CHAPTER 5 *Evaluation*

5.1 Introduction

This research proposes an architecture with political ramifications, and a sample application that demonstrates how such an architecture can be used. What are suitable metrics for evaluating it? The space of possible evaluation strategies, and the questions that could potentially be asked, is quite large. In this section, we shall whittle the problem down a bit. Later sections will cover:

- Simulation results of Yenta's network clustering algorithm — Section 5.2
- How can we collect data from running Yentas? — Section 5.3
- What data is currently collected? — Section 5.4
- What are some of the questions we can answer? — Section 5.5
- How can we evaluate Yenta's security? — Section 5.6
- A risk analysis of the architecture and the fielded system — Section 5.7
- Other applications suited to this architecture — Section 5.8
- Ideas on motivating businesses to use this technology — Section 5.9
- Future work — Section 5.10

First off, we aim to show that one can design an architecture which can *protect*, yet still *use*, personal information when implementing applications with certain characteristics. The architecture, and the types of applications for which it is suitable, was described in Chapter 2, and the particular application used to investigate the architecture was described in Chapter 4. We have claimed that this architecture is an advance over traditional methods of handling this information for the same types of applications.

The eventual goal of this research is to encourage system designers to change the way they design systems—in particular, to start from a social agenda and design forward from that, rather than ignoring such an agenda or assuming that it will hinder building systems that people can use for useful tasks. This is an essentially political motivation which attempts to give users systems that are more robust against failures and more likely to protect their rights. Actually observing such a change, however, involves a long timescale—we would be dependent upon finding some system which *was* going to be designed in a centralized, non-privacy-preserving fashion, but which is *now*

going to be designed differently, because of the research described here. Such designs take time, and often happen in with little public disclosure.

As discussed here, therefore, we are *not* aiming to show that the architecture has already made an impact, nor are we aiming to show that the sample application—Yenta—has or will be used sufficiently for the its political impact to be felt. That is beyond the scope of this dissertation—but certainly not beyond the scope of the political agenda that motivates the work. Indeed, one of the hopes in this research is that it will *contribute to the discourse* surrounding the design and implementation of such systems.

Therefore, evaluation focuses on *technical implementability* of the ideas involved. The sorts of questions we will answer are of the form: *Can Yenta be implemented? Do Yentas cluster? Does it help users? Does it appear sufficiently secure?*

A fascinating next step, after answering such questions, would be to investigate how users actually *use* Yenta. For example, the reputation system is likely to generate a set of social conventions, and it is not clear what those will be, and how those will change over time. Such sociological study is also outside our scope here, although the author does hope to do such investigation in the future.

Note that, in investigating Yenta's performance, we make no strong claims here about *optimality.* One could ask a variety of questions about Yenta's clustering methods, either those used to cluster documents within a single Yenta, or the way in which Yentas form clusters on the Internet. What's inappropriate about asking questions about optimality?

- There is no known metric for determining what optimal *means* when deciding whether or not two humans share an interest in a subject, nor in what it means for them to have similar-looking documents. One can invent a large number of definitions, but it's not clear whether this is a useful exercise.
- Optimal solutions usually take a long time to converge; most such problems are NP-complete. Real-world systems always change faster than can be accommodated by such slow methods. In such systems, it is always better to be *acceptably fast*—and approximately correct—than *unreasonably slow*—and perfect.

This lack of concern about optimality is one of many reasons why we make no claims that Savant, which turns documents into keyword vectors, advances the state of the art in information retrieval. Nor do we claim that the algorithm Yenta uses to cluster the resulting vectors—before it contacts other Yentas—into user interests is necessarily an advance, either. In both cases, they are simply *sufficient* to make Yenta useful.

In evaluating results from fielded Yentas, there are a large number of questions we could ask. We shall restrict ourselves to a small set here, but also demonstrate how a large number of different questions could be answered with the infrastructure that is available.

It is important also to keep in mind what we are investigating. We are looking at the *use* of a *particular implementation* of a *single application* that is the exemplar of a *general architecture*. This can answer certain questions, like whether the architecture works at all for *any* application, but also does not answer many others, such as how Yenta might be used differently if it had a slightly different mix of features, or ran on non-UNIX platforms, and so forth.

5.2 Simulation results

We first turn out attention to some simulation results for the clustering algorithm that Yentas use among themselves. This algorithm was described in depth in Section 2.8, with some additional details about the implications of this algorithm in Section 2.9.

Many of these results were also reported in [61] and [58]. (In addition, even earlier results of document clustering—e.g., within a single Yenta—using an older version of the clustering algorithm (not described here) and SMART [188] as the comparison engine, appeared in [56].)

Up to 1000 Yentas were simulated

The Yenta clustering algorithm was simulated for various numbers of interests, typical sizes of its rumor cache, and up to 1000 Yentas, and showed good performance and convergence. Graphical results of these simulations are presented in Figure 17, which have been excerpted from several animations produced to study Yenta's clustering behavior. We discuss the results below.

Three different simulations

Three different simulations are presented. For each, the format of presentation is identical. Each simulation is shown as a series of images taken at various timesteps. The final state of any given simulation is the large image on the right; the six smaller images to the left of that image represent earlier stages of the simulation, reading from left to right and top to bottom.

Interpreting the displays

Each Yenta in any given simulation was given a random interest from the total number of interests available, and then the size of its cluster cache was examined at each simulation step, which indicates how successful it has been at finding other Yentas which share its interest. For all Yentas that share the same pair of parameter values—for example, rumor cache size versus number of Yentas for the first simulation—and are hence in the same bar of the display, the size of their cluster caches were averaged. This average is then compared to the total number of Yentas that *could* have conceivably been in the cluster cache (if all Yentas sharing the interest had been found), and that ratio is expressed as the percentage height of the bar.

