

# A Data Collection and Analysis System for Yenta

by

Katherine E. King

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

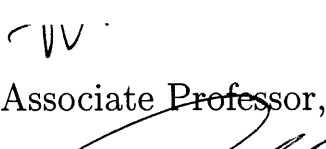
Massachusetts Institute of Technology

June 1998

© Katherine E. King, MCMXCVIII. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part and to grant others the right to do so.

Author .....  
Department of Electrical Engineering and Computer Science  
May 22, 1998

Certified by....  
  
Pattie Maes  
Associate Professor, Media Arts and Sciences  
~~Thesis Supervisor~~

Accepted by .....  
  
Arthur C. Smith  
Chairman, Department Committee on Graduate Theses

JUL 14 1998

LIBRARIES

# A Data Collection and Analysis System for Yenta

by

Katherine E. King

Submitted to the Department of Electrical Engineering and Computer Science  
on May 22, 1998, in partial fulfillment of the  
requirements for the degrees of  
Bachelor of Science in Computer Science and Engineering  
and  
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

Yenta is an experimental system designed to match people on the Internet who have similar interests. It uses a distributed, multi-agent architecture to form clusters of users who share interests. Users in a cluster may send messages to the cluster's other members and may request introductions to particular individuals in the cluster.

This thesis is the design and implementation of a data collection system for Yenta. Yenta was instrumented to enable the automatic collection of usage statistics, and a logging server was set up to receive the data. When Yenta is released, data will be collected and analyzed to evaluate Yenta's clustering and communications algorithms and the success of Yenta as a whole.

Thesis Supervisor: Pattie Maes

Title: Associate Professor, Media Arts and Sciences

## Acknowledgments

Thank you to Pattie Maes for sponsoring my work and my thesis.

Thank you to Lenny Foner, for advising, teaching, nagging, reading, and putting up with me.

Thank you to Daniel and Aaron, for help with code and for answering my questions.

Thank you to my parents and to Cat, Erik, Erin, Yonah, Gisele, Shabby, Rich, and everyone else who offered support, encouragement, and hugs.

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Overview of Yenta . . . . .	7
1.2	Statistics Collection . . . . .	8
1.2.1	Rationale . . . . .	8
1.2.2	Methods . . . . .	9
1.2.3	Privacy Concerns . . . . .	9
1.3	Structure of this Thesis . . . . .	10
<b>2</b>	<b>Description of Yenta</b>	<b>11</b>
2.1	Why Use Yenta? . . . . .	11
2.2	Interface . . . . .	12
2.3	Interest Finding . . . . .	12
2.4	Bootstrapping . . . . .	13
2.5	Clustering . . . . .	13
2.6	Affordances . . . . .	14
2.6.1	Cluster Messages . . . . .	15
2.6.2	Individual Messages . . . . .	15
2.6.3	Introductions . . . . .	15
2.6.4	Finding an Expert . . . . .	16
2.6.5	Attestations . . . . .	16
2.7	Privacy and Security Concerns . . . . .	17
2.8	Yenta Architecture . . . . .	18
2.8.1	Yenta Agent Application . . . . .	18

2.8.2	Logging Server . . . . .	22
<b>3</b>	<b>The Data-Collection System</b>	<b>25</b>
3.1	Agent Instrumentation . . . . .	25
3.1.1	Counters . . . . .	26
3.1.2	Event Log . . . . .	27
3.2	Logging Client . . . . .	28
3.2.1	What is Logged . . . . .	28
3.2.2	When to Log . . . . .	29
3.2.3	Sending the Log . . . . .	30
3.3	Logging Server . . . . .	30
3.3.1	Initialization . . . . .	31
3.3.2	Accepting Connections . . . . .	31
3.3.3	Receiving Data . . . . .	31
3.3.4	Timeouts . . . . .	32
3.3.5	Other Considerations . . . . .	32
3.4	Privacy and Security Considerations . . . . .	33
3.5	Testing . . . . .	34
<b>4</b>	<b>Analysis</b>	<b>36</b>
4.1	Goals . . . . .	36
4.1.1	User Preferences and Habits . . . . .	37
4.1.2	Clustering . . . . .	37
4.1.3	Messages, Introductions, and Attestations . . . . .	38
4.2	Means . . . . .	39
4.2.1	Parsing . . . . .	39
4.2.2	Graphing . . . . .	40
4.3	Results . . . . .	40
<b>5</b>	<b>Conclusion</b>	<b>42</b>
5.1	Summary . . . . .	42

5.2	Future Work . . . . .	42
5.2.1	Adjusting the Logging Frequency . . . . .	42
5.2.2	Acknowledging Log Receipt . . . . .	43
5.2.3	Flushing Logs to Disk . . . . .	43
5.2.4	An Analysis of the Reputation System . . . . .	43
<b>A</b>	<b>Counters</b>	<b>45</b>
<b>B</b>	<b>Events</b>	<b>50</b>
<b>C</b>	<b>Sample Log Entry</b>	<b>51</b>

# Chapter 1

## Introduction

The design and implementation of Yenta is the primary software project of an ongoing Ph.D. thesis at the MIT Media Lab [4] [6]. An experimental system, Yenta is designed to find and introduce people on the Internet who share interests. Users run separate, individual agents, which communicate with one another to form clusters of users with like interests. Users in a cluster may send messages to the cluster's other members and may request introductions to particular individuals in the cluster.

This project is the design and implementation of a data collection system for Yenta. It prepares Yenta for evaluation as a distributed, privacy-protecting match-making system. I instrumented Yenta to enable the automatic collection of statistics and implemented a logging server to receive the data. When Yenta is released, data will be collected and analyzed to determine strengths and weaknesses in Yenta's clustering and communications algorithms and to evaluate whether Yenta fulfills its goals.

### 1.1 Overview of Yenta

Yenta is designed to securely introduce people on the Internet who share some interest, without requiring them to reveal more information about themselves than they want to.

Yenta is a distributed, decentralized system. Each user runs his own copy of Yenta, which runs as a background process and acts as an agent for the user. Yenta

agents never communicate with a central server in order to accomplish their goal of matchmaking. However, in order to monitor Yenta’s operation as part of a research project, Yenta was modified, as described in this thesis, so that each agent reports statistics to a central server for collection and analysis.

Each Yenta agent determines its user’s interests based on keywords found in personal files. The agents then communicate with each other to find matches between their users’ interests. When they find matches, they form a *cluster*. Yentas may be in multiple distinct clusters.

Once clusters have formed, agents can send messages to other agents in a cluster. They can make statements called *attestations* about themselves using Yenta’s *reputation system*. They can also, at the user’s instruction, request introductions to other people in the cluster or ones that seem a good match. Clustering – both finding interests for a user and matching them to interests in other Yentas – is an ongoing process, since the data and the pool of agents is constantly changing.

Yenta’s communications, both between individual Yenta agents and between the user’s Yenta and his or her web browser, are encrypted. This prevents many common attacks against user privacy and the system as a whole [5].

## 1.2 Statistics Collection

The purpose of this thesis is to design and implement a data collection and analysis system for Yenta.

### 1.2.1 Rationale

Yenta is an experimental system that will be distributed to real users for testing. As it is a research project, we need some means to evaluate its success. We are interested in how well the agents run and how well the system works as a whole. We would also like to know how many and what sorts of clusters were formed, how the reputation system was used, how users tended to configure their agents, etc.



### 1.2.2 Methods

In order to gather information for evaluation of the Yenta system, we have chosen to collect extensive quantitative data from running agents, and to log the data to a central location, where it will be analyzed offline.

The Yenta agent code that each user runs has been augmented to collect a large amount of data about what that Yenta does and sees. These statistics tell us about the state of the Yenta agent and what events have transpired in its lifetime. They are chosen to give us information about the Yenta system as a whole, while not revealing private information about individual users.

If statistics logging is enabled for a Yenta agent, the agent automatically sends periodic logs, with the values of counters, parameters, and other statistics, to a server set up for this purpose. The server receives and stores the data on disk.

When Yenta is released, we will gather data from all participating Yenta agents. The collected data will be converted to readable forms such as tables, graphs, and charts, and analyzed in order to draw conclusions about the success of Yenta.

