time interval):

1. **Total number of times the SDE was opened.** The total number of "app::open" packets (3.1.1)

2. **Number of unique clients that use the SDE.** The number of unique computers that successfully sent at least one "app::open" packet to the server

3. **Ratio between the number of projects created and the number of projects uploaded** to understand what fraction of projects are shared with the online Scratch community

These metrics can be harnessed to answer various questions about user engagement such as how many users there are of the SDE, how actively they are using Scratch, and how many projects that are created on the SDE end up being shared with the online community. As users continue to upgrade their versions of the SDE and opt into the TDCS, we will continue to get a more precise idea of how users are engaging with Scratch. For example, looking at Figure 4-1, as more and more users started to opt into the TDCS after its deployment at the end of February 2018, we began to get more accurate information about the frequency at which the SDE was being opened.

We can also look at the ratio between the number of projects created and the number of projects uploaded to the online Scratch community as a way to measure how successful the TDCS has been in terms of making previously unknown projects visible. Based on the data collected between the deployment of the TDCS to early May 2018, approximately 157,000 new Scratch projects were created while only 11,234 were uploaded. Therefore, only 6% of the projects that were created using the SDE
Figure 4-1: Number of app::open events that occurred between 01/30/2018 - 04/29/2018

would be known about if the TDCS did not exist. Note that this statistic is only representative of the projects created by users who opted into the system.

4.2 Language / Geolocation distribution

*Note that the data displayed in this section was collected over a time interval of 24 hours and the timezone is Eastern Daylight Time (EDT)

The two figures below (Figures 4-2 and 4-3) give us a much clearer picture of where the SDE is being used by Scratchers. Using this, we can also compare activity across different regions in the world. For example, looking at Figure 4-3, we can see that depending on whether it is day or night in a given region, user activity increases or decreases accordingly. For example, peak activity in countries like China and Korea happens while it is either late at night or early in the morning on the American East
Coast. This is similar to the usage patterns seen with the online user base, as seen in Figure 4-4.

Figure 4-2: Highest language frequencies in "Project Save" event packets

Figure 4-3: Graph displaying the frequencies of all languages used in "Project Save" event packets
Additionally, there is a potential for cohort analysis. A cohort is a group of individuals who share a common characteristic such as same age, gender, geolocation, or experience [28]. We can use cohort analysis to compare usage patterns across various groups. For example, we can compare projects created by new Scratchers to more experienced children to get a better understanding of the learning process that Scratchers follow. We can also examine projects in different regions to see if there are any patterns in the types of projects being created based on the region.

4.3 Usage patterns

While data on user activity volume helps us understand the frequency of usage of the SDE, we also want to understand the behavior and trajectories of Scratch users. To understand these deeper levels of engagement, we use metrics such as the total number of "Save Project" events (Figure 4-5), the number of projects that were uploaded to the Scratch community (Figure 4-6), and the number of unique clients that uploaded projects (Figure 4-7).

We also aggregate the various counts for blocks, costumes, lists, scripts, sounds, sprites, and variables. This gives us an overall idea of what the average Scratch project looks like in terms of composition. For example, looking at figures 4-8 and 4-9 below, we can compare the average Scratch project created by Scratchers who set German as their language to those who set Brazilian Portuguese as their language. Note that the aggregates shown below are computed over all available data.
Figure 4-5: Number of "Save Project" events that occurred over a 7-day interval

Figure 4-6: Number of projects that were uploaded to the online Scratch community over a 7-day interval
Figure 4-7: Number of unique clients that uploaded projects to the online community over a 7-day interval

![Graph showing number of unique clients uploading projects](image)

**Table 4-7:**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique clients</td>
<td>455,245</td>
</tr>
<tr>
<td>Average blocks</td>
<td>36.13</td>
</tr>
<tr>
<td>Average costumes</td>
<td>2.079</td>
</tr>
<tr>
<td>Average list</td>
<td>34.908</td>
</tr>
<tr>
<td>Average sprite</td>
<td>12.311</td>
</tr>
<tr>
<td>Average sound</td>
<td>9.35</td>
</tr>
<tr>
<td>Average variables</td>
<td>9.917</td>
</tr>
</tbody>
</table>

Figure 4-8: Aggregates for projects created by users who set German ("de") as their language