Varying the rumor cache

The first simulation shows the effect of varying the size of the rumor cache for up to 1000 Yentas, given 30 different interests split amongst the Yentas. Roughly speaking, it shows that the size of the rumor cache does not make much difference in the speed of cluster formation for more than around 400 Yentas.

Varying the number of interests

The second simulation varies the number of possible interests shared amongst the Yentas with the total number of Yentas, given a rumor cache size of 50. As might be expected, it takes longer to find all the other Yentas one would want as the number of interests increases, or as the number of total Yentas increases.

Ratio of rumor cache to interests

Finally, the third simulation shows the effect of varying the size of the rumor cache for various numbers of interests, given 1000 Yentas. This seems to show that a rumor cache size of 15 is enough for small numbers of interests—between 10 and 30—and that raising this size beyond 35, even for large numbers of interests, does not buy us much.

The basic clustering works

These are strong results. They show that Yenta's clustering behavior is *stable*—Yentas do not try forever to find each other, nor do clusters of them break apart for no good reason—and at least acceptably *efficient*—the number of messages exchanged in order to cluster most of the Yentas is not unreasonable.

5.3 Collecting data from Yenta

Let us now turn to actual application. The general architecture described in Section 2.13 shows how to return data from running Yentas in such a way that it may be analyzed. In summary, the approach is to:

- Run a central server, at an address known to all Yentas, which can collect the data.
- Have each Yenta transmit certain statistical data to the central server, making sure to:
  - Blind the data before transmission by stripping out identifying information
  - Include a per-Yenta unique random number in the data so successive log entries

from the same Yenta may be correlated
    - Encrypt all data during transmission
- Write all the data to disk for later analysis

Collecting this sort of statistical information has considerable risks. If done incorrectly, it could jeopardize users' privacy, and their trust in the entire system, as well as enabling as a single point of attack for a malicious intruder. Therefore, let us follow the steps above, starting with transmission and ending with reception of the data, in order to demonstrate that the collection of this data is not a serious threat.

Central receiver

Each running Yenta knows the address of the central statistics server, and has a task which periodically collects certain information (see Section 5.4 below), creates a logging record, and sends that record to the server. A user's Yenta also sends this information immediately if its user instructs it to shut down.

Blinded data

The information sent is carefully blinded. For example, Yenta by default creates an attestation identifying the user's Yenta by its Yenta-ID, which users may get signed like any other attestation. This attestation is carefully removed from the logging data, before transmission, since otherwise its presence in the data would identify exactly which Yenta logged this record. For more details on the sort of data being logged and why it should be safe, see Section 5.4.

In addition, the statistics-ID which identifies the Yenta doing the logging is a 64-bit random number, having no connection to any other identifier in Yenta. It is communicated only to the logging server—not to other Yentas—and has no personal information embedded in it. Once the data has been written to disk, there is no record of which IP address logged this record, and hence no backpointer to identify where this data came from. All we can know is whether the same Yenta later updates it.

Encrypted transmission

Data being sent to the logging server is encrypted using a session key, in a very similar manner to the way in which Yenta saves its state to disk (see Section 4.8.2). This session key is randomly generated before each attempt to log, and is never reused. A preamble is sent before the actual record consisting of this session key, encrypted with the public key of the logging receiver, which is known by all Yentas. Since only the server knows the private key, only the server may decrypt the session key and thus decrypt the data.

Each individual Yenta keeps track of whether the logging receiver claimed that the logging record was successfully received. If the receiver appears to be down, Yenta simply abandons the attempt to log, remembers that is has done so, and tries again later. This keeps the receiver from potentially being a central bottleneck, whose failure could inhibit the normal operation of all Yentas everywhere. In addition, Yentas periodically prune their logging information if it is too old—this means that, if the receiver vanishes permanently, each Yenta does not store a monotonically-increasing amount of pending logging information.

Vulnerabilities

At the moment, the server decrypts the received data before writing it to disk. It would be slightly safer to leave the data encrypted instead—this would mean that even the server need not know the private key, which could be kept offline and used only when the data is being decrypted for analysis. This complicates analysis, but is being considered. Since all data logged is theoretically already safe—unlikely to compromise users' privacy—the marginal utility of including this step is dubious.

If the server's private key is revealed—by a cracker, say—it will only be useful in attacking a user's privacy if the person who knows the key already has access to the traffic from the Yenta which is the target. After the traffic has arrived at the server is too late—while an attacker who could read the disk could read arbitrary logging

records written there, he or she would have no way to know which machine that Yenta was running on.

Don't do this if you can avoid it

There is no question that running this central statistics receiver is a potentially large invasion of user privacy, and that it presents an inviting target for attack. Any non-research implementation of this architecture should *not* be running such a server. Further, users are well-advised to carefully vet the operation of any architecture which employs such a server—one of many good reasons for ensuring that the source code for any such application is open to public inspection. While the design of the logger was carefully constrained to use only data that appeared safe, it still presents risks.

5.4 What data is collected?

Yenta keeps track of two main classes of things for the benefit of the statistics receiver:

Events

- *Events*. These are changes of state, generally caused by some external action—such as a request by the user for a web page from the user interface, or an incoming connection by some other Yenta. Some events are internally generated, such as a timer expiring indicating that Yenta should rescan the user's documents.

Summaries

- *Summaries* of certain internal state. These are generated on-the-fly, when Yenta has determined that it is time to write a log entry, and are typically estimates of the size of internal data structures.