### 1.2.3 Privacy Concerns

Yenta is designed to protect the privacy of individual users while accomplishing its matchmaking goals. We have attempted to adhere to the same standards of user privacy while implementing the statistics collection system. Some of the issues considered are as follows:

- Participation in data collection is voluntary; users may choose not to log statistics, without affecting their use of Yenta.
- To prevent eavesdropping, all data is encrypted during transmission to the server.
- The identification information in the statistics is blinded to keep the statistics from being traced back to individual Yentas once it has reached the server.

## 1.3 Structure of this Thesis

Chapter 2 describes the Yenta system as it has been implemented. Chapter 3 discusses the modifications made to the system in order to collect data about Yenta. This includes both the instrumentation of the agents and the design of the logging server. Chapter 4 discusses the organization and analysis of the collected data. Chapter 5 concludes the report and suggests future improvements. Appendix A lists all the current counters, Appendix B lists the events that are logged, and Appendix C gives a sample log entry. A bibliography is included at the end.

# Chapter 2

## Description of Yenta

Yenta is designed to securely introduce people on the Internet who share some interest, without requiring them to reveal more information about themselves than they want to.

Yenta is a distributed, decentralized system. Each user runs his own copy of Yenta, which runs as a background process and acts as an agent for the user. Yenta agents talk each other in order to facilitate bringing people together who have similar interests. In this way, Yenta models the way people often meet each other – by being introduced by a mutual friend. Agents never communicate with a central server in order to accomplish their goal of matchmaking. However, agents may briefly query a server as a starting point (see Section 2.4); additionally, in order to monitor Yenta’s operation as part of a research project, Yenta was modified, as described in this thesis, so that each agent reports statistics to a central server for collection and analysis.

### 2.1 Why Use Yenta?

There are several reasons why one might want to use such a system [4]:

- People may be working on similar projects without realizing it. Yenta can help these people to find each other.

- People often wish to find an expert in a certain field. This can involve “asking around,” or finding a group which knows a little about the field and asking them for a contact who knows more. Yenta can take care of these tasks for the user.
- Rather than require that a user list his interests or choose from a given list, Yenta determines interests from documents the user has written and received (see Section 2.3). Thus Yenta may list interests for a user that the user might not have thought about, and automatically keeps the interest list up to date.
- With traditional online interest groups, such as mailing lists and Usenet, people are only noticed if they post something publicly. Users never find out about people who only read and do not post, although these silent users abound. Yenta can match people based on things they have received as well as what they have written, and do so in a way such that they are not publicly exposed.

## 2.2 Interface

Yenta itself must be run under Unix. However, the user may be on a different computer, as long as she can run a web browser capable of SSL (this includes most common browsers). The Yenta agent runs an HTTP server [3], and the user communicates with her Yenta through a web browser of her choice. All communications between a user’s web browser and her Yenta are encrypted using SSL [14].

## 2.3 Interest Finding

Each Yenta agent sifts through the user’s personal files and forms *granules* which describe the user’s interests, based on keywords found in those files. Users choose which documents Yenta should scan. These may include personal or list mail that the user has received, newsgroup articles that the user reads, or other files of the user’s choice. In addition, a user may explicitly describe an interest for his Yenta to

use.

Given documents from the user, Yenta uses Savant, the back end for the Remembrance Agent software [12], to determine the user's interests. Savant processes each document and records what non-common words appeared in the document and how often each occurred. Yenta takes the resulting document vectors from Savant and compares them for similarities. Clusters of words that tend to occur together and that tend to occur with some frequency are noted as possible interests. The user can look over the list that Yenta has collected and choose which ones to use when contacting other Yentas.

## 2.4 Bootstrapping

When a Yenta agent is started for the first time, it needs to find other Yentas, or it will never get to talk to anyone. To accomplish this, the Yenta tries three things, in order:

1. It queries the local network for other running Yentas.
2. If the new Yenta receives insufficient response from the local network, it queries a designated bootstrapping server and receives a list of the IP addresses of the last few Yentas to come up.
3. If the previous two attempts fail, Yenta will ask the user to specify the addresses of some other running Yentas.

## 2.5 Clustering

The agents talk with other agents they are aware of to find matches in their interests. When two Yentas meet, they reveal information bit by bit to determine similarities in the interests of their respective users. If they find a common interest, the Yentas form a *cluster*. When these agents meet others, they again compare interests, joining clusters when an interest overlaps sufficiently.

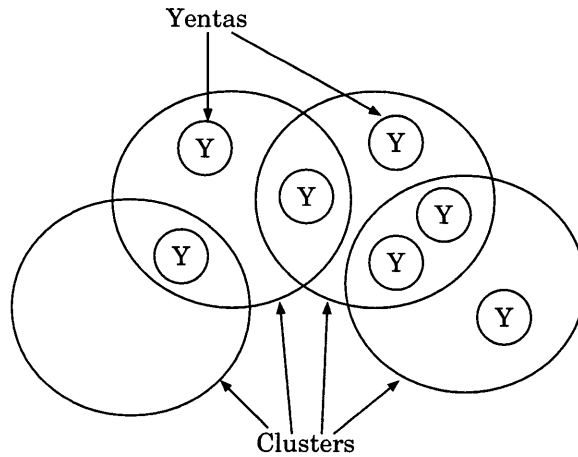


Figure 2-1: Yenta Agents in Clusters

A Yenta may be in multiple distinct clusters (Figure 2-1). Clusters are constantly changing, as the user's interests change and as agents discover other Yentas. Agents may enter or leave clusters as interests change.

A Yenta's *cluster cache* stores information about clusters the Yenta has joined. A Yenta's *rumor cache* stores information about other Yentas it has talked to recently. The rumor cache allows the agent to make *referrals* to other Yentas, to help them locate possible matches.

## 2.6 Affordances

Once clusters have formed, how can they be used? Yenta users may send messages to other Yentas. These messages may be directed to groups or to individuals. Users may attempt to find an expert, and they may request introductions to other users. Users may use Yenta to make statements about themselves. The following sections describe some of these uses.

### **2.6.1 Cluster Messages**

A user may send messages to any cluster that he is in. The message will be sent out to all the Yentas that are known to be in the cluster. These agents will in turn spread it to others they know of in the same cluster. When a Yenta receives a cluster message, it may choose to accept the message and display it for its user to read. Because of the way messages are propagated, and the fact that each Yenta will only know about a subset of any cluster, Yenta may receive a message more than once. When this happens, Yenta simply discards the duplicates. Additionally, users may request not to see certain messages. They may decide, for example, to reject all cluster messages, or to accept messages only from Yentas which meet specific qualifications. Cluster messages are not guaranteed to reach all possible recipients.

### **2.6.2 Individual Messages**

Users may also send messages to specific Yentas. These may be Yentas that they know about from cluster messages, or perhaps users that they know through channels other than Yenta. If the user knows the address of the computer on which the other Yenta is running, he may give it to his Yenta and have the message sent there. Otherwise, his Yenta encrypts the message with the public key of the recipient and sends it to other Yentas it knows of in a cluster that the sender and recipient are both in. Only the intended recipient – the possessor of the private key – can decrypt and read the message; other Yentas will simply pass the encrypted message along to other Yentas in the cluster.

### **2.6.3 Introductions**

When an agent finds through referrals another Yenta that may be a particularly good match for itself, it can request an introduction to the other Yenta, in which information is symmetrically and gradually revealed. A user may request explicit introduction to particular members of a cluster, or she can instruct her agent to request or accept introductions when it thinks it has found a good match [6]. When a

Yenta agent receives a request from another Yenta for an introduction, it may accept or deny the request, according to what its user has asked.

#### **2.6.4 Finding an Expert**

Users may use Yenta to seek help on a particular topic. The user expresses that topic as an interest, and the agent queries its cache of known Yentas to find a cluster for the interest and seeks for a Yenta who claims to be an expert in that area, presumably via the attestation system (Section 2.6.5). When an expert is found, an introduction can be made.

#### **2.6.5 Attestations**

With Yenta’s *reputation system*, users of Yenta may make statements about themselves, called *attestations*. For example, a user may proclaim, “I own a cat,” “I will not send spam mail,” “I work for Company X,” or anything else he or she can think of.