**Table 4-8:**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average blocks</td>
<td>138,493</td>
</tr>
<tr>
<td>Average costumes</td>
<td>19.074</td>
</tr>
<tr>
<td>Average lists</td>
<td>0.2</td>
</tr>
<tr>
<td>Average sprite</td>
<td>15.902</td>
</tr>
<tr>
<td>Average sound</td>
<td>7.977</td>
</tr>
<tr>
<td>Average variables</td>
<td>5.256</td>
</tr>
</tbody>
</table>

Figure 4-9: Aggregates for projects created by users who set Brazilian Portuguese ("pt-br") as their language

**Table 4-9:**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average blocks</td>
<td>7.977</td>
</tr>
<tr>
<td>Average costumes</td>
<td>5.256</td>
</tr>
<tr>
<td>Average lists</td>
<td>2.856</td>
</tr>
<tr>
<td>Average sprite</td>
<td>19.074</td>
</tr>
<tr>
<td>Average sound</td>
<td>15.902</td>
</tr>
</tbody>
</table>

Based on these numbers, we can infer that the projects created by German speakers contain more blocks, sounds, costumes, and sprites, on average which potentially
indicates more complex projects. The number of variables and lists are slightly higher too which may indicate increased data access complexity [32]. With deeper analysis, we can evaluate how project complexity changes for individual users as they gain more experience with Scratch, thus helping us better understand how Scratchers learn over time.

4.4 Opt-in rate

To estimate the opt-in rate, we compared the number of downloads of the Standalone Desktop Editor (SDE) for a given time interval of six hours to the number of unique client IDs (3.1.1) that generated their first packet during that time. We collected data for 20 different sets of 6-hour time intervals between April 4th, 2018 and April 23rd, 2018. For each set, we computed a conversion rate which was simply the number of new unique clients divided by the number of downloads. The median conversion rate was 6.48% with a standard deviation of 2.70% and a confidence level of 99%.

In conclusion, the telemetry data collection system achieved its primary goal of collecting and delivering SDE usage and summary data to the Scratch research team. With this system in place, we also laid the foundation to pursue future work to better understand the Scratch community and how Scratch is leveraged around the world.
Chapter 5

Future Work

5.1 Classification of Projects

We are interested in building a classifier that uses the block composition of a Scratch project as a feature-set to classify projects into various categories such as games, animations, or simulations. Additionally, we can also determine complexity of projects based on these features. We can then examine differences in project complexities between cohorts with varying levels of Scratch experience to better understand the learning process and progression that children go through.

5.2 Mechanism to update Consent setting

While we explicitly ask users to opt into sharing their data with us, we don’t have a mechanism in place where they can change this setting. For example, if a user decided to opt into the system but then wanted to stop sharing their data, there is no way for them to revoke consent at this time.

5.3 Bandwidth Adaptation

The Standalone Desktop Editor (SDE) is used all around the world. As a result, we cannot expect all users to have access to a reliable and affordable internet connection.
Therefore, it is crucial that the system accounts for these constraints. Ensuring the packet size is as small as possible is a good first step in this direction. The next step would be to make our system flexible so that it can adapt to varying network conditions. For example, we can adjust the number of times we attempt to retransmit a packet based on the speed of the network. If the user has access to more bandwidth, we could modify the packet schema to contain more data. We could also adjust the size of the Offline Queue so that it can store more or less packets on disk depending on the reliability of the network. Several such steps can be taken to ensure that the system is more robust and can function properly in any network setting.
Chapter 6

Conclusion

We have designed and developed a telemetry data collection system (TDCS) to collect and deliver usage data from regions with limited internet connectivity to the Scratch research team at the MIT Media Lab. To handle the network constraints, we minimized the size of the event packets to reduce bandwidth usage. Additionally, we utilized a packet retransmission strategy to make our system fault-tolerant against network failures. Finally, we used several techniques to reduce the chance of packet loss by only sending event packets when certain network conditions were met.

We have shown that the telemetry data has given us the ability to answer a range of questions such as how Scratchers engage with the Standalone Desktop Editor (SDE), what is the geo-distribution of Scratchers using the SDE, and what content is being generated using the SDE. With deeper analysis, we can follow the learning trajectory of Scratchers in hopes of better understanding how they learn. Also, as developers and designers of Scratch, we can use the telemetry data to find ways to improve the learning experience.

In conclusion, the TDCS is the first step in our goal to better understand previously hidden users of Scratch. Using the telemetry data, we hope to boldly go where no previous Scratch researcher has gone before [27].
Chapter 7

Bibliography


53