Counters

In general, any given event will increment a *counter* which keeps track of the number of times which this event has occurred. Some events—such an impending user-commanded shutdown—also cause logging to happen immediately.

The data is persistent

All counters and event logs are maintained in Yenta's permanent state, and are regularly checkpointed to disk. This means that any event which fails to be logged before Yenta is shut down will be logged the next time Yenta is restarted; similarly, counters such as the total number of minutes this Yenta has been in operation will accumulate across successive runs.

A single logging record

An actual logging record thus consists of the following information:

- The statistics-ID (see Section 5.3).
- The time of the message, in Universal Coordinated Time (UTC).
- The current values of all counters.
- The values of all user-settable parameters. These include the various thresholds and preferences the user has set through the interface.
- All attestation strings, and the number of signatures on each attestation. Note that the attestations are stripped of the Yenta-ID normally attached to them, and the signatures themselves are not sent, only a count of them.
- The number of interests known for this user. This is computed from the data structure that keeps track of the Yenta's interests. Note that this is a simple count, *not* the interests themselves.
- The number of clusters this Yenta is in, and the approximate number of known Yentas in each computed in part from cluster cache information. This is not necessarily the total number of interests that this Yenta knows about, since not every interest may have had a cluster found for it yet.
- The number of currently-open network connections to other Yentas. Since a connection is only open when two Yentas have something to say to each other—not simply because they know of each other's presence—this is more an indication of the instantaneous load being placed on the network by this Yenta than it is of how many clusters it is in. (How many network connections have been opened in the past, how many referrals have been done, and so forth, are found in the various counter values mentioned above.)

- A list of events which have transpired since the last log transmission.

Watching one Yenta over time

To actually determine how a particular Yenta has changed over time—such as how rapidly it manages to find clusters for its user's interests—successive records from the same statistics-ID are compared, along with the timestamp logged with each.

A sampling of the sort of counter data which may be collected includes:

- System operations counters: number of startups, shutdowns, errors, bug reports, and time in minutes that this Yenta has been running.
- User interface counters: number of pages and documentation pages fetched.
- Inter-Yenta communication and network statistics: connections initiated and served, protocol opcodes sent and received, network errors, and authentication failures.
- Document clustering 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.

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

Clearly, this is a great deal of potential data. We shall examine only a very small subset of it below.

5.5 A sample of results

To evaluate Yenta's performance in the field, a pilot study was undertaken in which Yenta was advertised to a small group of MIT users. This pilot study was deliberately restricted to a relatively small audience, and Yenta's availability was not advertised to a wider audience. The primary reason concerns the implementation of Savant currently present in Yenta—this version of Savant does not have logic to recognize and reject many common artifacts in electronic mail messages, such as included header fields, PGP signature blocks, URL's, and so forth. Thus, it tends to falsely cluster messages based on these machine-generated elements, as well as on their actual content as understood by users. This means that, in addition to clusters that most users would deem useful, there were a large number of clusters which were unhelpful.

A new version of Savant that does not have these disadvantages was made available shortly before this analysis, but not soon enough to facilitate its incorporation into Yenta. Doing such an integration will also change Yenta's understanding of clusters and is not a backwards-compatible change; hence, users in the field will be inconvenienced by having their existing clusters disrupted unless great care is taken. Operational concerns such as this have therefore encouraged only a small deployment at

present so as to minimize disruption of an existing user base. Once the new Savant is integrated, Yenta will leave pilot status and become available to a much wider audience, and will be advertised to such audiences.

5.5.1 Qualitative results

We now turn out attention to some of the available results. This data is derived from the pilot study, in which no more than 50 Yentas were operational at any given time. Exactly how many Yentas are in operation at any instant is somewhat uncertain, for several reasons. We can only know how many Yentas have run recently by investigating the statistics logs, which are keyed by the unique SID. Thus, we must determine whether any given Yenta is still in operation by waiting to see if it continues to log data. In most environments, this would be easy to see, but the particular environment to which Yenta was deployed in the pilot—MIT's Project Athena—tends to encourage users to shut down Yenta frequently, since users rarely have a workstation of their own available and must instead use public ones, which kill background tasks when the user logs out. Finally, the existing Savant implementation in Yenta tends to accumulate too much machine-generated data from email messages. Users tended to discard entire databases and start over on different collections of files when trying to determine which files would best reflect their interests. Since Yenta was not designed to discard its entire database in this fashion, its users took deleting Yenta's saved state and initializing brand-new copies of Yenta, hence artificially inflating the generated statistics. In the analysis that follows, Yentas that do not appear to have run recently have been omitted as having been started briefly and discarded in favor of a new run as a brand-new Yenta. This will be less of a problem with the newer Savant; also, providing users with easier ways to tune Yenta's initial selection of files will help.

After pruning the data for various artifacts such as these, and to reduce the analysis task somewhat, we were left with a sample size of 21 Yentas. This sample will be used in the discussion that follows.

Clustering works

In general, results from fielded Yentas bear out the simulation results in Section 5.2. For example, Yentas will cluster correctly if they share sufficiently-close interests, and, likewise, they will correctly conclude that they should *not* cluster if their interests are divergent. This is the case despite the technique, as described in Chapter 2.8.3, of mixing in other data from the local Yenta's rumor cache to provide plausible deniability for its user to a querying Yenta.

Yentas can find each other

Yentas have demonstrated that they can find each other in all the ways designed into the architecture—via the bootstrap server, via broadcast on the local Ethernet segment, and by detecting the presence of a formerly-unknown Yenta from the contents of some other Yenta's rumor- or cluster caches.