When Mary, a Yenta user, is talking to another Yenta, she may request to see the other user’s attestations. If she agrees with the accuracy of an attestation (typically through personal knowledge of the other user or from having known the user via Yenta for some time), she can cryptographically sign it. Then, other users who look at that attestation will see Mary’s signature, and know that she agrees with the statement. Yentas automatically sign and date any attestations they make themselves, so they cannot be easily forged. Other users can tell when the attestations were made, and place a lower bound on how long the Yenta has been around.

An analog to real-life reputations, attestations – and whether or not they are signed by other users – may influence how Yentas interact with one another and how users view other users. A user may allow or refuse contact from another Yenta based on its collection of attestations and signatures.



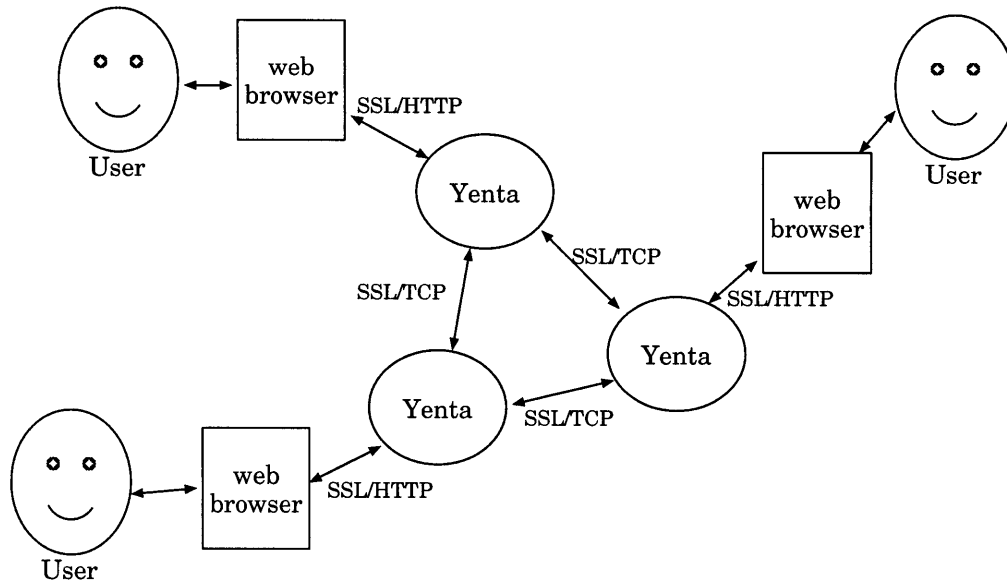


Figure 2-2: Security in Yenta's Communications

## 2.7 Privacy and Security Concerns

The privacy of its users and the security of data are foremost in Yenta's concerns. Following are some of the means Yenta uses to guard against attacks against user privacy and the system as a whole [5].

Yenta users are known to each other by handles or pseudonyms. Yenta agents are known by a Yenta ID, and a public key. Users may further reveal their identities to other users if they wish, but are by no means required to do so.

A Yenta agent does not traverse the user's private files and mail. But Yenta does *not* reveal this information wholesale to other Yentas. Instead, it forms keyword vectors from the documents. Yentas then jointly compare these vectors, revealing information gradually and symmetrically [4].

To prevent eavesdropping, all of Yenta's communications are encrypted. Communications between a user's Yenta and his web browser use SSL over HTTP. Communications between Yenta agents and between Yentas and a server are encrypted with SSL over TCP (Figure 2-2). Personal messages are encrypted using a public/private key encryption system. A Yenta agent authenticates itself to its user by giving a

public key fingerprint. A user authenticates to her Yenta with a password.

We would like to be able to collect information from running Yentas. This includes debugging logs, bug reports, and statistics. However, in the interests of privacy, any user who objects to her Yenta sending reports may easily disable this property.

So that users may detect forged or modified distributions, Yenta uses cryptographically signed binaries. For those who wish to inspect the source, Yenta's source code is freely available. Anyone may examine the code to assure himself of its soundness. In addition, the Yenta developers have created *Yvette*, a Web-based tool for examining and evaluating the code for Yenta. Using Yvette, anyone may view any piece of the source code,<sup>1</sup> evaluate it, comment on it, and cryptographically sign their comments. Other users can see which parts of the code have been vetted, and by whom, and base their trust of the Yenta code on this knowledge [5].

## 2.8 Yenta Architecture

This section briefly describes aspects of Yenta's architecture. Many of these are described adequately in previous sections of this chapter. Aspects of the architecture that do not affect the user but are important for the understanding of this thesis are given in more detail.

### 2.8.1 Yenta Agent Application

Figure 2-3 shows the various pieces of the Yenta agent implementation. SCM, TCP, SSLeay, and Savant are written in C. The other pieces of the application are written in SCM.

#### SCM

Much of Yenta is implemented in SCM [7], a Scheme [1] interpreter written in C.

---

<sup>1</sup>Within the cryptography export regulations, as specified in EAR [2], formerly ITAR [9].

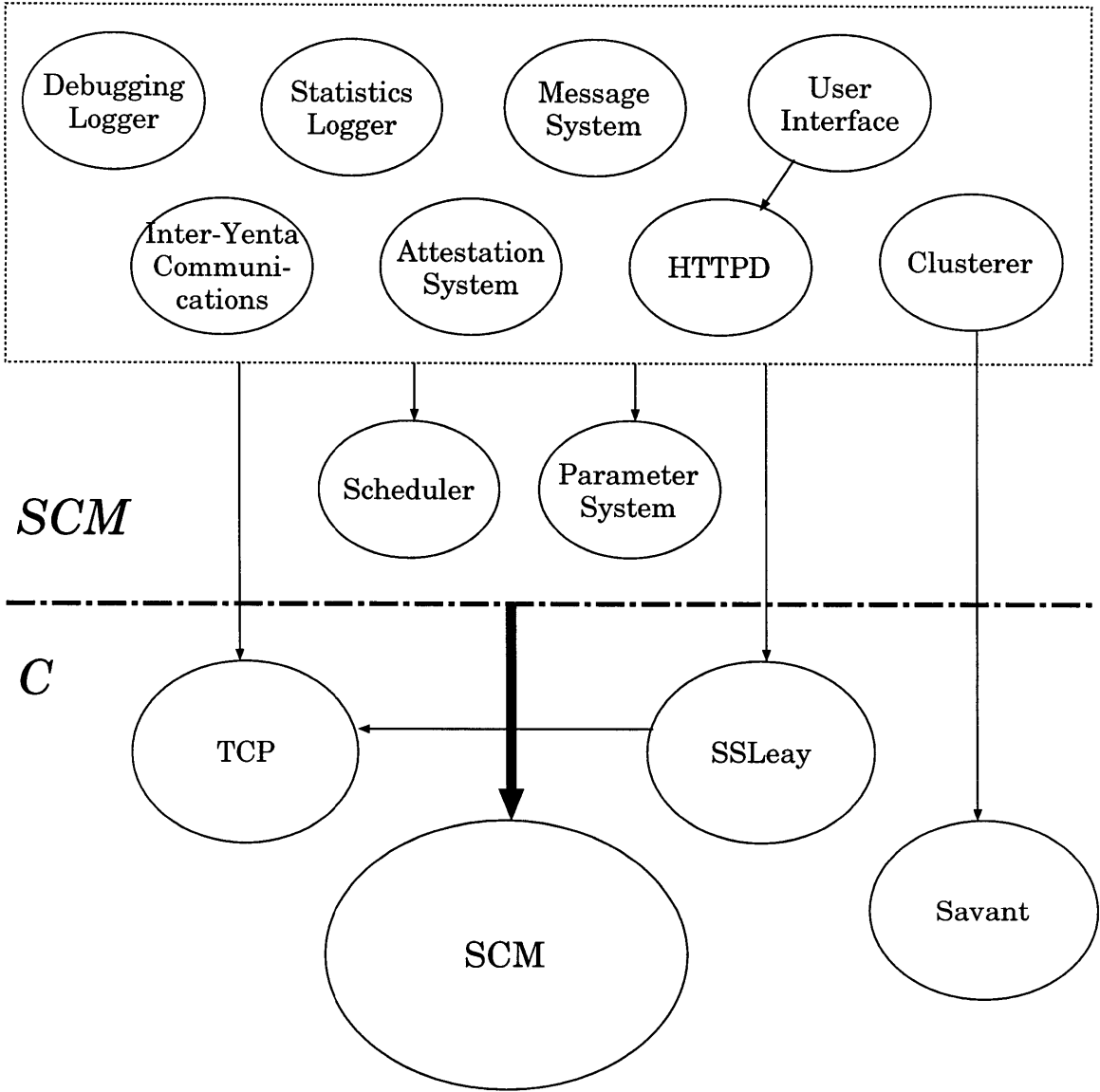


Figure 2-3: Architecture of the Yenta agent application. Arrows show dependencies.