The protocol works

Further, the running Yentas do not display serious protocol abnormalities—any given pair of Yentas that were formerly unknown to each other initiates a conversation, dumps interests back and forth, and correctly clusters, or not, based on those. They do not get hung up exchanging data forever, and correctly revisit each other at various intervals to see if anything has changed.

Message and attestation relaying works

Yentas which share an interest can correctly relay messages back and forth to each other. This behavior was verified both in one-to-one messaging and in one-to-cluster situations. Similarly, attestations may be created, signed, and displayed to Yenta's users.

Determining the user's initial interests has an obvious path to improvement

Yenta's determination of user interests was judged subjectively by investigating the interests that it found from a variety of files. In the currently-fielded Yenta, its determination was sufficient, but not as good as it can be. In large part, this is due to its use of an older version of the Savant comparison engine, as detailed above.

Saving state works

Individual Yentas can correctly save and restore their state across crashes—either of Yenta or the underlying machine—and across user-commanded shutdowns. They have also been shown to interoperate with a variety of common web browsers.

Yenta is fast enough

Yenta has also proven to be acceptably fast. Even running on five-year-old hardware (an HP 9000/725), it can scan and cluster several megabytes of mail—about as much as is reasonable to use—in a handful of minutes. A typical clustering attempt with another Yenta, in which both Yentas must share not only their interests but many interests from their rumor caches for plausible deniability (see Section 2.8.3), takes a few minutes. In part, this is due to the throttling effect of the network, but it is also the case that we do not wish Yenta to consume all available CPU resources on the machine on which it runs—after all, it runs as a background task most of the time.

Since Yenta is designed to run with only occasional user attention, even these results are better than they appear. For example, even though it takes a few minutes for two Yentas to determine whether or not they share an interest, the user can still fetch pages from the user interface, talk to other Yentas already known to be a shared cluster, and so forth.

Logging errors helps a lot

Handling internal errors and reporting them to the statistics server was very useful in the field. The very first deployment of Yenta to users turned up a number of minor bugs, generally caused because users tended to use Yenta slightly differently than its implementors sometimes did—that caused tasks to occasionally err. The symptom of such a failure is generally that the user sees a page request of the user interface simply hang until it is retried; this starts a new task, and generally whatever bug was encountered would not be retriggered. However, because such failures were reported to the statistics server, complete with backtraces, tracking down the bugs and fixing them was *much* simpler than it would have been had self-reports from the field been the only method.

5.5.2 Quantitative results

To lend some concreteness to the discussion above, let us examine just a few selected statistics from those logged by Yentas in the field. These statistics cover 21 Yentas deemed representative, logged over a period of about 25 days, from the pilot study. They are drawn from approximately 2200 individual entries to the statistics logger.

The table in Figure 16 below summarizes the results. We investigated a few elements from several different areas of Yenta's operation: how the user interface was used; how many documents were scanned and how many interests were determined as a result; how the attestation system was used; some clustering data; a quick look at Yenta's networking protocol, and how long Yentas tended to run. For each such element, we present the total across the *n=21* Yentas, the minimum and maximum values seen, and their average and standard deviation. Many of the minimum values are zero, generally due to a Yenta being started, minimally configured, and then shut down without rerunning it later. Approximately 3 Yentas from the sample below show a short enough total runtime that this is likely for them, but their results were included in the totals because there was still useful data—such as number of documents scanned and number of interests found—from such Yentas, even though they were not allowed to continue running long enough to do anything useful for their users.

The statistics above show that users made extensive use of the UI—in other words, they interacted a lot with their Yentas. They also fetched a large number of help pages, which is to be expected of a new application. One user scanned a very large number of documents (over 8000), although must scanned a must more reasonable number (the median was around 400, and the average around 600). From these, users were typically presented dozens to a hundred or so interests, and tended to find at least a few other clusters to join. A typical Yenta sent a bit more than a megabyte—spread

Parameter	Sum	Min	Max	Average	Std dev
UI pages fetched	2925	11	648	139.2	143.1
Help pages fetched	264	1	32	12.5	6.9
Documents scanned	12773	0	8388	608.2	1779.5
Number of interests	2592	0	1406	123.4	312.9
Signatures verified	353	0	75	16.8	22.5
Clusters joined	89	0	50	4.2	10.9
IY opcodes sent	3032	0	811	144.3	216.2
IY kilobytes sent	28854	0	10770	1374.3	2715.9
Minutes of operation	117719	0	34122	5605.7	10287.1

Figure 16: Some selected statistics from fielded Yentas.

out over the three weeks of the pilot—to accomplish this level of clustering. Finally, any given Yenta typically accumulated around 100 hours of operation in this interval—being generally shut down when its user was logged out, for those using MIT Project Athena machines—although some ran almost the entire time and are still running as of this writing.

5.6 Security

Let us now turn to evaluating Yenta's security. It is widely accepted that there is no way to be absolutely sure that any particular piece of software, if at all complicated, is completely secure. However, there are many potential ways to increase our confidence, which include, among others:

- *Black-box analysis.* This involves attempting to crack Yenta's security completely from the outside, as if it was a black box.
- *Formal methods.* These involve proving theorems about the underlying cryptographic operators, *and* about how they are used in Yenta's actual implementation.
- *Design review.* This involves examining the overall principles of the architecture advanced in Chapter 2 and Chapter 3, and combining that with the description in Chapter 4 of the actual application fielded.
- *Code review.* This involves actually reading the code and looking for weaknesses.

While it is certainly *possible* that someone will subject Yenta to black-box analysis, we have no intention of doing so here; there seem to be much better options at our disposal. And, unfortunately, formal methods are quite attractive, but typically are not feasible for entire applications. They can be quite helpful in evaluating particular network protocols (such as SSL) or particular cryptographic functions (such as DES), but are less likely to reveal whether a particular application correctly implements the design which has been formally analyzed, due to the time and effort required to do rigorous analysis of a large body of code. They can also miss incorrect design assumptions, such as incompleteness of the threat model.

Yenta's design, and the design of the architecture of which it is a part, are public information. This encourages review. In addition, the actual source code of Yenta is also available, for a number of reasons, including pedagogy, increasing the portability of the application, and the presumption that openly-available code is itself a social good. However, one of the most compelling reasons to make code for an application such as Yenta public is to increase the chances that others will find weaknesses.

The strategy chosen for Yenta is twofold:

- Make it easy to vet Yenta's code
- Give people incentives to do so

The first of these is partially accomplished by Yvette, as described in Section 3.4.4. Briefly, Yvette encourages collaboration among people who are interested in evaluating a large body of code, by enabling them to divide up the work, write reviews of small sections, and review the work of others. Yvette also enables those who are less skilled to nonetheless peruse the reviews, by showing how much of the entire corpus of code has been reviewed and, for sections that have received several reviews, whether those reviews have been generally positive or negative.

There are several possible incentives for others to review Yenta's code. Making Yenta more secure is clearly a social good, at least among those reviewers who share the author's political agenda. Further, as is commonly the case in software projects whose source is publicly available, those who make particularly important contributions either to the code or its review are often rewarded by improvements to their reputation in that social group.

Yenta also tries directly to appeal to other programmers for review. The following rather long insert is an excerpt from the web pages which announce Yenta, and is indicative of the sort of things we are asking others to look for:

> **Please help improve Yenta's security, so that all of its users may benefit.** We are offering incentives for finding major flaws. To be most helpful to us, and hence to do the most to improve Yenta's security, please read *all* of the topics below. They cover:
>
> - How to comment on the code.
> - What's in it for you.
> - What counts as a flaw.
>
> **Commenting on the source code**
>
> Your easiest starting point is probably to *critique Yenta's source code directly.* Yenta's current source code is available via Yvette, which allows *collaborative critique* of a body of code: each person may make comments on a single function, a whole file, or an entire subtree of the source, and others may view these comments. This allows dividing up the work.
>
> Since it is expected that most possible flaws will concern some well-defined area of the source code, you should remark on it at the appropriate point in the source tree that Yvette gives you. *If you think you have found something particularly serious*, you may want to send mail to bug-yenta@media.mit.edu telling us what you found. Please see also our description of what counts as a flaw.
>
> **What incentives we have for you**
>
> There are several incentives available to encourage people to improve Yenta's security:
>
> • **Community good.** This is worth doing for its own sake, because you are helping everyone who uses Yenta to be able to use a system that will not inadvertently expose personal information, will not crash, and will be useful to its users.
>
> • **Public recognition.** All comments about Yenta that have been given to Yvette are available to everyone to read. Particularly insightful comments may also be mentioned in various acknowledgments when papers about Yenta are published.
>
> • **Goodies.** If you are the *first* to report a particularly serious security problem in Yenta, we'll give you something. If you're local, this might be dinner. If you're not

local, it might be something else appropriate. If you care strongly about getting something, then please *comment in Yvette* (if there is a particular area of the code that is affected), *and* remember to send us mail. Please note that our judgment of what counts as "serious" is absolutely at our discretion. But don't worry—we'll be fair. It is quite probable that there are things missing from the description below (perhaps we forgot one of the cases that don't count in the threat model description); this doesn't mean we owe you dinner if you find something we don't think it a major flaw, but which we didn't mention. On the other hand, we'd still like to hear about it—if nothing else, to correct our description.

What counts as a flaw?

This is a description of our **threat model.** In other words, what sorts of flaws are we looking for?

Security bugs versus other bugs

• **We're interested in *all* bugs...** so please, if you spot something in the source which is a bug in *functionality*, even if it does not have security implications, please comment about it in the source and also send us mail at bug-yenta@media.mit.edu. If you trip over a bug while using Yenta, but don't know where it is in the source, send us mail and at least let us know.

• **...but we're most interested here in *security* bugs.** Not only are undetected security bugs dangerous to users, but they are likely to go unreported unless someone actively looks for them. After all, a bug in functionality, such as Yenta crashing, or doing the wrong thing with a command, is likely to be noticed by the user who experiences it, but a security bug could be totally silent and yet deprive all users of their privacy.

What sort of attacks are we talking about?

• *Things which don't count.*

  • **Denial of service doesn't count.** In other words, if someone can arrange to make your Yenta do *nothing*, either by overloading it, running it out of resources, or attacking the connection of its machine to the net, that's outside of the scope of what Yenta is designed to survive. Of course, if you see a simple way to prevent a denial of service which is *specific to Yenta*, please let us know.

  • **Careless users don't count.** Users who deliberately choose poor passphrases will compromise their own security. Yenta can't stop them. Similarly, even though Yenta takes care to arrange a very strong SSL connection between the user's browser and Yenta itself, if the user is running their web browser with an insecure connection *between their keyboard and their web browser*, Yenta cannot possibly know this, and cannot prevent it from occurring. This can easily happen if the user is using X—with the keyboard and screen on one machine, and Yenta running another—and is not using SSH or some similar protocol between the two machines. Similarly, if the user is running a crippled browser that supports only 40-bit session keys, Yenta *is* willing to talk to the browser, but this connection is only secure against attackers without many resources.