## **TCP**

The Yenta developers have written in C an implementation of TCP [11] for SCM. TCP is used throughout Yenta for network communications.

## **SSLey**

SSLey [16] is an implementation of the open, nonproprietary Secure Sockets Layer (SSL) protocol, originally developed by Netscape. SSL “provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection” [14]. SSLey was developed in Australia and is therefore available worldwide, directly from its source, regardless of U.S. cryptography export regulations which prohibit its re-export from the U.S. [2]. SSLey is written in C and is imported into Yenta’s C code. It is used extensively for authenticated, encrypted communications between two Yenta agents, between a Yenta agent and a web browser, and between a Yenta agent and a server.

## **Savant**

Yenta uses Savant to form document vectors from user documents. The resulting vectors are used when determining a user’s interests and in the clustering algorithms to find matches between Yentas. Savant is written in C by the developers of Yenta and those of the Remembrance Agent [12].

## **Scheduler**

Yenta implements its own scheduler in Scheme. The scheduler is non-preemptive, single-address-space, and multithreading. It acts as the toplevel loop for the Yenta application. A *task* is some piece of Scheme code that is scheduled and run by the scheduler. All Scheme-level code depends on and is controlled by the scheduler. Any Scheme part of Yenta may schedule or remove tasks within Yenta.

## **Parameter System**

The architecture of Yenta allows the developers to define Scheme variables and parameters whose values are retained across sessions. Some of these variables can be set only by the Yenta code; others may be adjusted by the user. When values of such variables are changed, the changes persist across shutdowns, rather than being reset next time Yenta is restarted. The collection of persistent variables is saved to disk periodically and before a Yenta shutdown, to avoid losing data. The persistent parameter system is used throughout the Yenta code, wherever persistent values are needed. The counters used in the statistics logging system, discussed below and in Section 3.1.1, are implemented on top of the parameter system.

## **User Interface**

Yenta communicates with its user entirely through a web browser. The output is HTML, and the input is via forms and links.

## **HTTPD**

The base of the user interface is the HTTP server that the agent runs. It relies on SSLey for secure communication between the Yenta application and the user's web browser, which may be running on different computers.

## **Clusterer**

The clusterer compares the document vectors produced by Savant to determine interests for a user and to match up interests between users. Cluster information for a Yenta agent is stored in its cluster cache. Information about other Yentas that the agent has talked to but has not necessarily matched with are stored in its rumor cache.

## **Inter-Yenta Communication**

Yentas talk to each other to share information. Communications between Yentas are over TCP and encrypted via SSLeay.

### **Message System**

The message system allows users to send messages to other Yenta users. Messages may be sent to all Yentas in a cluster or encrypted for individual users. The message system relies on TCP and SSLeay to pass messages between Yentas.

### **Attestation System**

With the attestation system, users make statements about themselves, which they and other users may cryptographically sign. The attestation system uses public-key cryptography for signatures on attestations.

### **Statistics Logger**

The statistics logger was implemented for this thesis. It is described fully in Section 3.2. The statistics logger uses the scheduler to schedule tasks; the parameter system for persistent parameters and to implement counters; and TCP and SSLeay for communication with the server. It gathers data from all parts of the application.

### **Debugging Logger**

The debugging logger is used during testing and debugging of Yenta to record debugging and error information. It is separate from the statistics logger and should not be confused with it.

## **2.8.2 Logging Server**

Figure 2-4 shows the various pieces in the implementation of the Yenta statistics logging server. Note that the logging server is based on the same foundation as the

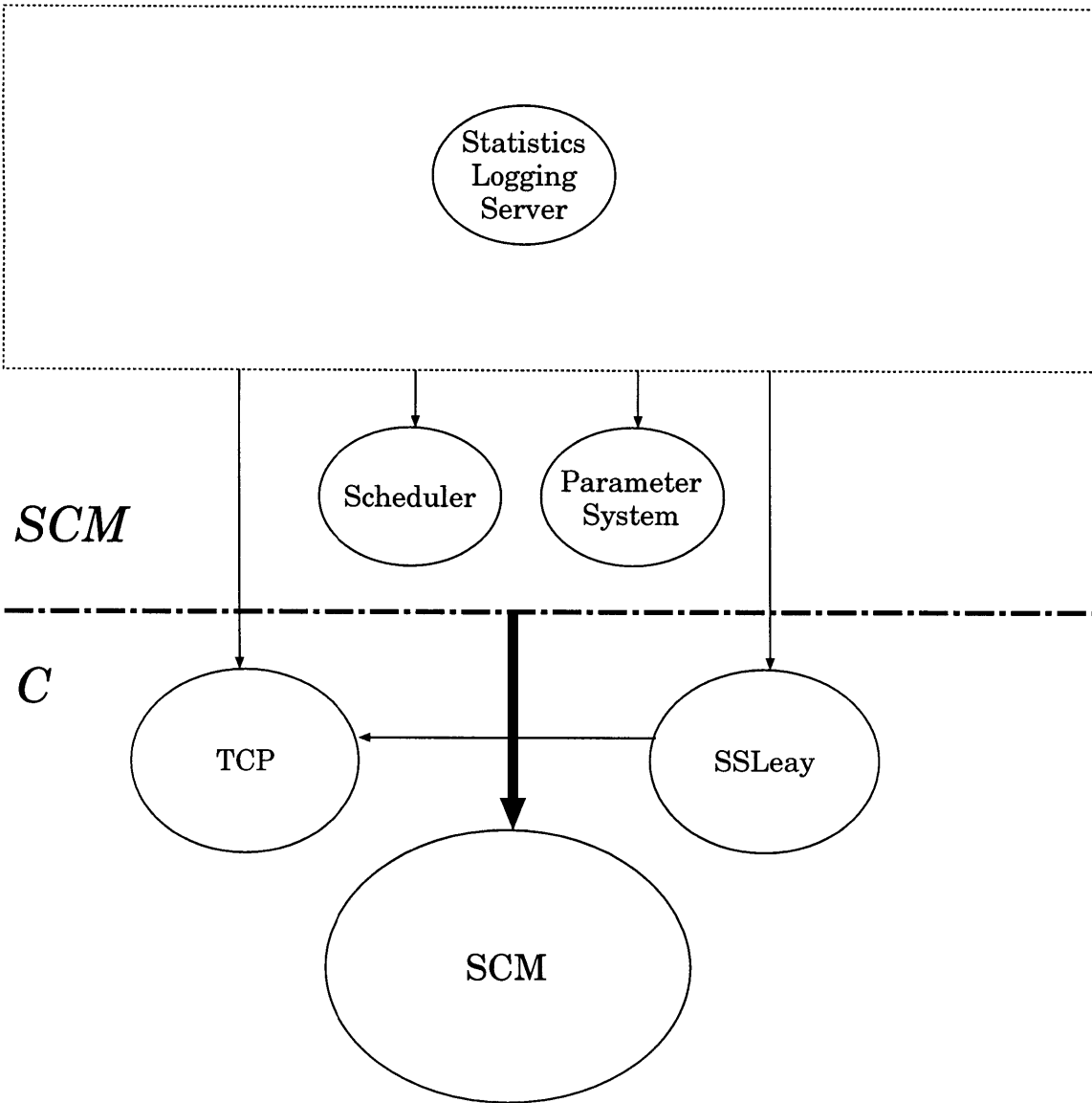


Figure 2-4: Architecture of the Yenta statistics logging server. Arrows show dependencies.

agent, but that much of the agent/client functionality is not needed. The statistics logging server is discussed further in Section 3.3.



# Chapter 3

## The Data-Collection System

The data collection system consists of client code included in each Yenta agent, and logging server code run on a machine designated to receive statistics logs.