  • **Attacks by root on the same machine don't count.** A superuser on someone's workstation can read any bit of memory, can substitute compromised versions of binaries for formerly good ones, can install trojan horses that capture every keystroke the user types before it gets to any application, and so forth. Yenta cannot hope to avoid such attacks. Note in particular that Yenta is *more* vulnerable to a memory-sniffing attack than programs like PGP, because Yenta must remember the user's private key at all times—PGP need only remember it for the instant that the session key is being encrypted. And any attack that compromises the binary—whether on the local workstation, or by altering NFS data if the binary is fetched over the network—also cannot be countered.

  • **Byzantine failures don't count.** In other words, if you surrounded some in-

nocent victim's machines with *only* machines running bogus, compromised versions of Yenta that are all under your control, you could certainly figure out what the user is interested in, and probably do a lot worse damage as well. Yenta explicitly assumes that all the rest of the Yentas on the net are *not* evil. One or two is okay, but a vast majority is not.

• **Savant index files don't count.** Yenta stores its crunched, vectorized information about your mail in a binary but unencrypted form on disk, in your ~/.Yenta directory. Although the directory is read-protected against all but its owner, this is not secure against an attacker who can read the filesystem. Since this information originally came from plaintext files which are *also* in the filesystem, it is assumed that this approach does not compromise the user's privacy any more than it already was. Note that Yenta's other saved state, such as the user's private key, his stored conversations, and so forth, *are* encrypted and never appear on disk in the clear, even for a moment.

• **Attacks on the maintainers' machines don't count.** Even though the distributions are cryptographically signed, and even though the source code is available via Yvette, there is certainly the potential for corrupting the actual code being distributed, by attacking the machines upon which Yenta is developed. While it would be possible to secure these machines better, doing so gets in the way of getting work done, and Yenta *is* a research project. So you are not allowed to attack the actual source—*or* our machines!—and then claim a victory. Just don't.

• **Traffic analysis doesn't count—yet.** The current version of Yenta uses point-to-point IP connections when passing a message from one Yenta to another. Later releases will employ a broadcast-flood algorithm, either by default or on request, to make it harder to tell where the real endpoints are of a communication. This makes it more difficult for an attacker who cannot monitor every link in real time to know which pairs of Yentas are exchanging a lot of traffic (and hence which may have users who are interested in the same things).

• *Things which do count.*

• **Problems in Yenta's cryptography.** This could be insecure encryption modes, vulnerabilities in the protocols used between Yentas or in the way that permanent state is stored on disk, and so forth.

• **User confusion that leads to exposure.** If Yenta does something that causes a user to be a confused and inadvertently reveal something that he did not wish to, that is a bug. But this must be *Yenta's* doing—not a con game perpetrated by another user, for example.

• **Failures of anonymity.** Yenta tries to keep the connection between an individual user's true identity and his Yenta identity unknown, unless the user deliberately divulges that information. If there are easy ways to defeat this, we need to know. However, see the note about traffic analysis *not* counting, so far—a later release will fix this.

• **Spoofing.** If one user can masquerade as another, complete with valid-looking attestations which are fraudulent, this is certainly a bug.

• **Missing items from the description on this page.** We may be missing examples, either in the listing above of threats which Yenta is just not designed to handle, or in this listing of possible places to look for problems. If so, please let us know, so we can update the list. This helps two sets of people: Yenta's users, who get a more accurate picture of what Yenta can and cannot do, and Yenta's reviewers, who won't waste their time investigating a vulnerability which we consider to be outside of Yenta's scope.

5.7 Risk analysis

Let us now turn our attention to an analysis of Yenta's residual risks, using the criteria in the previous section as guidelines. Where are the weak links? If Yenta is to be trusted, is it actually trustworthy?

5.7.1 Denial of service

Certainly the most obvious weak link in the design is that of *denial of service*. We have explicitly stated that this is *not* a problem we are trying to address, but how well do we sidestep it anyway? Let us first ignore any denial of service which takes down the actual host upon which an individual Yenta is running on, or its network connections anywhere else. We shall also assume at the moment that we are talking about an attack on a *single user's* Yenta—not on all Yentas. Assuming that the underlying host and its network are functional, how vulnerable is Yenta to a denial-of-service attack?

Single Yentas

There are several ways to mount such an attack. One involves simply opening a connection to a Yenta and giving it an infinite list of possible interests, or making one particular interest of infinite length. Yenta throttles its reception of any network connection to a fixed maximum number of bytes per task timeslice, hence other tasks will not be starved even if another Yenta attempts to monopolize its attention. In addition, Yenta will start dropping interests if the list from any given Yenta is too large, and will drop additional vectors of any one interest if it exceeds a threshold.

It is certainly possible to make Yenta's rumor- or cluster-caches useless by inventing a very large number of unique-seeming Yentas—e.g., for example, with a single process that keeps claiming to have a different Yenta-ID for every connection—and then bombarding a stream of connections. This can certainly fill the rumor cache with a large amount of junk, making this particular Yenta's referrals useless to *other* Yentas. It can also fill the rumor cache with *known* junk, hence compromising the digital mix described in Section 2.8.3; we shall have more to say about that below.

If the attacker can deduce the local Yenta's interests accurately enough, such a bombardment might conceivably also fill its cluster cache with entries which all correspond to the attacker's identities. This can effectively cut off the Yenta from *real* clusters that share its interests, and resembles the case of a Byzantine attack, but mounted from a single host. Whether or not this attack can succeed also depends on whether the local Yenta is using the attestation system to reject other Yentas which do not have appropriately-signed attestations.