First, the agent code was instrumented to keep track of data in a form that can be sent to a server. Then client logging code was written to send that data to the server. These are both distributed as part of the Yenta agent and run by all users. The statistics logging server, run by the Yenta developers, receives and stores the data.

The following sections describe the implementation of the data collection system, and the security considerations taken into account.

### 3.1 Agent Instrumentation

This section describes the modifications made to the Yenta agent code – the code that each user runs – in order to enable gathering of statistics for that agent. The primary additions were counters, which keep track of how many times things have happened in a Yenta’s lifetime, and event logging, which keeps track of when the Yenta does certain things.

### 3.1.1 Counters

Counters keep track of the number of occurrences of events, so that we know what the history of a given Yenta is even if we have missed log entries from it. They begin at zero when the Yenta is started for the first time, and are incremented appropriately when the Yenta does and sees certain things.

The existing architecture of Yenta allowed the developers to define variables and parameters whose values are retained across sessions. Some of these can be set only by the Yenta code; others may be adjusted by the user. I defined a new type of parameter, called a counter. Counter values may be any nonnegative integer, but they are only incremented, and never reset over the lifetime of the Yenta. Users may view the values of counters but may never directly change them.

Counters are all defined in one place, but are incremented by code throughout Yenta, wherever the corresponding events happen. For example, the attestation code would include the following line in the code that creates attestations:

```
(counter:increment! *ctr:atts-made*)
```

So every time this piece of code is run, the value of the counter `*ctr:atts-made*` is increased by one.

Approximately 40 counters were defined. They include:

- System operations counters – number of startups, shutdowns, errors, bug reports, and time this Yenta has been alive.
- User interface counters – number of pages and documentation pages fetched.
- Inter-Yenta communication and network statistics – connections initiated and served, opcodes sent and received, network errors, total bytes sent and received, and authentication failures.
- Preclustering counters – number of and total size of documents read, and the number of rescans and reclusters performed.
- Matchmaking, clustering, and messaging counters – number of Yentas encountered, number of clusters joined and left, number of introductions initiated and

responded to, the number and total size of individual and cluster messages sent and received.

- Attestation system – number of attestations made and the number fetched, and the number of signatures made and received.

A full list of counters is included in Appendix A.

### 3.1.2 Event Log

Events tell us what has happened to a Yenta and what it has done. When certain events (outlined below) transpire during a Yenta’s lifetime, they are recorded, with a timestamp, into a local event log. This event log is implemented using the parameter system, so that it, like other Yenta variables, is periodically saved to disk so that not much is lost in the case of a crash. Most events that are logged also trigger a counter increment for that event, so the totals are kept even if the event log is lost.

The events that are logged include the following:

- Startup and shutdown
- Contact with another Yenta
- Exchange of cluster information
- Cluster entered or left
- Referral made
- Introduction initiated, granted, or refused
- Message sent or received between users
- Attestation created, signed, or fetched

A full list of events logged is given in Appendix B.

For example, I inserted into the code that creates attestations,

```
(stats:log-event ' :attestation-created)
```

So every time this piece of code is run, the symbol `:attestation-created` is appended with a time to the event log.

Certain events may cause Yenta to report soon or immediately to the logging server. This is discussed further in the following section.

## 3.2 Logging Client

This part of the Yenta agent code, also run by each agent, formats the statistics and sends them to the server.

### 3.2.1 What is Logged

Each log entry includes the following:

- A 64-bit random key unique to the Yenta and used only for reporting statistics. This key is generated when the Yenta first starts and is remembered thereafter. This key is discussed further in Section 3.4.
- The time of the message, in universal coordinated time (UTC).
- The current values of all the counters. Counters are maintained by various pieces of code throughout Yenta, as described in Section 3.1.1, and are gathered together by the logging client when needed.
- The values of all user-settable parameters. These include the various thresholds and preferences the user has set through the interface. Like the counters, these parameters are maintained by the appropriate parts of the Yenta code and gathered together when they are needed.
- All attestation strings, and the number of signatures on each attestation. The attestations are stripped of the Yenta ID normally attached to them, and the signatures themselves are not sent, only a count of them (see also Section 3.4).
- The number of interests known for this user. This is computed from the data structure that keeps track of the Yenta's interests.
- The number of clusters this Yenta is in, and the number of known Yentas in each. These are computed from the data structure that keeps track of the Yenta's interests and corresponding clusters.

- The number of Yentas this Yenta is currently talking to. This is maintained by the inter-Yenta code.
- A list of events which have transpired since the last log transmission (the event log). Each time an event is logged locally anywhere in this Yenta, it is added to the event log, as described in Section 3.1.2.

The statistics are collected as a Scheme list of (`attribute value`) pairs, which is formatted as a string. A typical log entry is given in Appendix C.

### 3.2.2 When to Log

A Yenta client sends updates to the server each time it is shut down and periodically while it is running. There are three ways that logging may be initiated.

1. A `:shutdown` event automatically triggers logging to the server. Data – all counter values, parameters, and the event log – are preserved across shutdowns; however, we do not know for certain that this Yenta will be restarted, so we always attempt to log just before shutting down.
2. Logging any other event to the event log causes a flag to be set telling the client to log to the server “soon.” This flag is checked every 30 minutes, and if it is set, the client logs to the server.
3. Since we want some logging even if the Yenta is completely idle, we set a timeout. When this timeout (3 hours) expires, the Yenta client logs to the statistics server. This timer is reset after each successful log.

Each of these timeouts is configurable. The current values are chosen as reasonable starting points. When there are few Yentas and we are still in the debugging stage, we would like them to log more often; when there are many Yentas being very active, each can log less often. Since we may want to change how often we receive data, over time, we would like to have the server be able to change the timeout during a connection. This feature is not yet implemented.

### 3.2.3 Sending the Log

When logging is triggered, the client compiles the log entry as given above. The result, as stated, is a string in the form of a Scheme list.

The client attempts to open a TCP connection to the designated statistics logging port on the statistics server. Once the TCP connection is established, the client and server change over to a encrypted SSL session on a different port. If either the TCP or the SSL connection could not be established, the open ports are closed, and the procedure returns failure to the log cleanup (see below).

When a valid SSL connection is confirmed, the procedure sends the log string to the server, closes the ports, and returns, presuming success. It does not wait for a confirmation from the server (and the server does not send one) – the TCP protocol [11] ensures that once the connection is established, data will only be lost if the server or the network route to the server subsequently fails.<sup>1</sup>

If the logging procedure returns successfully, the event log is cleared. Otherwise, the event log is not erased, and will be sent again in the next attempt. The flags indicating that logging should happen soon or immediately are cleared, and the logging timeout is reset.

If an error is encountered during logging, a connection times out, the logging server cannot be contacted, or an encrypted connection cannot be established, a counter is incremented to keep track of the number of logging errors. No other error information is generated, and none is passed to the user, since logging should not interfere in any way with the user.

## 3.3 Logging Server

The logging server accepts SSL/TCP connections from Yenta agents, and writes the data to disk, with the time in UTC that the message was received by the server. The server is maintained by the Yenta developers.

---

<sup>1</sup>We did not deal with the client waiting for the server to acknowledge receipt of the log. This is addressed in Section 5.2.

### **3.3.1 Initialization**

When the log server starts up, it reads its private key from disk. It verifies that the log file at the given path exists and is writable. If not, it simply creates a new file. Thereafter, before each write, it rechecks the file, appending to the existing one if possible and creating a new file if necessary.

The statistics logging server makes use of Yenta's task scheduler. The task which listens for and accepts incoming log requests, and that which receives data from the current connection and writes it to disk, have equal priority.

### **3.3.2 Accepting Connections**

A scheduler task listens on a designated port for connections from Yentas. The server can have up to five pending connections at a time, enforced by the TCP implementation on Unix systems. Since Yenta clients will not log often, there should not be too much demand for connections to the logging server, and five should be acceptable.

When the server sees that it has a pending connection, it accepts it. It accepts a TCP connection from a client, confirms the connection, and then layers an SSL connection on top of it.

### **3.3.3 Receiving Data**

Once the encrypted SSL connection has been established, the server creates another scheduler task to receive data from the client. This task watches for characters to be ready on the SSL port. When there are incoming characters, they are read one by one, checked for EOF, and stored into a dynamic string buffer.