Defending against such attacks can be quite difficult. One easy solution, which is not currently implemented in Yenta but which would be quite simple to do, involves throttling the number of unique Yentas accepted from any given IP address in some time interval—for example, no more than 100 unique YID's from any given IP address in a month. This raises the workfactor for the attacker considerably, since the attacker must now control many more hosts. [IP spoofing is not a reasonable approach for the attacker, since the communications protocol depends upon two-way TCP traffic, including a cryptographic challenge, which means the attacker must see return packets. And yet we have also assumed that the host's underlying network is working; this means that routing is working and the attacker therefore cannot simply be sitting on the host's local interface, or on the local wire, and modifying all packets—this counts as the network *not* working.] Note that we cannot throttle the number of YID's per IP address to only *one* unless we wish to break the ability to use Yenta on a timesharing system—if we were to do this, every Yenta connecting to the local Yenta from the timesharing host, except the first would be dropped as an attack.

All Yentas

Denying service to *all* Yentas is a trickier task. Assuming that the distribution is not corruptible—meaning that both the signatures on the distribution and the evaluations in Yvette are secure—one possibility would be to spuriously advertise some critical problem in Yenta to its user community, perhaps via a mailing list. If this ever becomes a problem, all mail from the maintainer to Yenta's users will have to be digitally signed so they may check it for authenticity. At the moment, this is a fair amount of overhead, so such messages remain unsigned.

Because both the debugging logger and the statistics receiver are expected to occasionally be down, all Yentas can cope with the results and will not fail no matter how long these servers are down. Hence, even a total failure of these servers will not bring down all Yentas. Further, since the communication from Yentas to the central server is essentially one-way at the level of the Scheme protocol sent, there is little opportunity for the server to freeze a Yenta via an inappropriate response.

It *is* possible that there is some way to subvert the SSL implementation of the statistics server—e.g., at a protocol level below the actual Scheme forms exchanged—such that it causes a Yenta which is trying to log statistical information to freeze—for example, by exploiting some bug in the implementation which causes SSL negotiation to hang if the server simply halts at the appropriate moment. If the C code of the SSL implementation is frozen, Yenta will freeze, because no other tasks will ever run. Since all Yentas eventually attempt to log to this server, this will freeze them all. It is currently unknown whether such a vulnerability in the SSL implementation exists. A possible solution, if it *does* exist, would be to wrap a timeout around all SSL session negotiations and simply abort any tasks whose timeout expires. This can cause each Yentas to become momentarily sluggish each time it tries to log, but this should not be a major performance problem if the timeout is not excessive.

5.7.2 Integrity and confidentiality—protocols

As discussed in Section 5.6 above, it is believed that the most serious risk to either integrity or confidentiality of the data exchanged by Yentas is that of poor security practices by its users. This ranges from running Yenta on compromised machines to picking poor passphrases to using web browsers which only allow 40-bit keys to typing passphrases or otherwise using the user interface across insecure links—e.g., running a web browser via X and then using an insecure connection between the X server and the X client. Such mistakes and poor practices are incredibly common and very difficult to guard against—for example, it is essentially impossible to know for sure, from a program's point of via, whether any given X connection is or is not secure, since the program must know more about the environment and the threat model than can be sensibly expected. Similarly, while Yenta *can* trivially simply refuse to talk to any web browser which fails to use encryption with enough bits in its keys, one can make the argument that this might needlessly disenfranchise users who are using Yenta in a way that even 40-bit browser keys are perfectly acceptable—such as the case wherein the browser is running on the same host as the Yenta and no bits are traversing the network.

The problem of weak passphrases

Yenta does not currently make any attempt to ensure that the passphrase chosen by the user is at all secure. This would be a relatively simple addition, but raises important concerns about users forgetting passphrases if they are forced to be long. Most users find themselves unable to remember an 80 to 160-bit string, even expressed as a passphrase of random words, on first sight. (It is commonly accepted that most humans can only commit about one bit of information per second to long-term memory; this has obvious implications for the long-term memorability of a newly-generated passphrase which is random enough to be unguessable by someone else.)

Maybe users should write them down?

The best solution to the passphrase issue might actually be to *encourage users to write down their passphrases* somewhere, such as on a scrap of paper in a location known to the user. This is heresy to the traditional security establishment, but certain threat models may make it sensible. For example, many users may be in an environment where local users can become root on their workstation (e.g., system administrators) and no passphrase will protect such users. However, nonlocal users may be arbitrarily distant and may have no idea where the user is physically. Given such an attacker, a written-down passphrase is no less secure than one which is not—but writ-

ing down the passphrase may encourage the user to pick one that is sufficiently long as to have a useful number of random bits in it.

5.7.3 Integrity and confidentiality—spies

Assuming that the basic cryptographic protocols are adequate, and that the user is using Yenta safely—no insecure browser connections, a good passphrase, and so forth—there are still underlying issues of confidentiality in particular. Consider the denial-of-service attack described above in Section 5.7.1, in which a single Yenta is targeted by an effectively unlimited number of bogus other Yentas, all under control of a single attacker. Whether or not Yenta throttles unique YID's per IP-address and unit time, there is some combination of resources which is guaranteed to cause an arbitrary proportion of the local Yenta's communications to *all* be with the attacker's Yentas. At this point, the local Yenta has been captured and is in a case explicitly disallowed by our criteria in Section 3.2.4. How bad is the damage?

In the simplest case, this attack breaks the digital mix described in Section 2.8.3. This means that, when the local Yenta exchanges interests with the attacker's Yentas, any interest which does not come from the attacker's supplied interests is known to belong to the local Yenta. This means that these interests are no longer plausibly deniable.