Up to 1000 characters may be read during one timeslice (this is configurable). This limit ensures that the task will not run forever without yielding, so that other connections may be accepted at the same time. Characters are read until there is no character waiting on the port, the connection is closed, or the limit for the timeslice is reached. If an end-of-file marker (EOF) has not been read by the time the task

yields and the connection is still open, the task will be scheduled again to read more characters.

When an EOF is encountered, the buffer is written to disk with the time that the message was received. The SSL and TCP ports are closed, and the task that receives data from this connection is unscheduled.

There is a 100K limit on the size of a log entry, so that a malicious or misconfigured Yenta client cannot break the server by sending too large a message.

### **3.3.4 Timeouts**

An open connection is considered to be timed out when no characters have been received for five minutes. We determine a timeout by recording the time the most recent character was received and comparing this to the current time. When a connection times out, the data is written to disk with a note labelling it as timed out, the connections are closed, and receive task is unscheduled.

Whether a client sends a full entry and closes the connection on the remote end, or times out without ending the entry, the server terminates its connection cleanly.

### **3.3.5 Other Considerations**

The log file is closed after each write to ensure that the data is flushed to disk. Otherwise, if the server crashes, an arbitrary amount of data may be lost.

The statistics logger is trying only to store data, and does not care if log messages are late, missing, or out of sequence. The data is stored exactly as it is received. All log data should be in the form of a Scheme list; however, for the sake of efficiency in receiving data, the logging server receives and writes the data as a string of characters and does not check for the validity of log entries. Once it is written, analysis programs can parse the data, check for errors, and convert it to more useful forms. These are described in Section 4.2.



### 3.4 Privacy and Security Considerations

We wanted to be able to distinguish one Yenta from another in the statistics, but for privacy reasons we did not want any mapping back to the actual Yentas or users. To this end, when a Yenta logs statistics, it also sends a 64-bit random key unique to this Yenta and used only for statistics logging. This key cannot be traced back to the Yenta, and it provides a unique identifier to label corresponding statistics on the server. No other identifying information is sent. Individual Yentas are responsible for not revealing their logging keys, assuming they do not wish to be traced. Stored data is not identified with individual Yentas except by this logging key. Once received, logs cannot be traced back to a specific Yenta, and so do not need to be encrypted on disk.

The fact that the data is blinded and not tied to a user in any way after transmission should increase user acceptance of statistics logging [4]. However, if the user wishes, he may turn off statistics logging and continue to use Yenta normally.

Attestations by default contain the Yenta ID of the Yenta. Since the data is not to be associated with a Yenta, we of course strip this ID from the attestation before including it in the log. The next question was, How revealing are the attestations themselves? They have the potential to be quite revealing, as this is one area of Yenta in which users are not constrained; they can say whatever they can think of. However, for this very reason, we want to know for our evaluation of Yenta what kinds of attestations people make (see Section 4.1). We decided that, since users are making essentially public statements when they make attestations, and since including them gains us potentially very valuable information, that we would include the attestation strings (minus the Yenta ID) in the log entries. The number of signatures associated with each attestation is included, but no actual signature information (signer, date, or key fingerprint) is sent.

In order to protect the data from eavesdroppers during transmission, it is sent from the client to the server over an encrypted connection. SSL is layered over TCP for all communications between Yenta agents and the statistics logging server. The client

and server first establish a TCP connection, and then they negotiate an encrypted SSL session on top of it.

## 3.5 Testing

The data collection system was tested using Yentas within the development team.

The incrementing procedures and event logging code were first tested separately. Then the increment calls and local event logs were put into the proper places in the code and tested there.

The procedures to gather data from various parts of the Yenta application code were tested separately, to be sure that they dealt correctly with the data structures they read from.

The log transmission was implemented in stages. First, the client and server were both implemented using only TCP for connections. Data was sent in the clear, and the sending and receiving procedures tested. The client logging procedures were called by hand, and the entry that the client thought it was sending was displayed and compared to the entry the server was receiving.

The client was tested in the context of a running agent logging much more frequently than actual agents will log. I made sure that the timeouts worked – that the client attempted to send logs the correct interval after a startup (log “soon”), appropriate intervals thereafter (general timeout), and just before the agent shut down. I discovered that the shutdown code needed to call directly the procedure that logs to the server, rather than just adding a task to do so, so that the scheduler did not shut down before logging could occur.

Next, the SSL cryptography was layered on top of the TCP, and the result was tested. This was not a large addition, but the function calls between the TCP and SSL packages differed enough that This testing helped uncover some bugs in the SCM interface to SSL. For example, there was a timing bug because there was a small delay between the time the connection was established and when characters started appearing at the input port for the server to read. This delay was entirely

reasonable, and will be larger for clients that are farther away. However, the lack of data was being interpreted as the end-of-file marker. The server would think the entry was complete and close the port before receiving any data. This was resolved in the SSL code.

Both the client and the server printed out extensive debugging information during testing, which helped in resolving differences in their interactions.

# Chapter 4

## Analysis

The analysis of the collected statistics is yet to be done and is not part of this thesis. However, in the following sections we discuss plans for organization and analysis of the data.

Once we have a pool of statistics on usage and preferences, we must make sense of the data. We want to be able to see how Yenta was used, which of its algorithms worked well and which did not, and ultimately how well Yenta met its goals as a multi-agent matchmaker.

We will automate the process of converting the data from the stored form into a more useful tabular format. Most of the statistics will be graphed for better analysis. We can then use the data and qualitative observations such as interview with users to answer questions such as those presented in the following section.

Section 4.1 presents our goals in analyzing the statistics, and indeed for collecting many of them in the first place. Section 4.2 describes the sorts of tools that will aid us in our analysis. Section 4.3 offers a view of the results to come.

### 4.1 Goals

The reason we have instrumented Yenta to gather statistics is so we can use those statistics to learn about the use of the system. This section discusses some of the questions we may ask about Yenta use. Not all of these will be crucial in analyzing

the system, but many will help us in our goal of evaluating Yenta as a matchmaking system.

Some of these questions ask for a total for a single variable. Many seek to determine a correlation between two variables. Others track changes of one variable, or the correlation between two variables, over time. Three categories of queries are listed below: user preferences and habits; clustering; and messages, introductions, and attestations.

#### 4.1.1 User Preferences and Habits

- What changes did users tend to make to the standard configuration parameters on their agents? What values did users set for these parameters? The values of all user-settable parameters are reported for this reason.
- How often did users consult their Yentas? This can be inferred from the number of pages users fetched using their web browsers, and also by asking the users.
- Were the help and documentation pages helpful? We can see how often the documentation pages were requested.
- How many errors did Yentas encounter, and when? Errors are automatically logged, and bug reports sent.

#### 4.1.2 Clustering

- How soon did clusters tend to form and, once formed, how did they change over time? The timestamped event logs of when Yentas entered and left clusters will be helpful here.
- What sorts of clusters formed? How many clusters were there? We can only approximate the total number of clusters, since no Yenta has complete information, but we can make estimate based on how many clusters each Yenta sees and how many other Yentas it knows. Types of clusters are not reported, but are important and should be determined through interviews or other means.

- How many interests did the typical user have? Did most interests tend to get matched with other Yentas in clusters? These are directly reported for each Yenta.
- How many clusters did the typical Yenta belong to? This is directly reported.
- How soon after startup did Yentas tend to find clusters? How did this change over time? Again, this information can be inferred from event logs.

### 4.1.3 Messages, Introductions, and Attestations

- How many messages were sent and received? This can be computed directly from the counter totals.
- Did users tend to send messages more to individuals or to all Yentas in a cluster? We have counters for individual and cluster messages sent and received for each Yenta.
- Did the number of clusters a Yenta was in influence the number of messages (individual or cluster-wide) that the user sent through the Yenta system? Counters tell us the number of clusters and the number of messages sent for each Yenta; we can compare between Yentas and different values over time for the same Yenta.
- How many introductions were made? How many introductions did a typical user request or accept? There are counters for introductions requested, accepted, and refused.
- Did the age of a Yenta affect the number of introductions initiated by or offered to the Yenta? The earliest cryptographically signed date on a Yenta's attestations gives other users a lower bound on the age of that Yenta. The number of introductions and when they occurred can be determined from the counters and event logs.

- What sorts of attestations were made? What sorts of attestations were signed? Were most attestations signed? How many signatures did attestations tend to accumulate? We record all attestation strings and the number of signatures on each.
- Did the number of attestations a user made affect the number of introductions that were initiated by or offered to the Yenta? The number of attestations and the number of introductions for a Yenta are reported directly via counters.
- Did the number of signatures on a Yenta's attestations affect the number of introductions that were accepted or refused? The number of signatures and the number of introductions for a Yenta are recorded in counters.

## 4.2 Means

We do not need to wait long before looking at the collected data. We can start sorting and analyzing the data as soon as it starts appearing. At any time, we can copy over the log file, or start a new one, and begin to turn the text logs into meaningful statistics.

As discussed in Section 3.3, the logs will be written onto disk exactly as they are received by the server. It is efficient for writing, but the data is not of much use in this form. There will be a lot of data, and much of it is redundant: to a great extent, the counters and event logs describe the same things, though they are different enough that we do want to report both. Log entries can be parsed and stored in smaller formats, and the original logs compressed and archived. We need tools to convert the logs into readable form and help us make sense of them.

### 4.2.1 Parsing

Properly formed log entries will be in the form of Scheme lists, as described in Section 3.2. These entries can simply be read using Scheme's `read`.

If a log entry is malformed, due to a client timeout or to errors in log creation, transmission, or reception, we will still attempt to extract what data we can from it, since for our purposes some information is better than none.

Either way, we will wind up with a list consisting of (tag value) pairs. These will be parsed to form a statistics database, with tallies and tables separated by:

- Yenta (based on the logging ID), so we can make comparisons between Yentas. We will collect final counter values for each Yenta. We can also compile complete (allowing for missing entries), time-ordered event logs for each Yenta.
- Timestamp, so we can see differences over the lifespans of Yentas and of the Yenta project.
- Totals, so we can make overall comparisons and judge effectiveness.
- Other fields as we see fit.

### 4.2.2 Graphing

Graphs, if they are done well, are often better media than tables and tallies for conveying meaning in data [15]. Consequently, we will want to create graphs for each of the pairs (or triples) of variables we are correlating. Rather than construct graphs by hand, we will use tools to take the database, created as above, and selected variables, and produce graphs of the data.

## 4.3 Results

As of this writing, Yenta has not been released. Starting soon, it will be released internally to the MIT Media Lab, then to the MIT community on the Athena computing environment [8] and to beta testers, and finally to the public, via the Internet.

Once there is even a small user base reporting statistics, we can start gathering and analyzing data. We hope to answer many of the questions listed above, and



others as they are asked. More data is recorded than we expect to need, so we can be flexible about what is used.

From these quantitative analyses, and from interviews conducted with Yenta users, we will draw conclusions about the effectiveness and success of Yenta.

# Chapter 5

## Conclusion

### 5.1 Summary

This thesis demonstrates a central, privacy-protecting logging system for a decentralized system. To aid us in evaluating Yenta, a distributed, privacy-protecting matchmaking system, I have implemented a data collection system in which each running agent logs data to a central logging server. The data collection and logging system continues to protect user privacy, while giving us information about the state of the system as a whole. When Yenta is put into operation, the data collected will be analyzed to enable us to draw conclusions about the success of Yenta.

### 5.2 Future Work

Below are some possible extensions to the data collection system.

#### 5.2.1 Adjusting the Logging Frequency

As discussed in Section 3.2, we may want to adjust how frequently the Yenta agents log to the statistics server over the course of the research. For example, at first, when there are fewer Yentas and we are still debugging, we would like them to log more often; later, we will probably want each to log less often. Or we may simply find that

we are getting more or fewer log entries than we expected, and wish to change the timeout. Rather than have users download new copies of the agent code, we would like to have the server be able to send a new timeout value back to the client in the process of receiving a log entry.

### **5.2.2 Acknowledging Log Receipt**

In the current implementation of the logging system, the client closes the connection and presumes logging success once it has written the log entry to the output port. Although TCP makes it unlikely that the server will not receive the entry after this point, we should modify the server to acknowledge receipt of log entries, and the client to wait for acknowledgement before closing the connection. Especially if the server may already be sending an adjusted timeout back to the client, this is not a major addition.

### **5.2.3 Flushing Logs to Disk**

Currently the server ensures that logs get flushed to disk by closing the file after each write. If logging is frequent, this means that the server is doing a lot of file opens, closes, and seeks (since it is appending to the log file). To avoid having to close the file after every write, we can let the server keep the log file open all the time and use a separate watcher task that periodically flushes the file to disk.

### **5.2.4 An Analysis of the Reputation System**

There are a lot of privacy issues involved in logging anything to do with the attestations and associated signatures. We decided to record the attestations made by the Yentas because we wanted to see what sorts of attestations people made. We also recorded the number of signatures with each, so we could see what sorts of attestations tended to be signed. But we did not include other signature information.

It would be interesting to track the web of attestations and signatures, to see who said what and who signed what. Did the sorts of attestations people made affect

what they were willing to sign? How many different Yentas' attestations did a user sign? Which Yentas did a user receive signatures from?

This sort of tracking would have even more privacy issues, which are left for future work.

# Appendix A

## Counters

```
;;; YENTA COUNTERS

;; Operations
(def-yenta-counter *ctr:startups*
  "Startups"
  "Number of startups of this Yenta")

(def-yenta-counter *ctr:shutdowns*
  "Shutdowns"
  "Number of shutdowns of this Yenta")

(def-yenta-counter *ctr:uptime-minutes*
  "Uptime in minutes"
  "Number of minutes this Yenta has been in operation")

(def-yenta-counter *ctr:fatal-errors*
  "Fatal errors"
  "Number of fatal errors this Yenta has experienced")

(def-yenta-counter *ctr:log-count*
  "Number of log messages sent"
  "Keeps track of the number of log messages sent to the logging
  server.")

(def-yenta-counter *ctr:logging-errors*
  "Statistics-logging errors"
  "Number of errors this Yenta has had while logging statistics")
```

```

(def-yenta-counter *ctr:bug-reports*
  "Bug reports"
  "Number of bug reports submitted from this Yenta")

;; User Interface
(def-yenta-counter *ctr:pages-fetched*
  "Pages fetched"
  "Number of HTML pages fetched by this Yenta")

(def-yenta-counter *ctr:docs-fetched*
  "Documentation pages fetched"
  "Number of documentation pages fetched by this Yenta")

;; interYenta Communication
(def-yenta-counter *ctr:conns-initiated*
  "Connections initiated"
  "Number of interYenta connections this Yenta has initiated")

(def-yenta-counter *ctr:conns-served*
  "Connections served"
  "Number of interYenta connections this Yenta has answered")

(def-yenta-counter *ctr:opcodes-sent*
  "Opcodes sent"
  "Number of opcodes this Yenta has sent to other Yentas")

(def-yenta-counter *ctr:opcodes-received*
  "Opcodes received"
  "Number of opcodes this Yenta has received from other Yentas")

(def-yenta-counter *ctr:network-errors*
  "Network errors"
  "Number of network errors this Yenta has encountered during
  interYenta communication")

(def-yenta-counter *ctr:bytes-sent*
  "Bytes sent"
  "Total number of bytes this Yenta has sent to other Yentas")

(def-yenta-counter *ctr:bytes-received*
  "Bytes received"
  "Total number of bytes this Yenta has received from other Yentas")

(def-yenta-counter *ctr:auth-failures*
  "Authentication failures"

```

```

"Number of authentication failures this Yenta has had during
interYenta communication")

;; Interest finding
(def-yenta-counter *ctr:bytes-scanned*
  "Bytes scanned"
  "Number of bytes of user files this Yenta has read")

(def-yenta-counter *ctr:docs-scanned*
  "Documents scanned"
  "Number of documents (user files) this Yenta has read")

(def-yenta-counter *ctr:rescans*
  "Rescans"
  "Number of rescans of user documents this Yenta has performed")

;; Matchmaking
(def-yenta-counter *ctr:yentas-encountered*
  "Yentas encountered"
  "Number of other Yentas this Yenta has encountered (includes
  repeats)")

(def-yenta-counter *ctr:clusters-joined*
  "Clusters joined"
  "Number of clusters this Yenta has joined")

(def-yenta-counter *ctr:clusters-left*
  "Clusters left"
  "Number of clusters this Yenta has left")

(def-yenta-counter *ctr:intros-i-requested*
  "Introductions I requested"
  "Number of introductions this Yenta has requested")

(def-yenta-counter *ctr:intros-i-answered*
  "Introductions I answered"
  "Number of introductions this Yenta has responded to")

(def-yenta-counter *ctr:intros-i-refused*
  "Introductions I refused"
  "Number of introductions this Yenta has refused")

(def-yenta-counter *ctr:intros-requested-of-me*
  "Introductions requested of me"
  "Number of introductions that have been requested of Yenta")

```

```

(def-yenta-counter *ctr:intros-answered-me*
  "Introductions answered"
  "Number of introductions this Yenta has requested that were
  answered")

(def-yenta-counter *ctr:intros-refused-me*
  "Introductions refused me"
  "Number of introductions this Yenta has requested that were
  refused")

(def-yenta-counter *ctr:ind-messages-sent*
  "Individual messages sent"
  "Number of messages this Yenta has sent to other individual Yentas")

(def-yenta-counter *ctr:ind-bytes-sent*
  "Individual message bytes sent"
  "Number of bytes this Yenta has sent in messages to other individual
  Yentas")

(def-yenta-counter *ctr:ind-messages-received*
  "Individual messages received"
  "Number of individual messages this Yenta has received from other
  Yentas")

(def-yenta-counter *ctr:ind-bytes-received*
  "Individual message bytes received"
  "Number of bytes this Yenta has received in individual messages from
  other Yentas")

(def-yenta-counter *ctr:cluster-messages-sent*
  "Cluster messages sent"
  "Number of messages this Yenta has sent to Yenta clusters")

(def-yenta-counter *ctr:cluster-bytes-sent*
  "Cluster message bytes sent"
  "Number of bytes this Yenta has sent in messages to clusters")

(def-yenta-counter *ctr:cluster-messages-received*
  "Cluster messages received"
  "Number of cluster messages this Yenta has received")

(def-yenta-counter *ctr:cluster-bytes-received*
  "Cluster message bytes received"
  "Number of bytes this Yenta has received in cluster messages")

```



```
;; Attestations
(def-yenta-counter *ctr:atts-made*
  "Attestations made"
  "Number of attestations the user has made")

(def-yenta-counter *ctr:atts-fetched*
  "Attestations-fetched"
  "Number of times this Yenta's attestations were requested")

(def-yenta-counter *ctr:sigs-received*
  "Signatures-received"
  "Number of signatures this Yenta has received on its attestations")

(def-yenta-counter *ctr:sigs-made*
  "Signatures made"
  "Number of signatures this Yenta has made on other Yentas'
  attestations")

(def-yenta-counter *ctr:sigs-verified*
  "Signatures verified"
  "Number of signatures on attestations this Yenta has verified")
```

# Appendix B

## Events

The following events are logged:

- startup
- shutdown
- contact-with-new-yenta
- contact-with-known-yenta
- exchange-of-cluster-info
- referral-initiated
- referral-granted
- cluster-entered
- cluster-left
- introduction-initiated
- introduction-granted
- introduction-refused
- attestation-created
- attestation-signed
- attestation-fetched

# Appendix C

## Sample Log Entry

A possible log entry as recorded by the logging server.

```
(894484313 ((:id "Y^BcV*\345\201\336") (:time 894484313) (:counters (
(*ctr:sigs-verified* 1) (*ctr:sigs-made* 2) (*ctr:sigs-received* 3) (*
ctr:atts-fetched* 2) (*ctr:atts-made* 2) (*ctr:cluster-bytes-received*
 4825) (*ctr:cluster-messages-received* 6) (*ctr:cluster-bytes-sent* 2
36) (*ctr:cluster-messages-sent* 1) (*ctr:ind-bytes-received* 0) (*ctr
:ind-messages-received* 0) (*ctr:ind-bytes-sent* 1351) (*ctr:ind-messa
ges-sent* 1) (*ctr:intros-refused-me* 0) (*ctr:intros-answered-me* 1)
(*ctr:intros-requested-of-me* 0) (*ctr:intros-i-refused* 0) (*ctr:intr
os-i-answered* 0) (*ctr:intros-i-requested* 1) (*ctr:clusters-left* 0)
(*ctr:clusters-joined* 3) (*ctr:yentas-encountered* 17) (*ctr:rescans
* 6) (*ctr:docs-scanned* 324) (*ctr:bytes-scanned* 35629) (*ctr:auth-f
ailures* 0) (*ctr:bytes-received* 542) (*ctr:bytes-sent* 676) (*ctr:ne
twork-errors* 0) (*ctr:opcodes-received* 52) (*ctr:opcodes-sent* 50) (
*ctr:conns-served* 23) (*ctr:conns-initiated* 16) (*ctr:docs-fetched*
3) (*ctr:pages-fetched* 24) (*ctr:bug-reports* 0) (*ctr:logging-errors
* 1) (*ctr:log-count* 7) (*ctr:fatal-errors* 0) (*ctr:uptime-minutes*
9074) (*ctr:shutdowns* 2) (*ctr:startups* 3))) (:settings ((:interests
:follow-threshold* 0.15) (*interests:match-threshold* 0.2))) (:interes
ts 4) (:clusters (23 2 14)) (:conns 0) (:attestations (("my public key
's fingerprint is D0:78:D7:29:C9:E6:71:69:7D:10:6B:AF:49:4A:92:B9." .
1) ("I am a doctor." . 2)) (:events ((894484282 :startup))))))
```

# Bibliography

- [1] Clinger, William, and Rees, Jonathan, ed. "Revised<sup>4</sup> Report on the Algorithmic Language Scheme." 1991. <http://www-swiss.ai.mit.edu/~jaffer/r4rs.toc.html>
- [2] Export Administration Regulations("EAR"), 15 C.F.R. pts. 730-74.
- [3] Fielding, et al. "Hypertext Transfer Protocol – HTTP/1.1.," RFC2068. January 1997. <ftp://ftp.isi.edu/in-notes/rfc2068.txt>
- [4] Foner, Leonard N. "A Distributed, Privacy-Protecting Matchmaking System." Doctoral Thesis Proposal, MIT Media Arts and Sciences Program. November 1997.
- [5] Foner, Leonard N. "A Security Architecture for Multi-Agent Matchmaking." *The Second International Conference on Multi-Agent Systems*, Kansai Science City, Japan, December 1996.
- [6] Foner, Leonard N. "Yenta: A Multi-Agent Referral-Based Matchmaking System." *First International Conference on Autonomous Agents (Agents '97)*. Marina del Rey, California, February 1997.
- [7] Jaffer, Aubrey. "SCM Home Page." May 8, 1998. <http://www-swiss.ai.mit.edu/jaffer/SCM.html>
- [8] "Getting Started on Athena" MIT Athena On-Line Help (OLH) Home Page, <http://web.mit.edu/olh/>
- [9] *International Traffic in Arms Regulations*, 58 Federal Register 39,280 (1993) (to be codified at 22 C.F.R.§§120-128, 130).
- [10] Miles, Matthew B. & Huberman, A. Michael. *Qualitative Data Analysis: An Expanded Sourcebook*. Thousand Oaks, California: SAGE Publications, Inc., 1994.
- [11] Postel, J. "Transmission Control Protocol," RFC793. September 1981. <ftp://ftp.isi.edu/in-notes/rfc793.txt>

- [12] Rhodes, Bradley, "The Wearable Remembrance Agent: A system for augmented memory." *The Proceedings of The First International Symposium on Wearable Computers (ISWC '97)*. Cambridge, Mass., October 1997, pp. 123-128.
- [13] Schneier, Bruce. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, second edition. John Wiley & Sons, 1996.
- [14] SSL 3.0 Specification. <http://home.netscape.com/eng/ssl3/index.html>
- [15] Tufte, Edward. *The Visual Display of Quantitative Information*. Graphics Press, 1992.
- [16] Young, Eric A. SSLeay information, CryptSoft Pty Ltd home page. <http://www.cryptsoft.com/>