Trusted attestations may be the only feasible solution

This is quite a difficult attack to defend against. We cannot even attempt to spread information by insisting that third-party Yentas do the comparison of each interest, and then collating their responses, because all such third-party Yentas are themselves still under control of the attacker. It appears that the only obvious solution to this problem is to have the local Yenta insist that *every* Yenta which it talks to possess some attestation signed by a party which can be reliably known to *not* be the attacker. Of course, this is very likely to dramatically reduce the number of other Yentas that will be listened to by the local Yenta, perhaps to zero, but there seems little choice—if *everyone* you talk to is lying to you, and yet you feel compelled to tell *someone* of your interest in something, you are in trouble. Your only alternative may to figure out how to get someone you already trust to vouch for someone else.

5.7.4 Contagion

Is a network of Yentas vulnerable to contagion? Such an outcome could allow a malicious attacker to disable the entire system; it also allows cases in which the system might simply fail all on its own, by accident.

While it does not *appear* that there is any potential for such a thing, bizarre failures of this type have been seen in the past in other systems [121]. Yenta never accepts any code fragments from elsewhere, which should minimize the chances of a true virus being able to propagate. For example, when reading a Scheme form from a network connection, Yenta uses a custom-written parser that disallows almost all Scheme forms except lists, strings, numbers, and booleans. This guards against an attack which is otherwise possible against both Common Lisp and Yenta's particular version of Scheme, in which the attacker uses the #. reader macro—which means evaluate this form at *read* time, not load or compile time—to cause the machine parsing the form to execute arbitrary code. [For example, if Common Lisp calls *read* on the form *(+ 2 3 #.(malicious-code-here) 5)*, it will execute malicious-code-here before reading the rest of the form. Even if *eval* is not called on the result of the form (and hence the addition is not performed), the malicious code will have been already run. Common Lisp has the *with-standard-io-environment* form, which will inhibit #., but SCM does not and hence requires a home-grown solution to this problem.]

Yenta does not use this custom-written, safe parser when reading forms from the file in which is saves its permanent state. However, since this file is encrypted, an attacker would have to break the encryption to cause Yenta to execute arbitrary code, which seems like a much more difficult problem than simply causing the user to run the wrong application via a wide variety of easy attacks involving subverted hosts and the

like. Hence, attempting to propagate a virus in this manner would require manually compromising each host in order for it to succeed, at which point it can hardly be said to be a virus at all.

Deliberate shutdown

There was some thought given, early in the Yenta project, to having a global shutdown code installed in at least early versions of Yenta as released. Such a code would be intended to halt *all* Yentas reliably, in case the Yenta network protocol behaved badly and threatened the usability of the network infrastructure. The intended method of action would be to have each Yenta first broadcast the shutdown code to all neighbors, and to then halt for, e.g., no less than a week, before allowing itself to be run again. The code itself would be cryptographically signed using a private key known only to the implementors, and whose public half would be installed in every Yenta. It would contain an expiration date, after which any Yenta would ignore the stop code, such that it could not permanently kill all Yentas forever. And, to be extra safe, the code would presumably be implemented using a threshold scheme, such that several individuals would have to collaborate to reconstruct the required key to emit the code. Not only would this guard against an unfortunate mistake, but having several of the individuals be in different sovereign countries would aid in preventing a duress attack, in which the implementor was forced through legal or extra-legal means to disable the Yenta network—presumably by some government actor that wished to suppress anonymous encrypted speech.

Such a shutdown scheme would be a deliberately-installed method of destroying the Yenta network, at least for a time, due to an intentional contagion. Early results from Yenta indicated that the potential for a network meltdown due to Yenta was low, while the hazard of ever having such a mechanism installed in Yenta was high. Given this, and the implementation work required to install such a feature—and to verify that it *would* act correctly when triggered but would *not* trigger falsely—the feature was intentionally omitted from the fielded system.

It is nonetheless still entirely possible that there exists some pathological interest, message, or attestation which will be propagated to other Yentas and which causes any Yenta possessing it to malfunction. Such an outcome is exactly analogous to the ARPAnet collapse described in RFC528. No such mechanism is currently known. It is hoped that careful code review, for example via Yvette, may discover any such mechanisms *before* they are accidentally triggered. It is also hoped that such a malfunction will at least allow logging data to be returned to the implementor; this might allow the issuance of an updated version before all Yentas are crippled. However, in most scenarios it may be that the logged data would be insufficient—since interests, messages, and many attestations are *not* returned to the logging receiver, a pattern-dependent pathology in them will not be returned. Only when the implementor's Yenta failed would the actual pathological case be made available for inspection.

5.7.5 Central servers

Every central server in the fielded Yentas represents a vulnerability. As discussed in Figure 2.13, for example, the mere existence of the statistics receiver represents a great risk of accidental information disclosure if Yenta logs some identifiable information by mistake. In addition, the existence of such a server represents a user-perception risk—some users may not be interested in any protestations that such a design is safe, may distrust it on principle, and may not user Yenta for that reason. Given the sorry results from trusting similar sorts of assertions in other systems, it would be hard to blame them for such a stance.

The bootstrap server also represents a small risk. In particular, a malicious takeover of the server could cause all newly-started Yentas to be forced to talk to a particular set of other Yentas—presumably those under the control of whoever took over the bootstrap server. This is not guaranteed to work, since each Yenta that starts first broad-

casts on the local wire and only ask the central server if not enough responses are received, but it may succeed against Yentas that start in environments where few other Yentas are already running. Similarly, someone who can control answers on the local wire can subvert a newly-started Yenta into only using a particular set; since such an attack can only affect new Yentas on a particular wire segment, it has less potential for widespread mischief than taking over the bootstrap server.

The debugging server presents few risks, save that it is possible that a bug in Yenta's implementation—such as logging the value of some variable that could reveal something about the user's interests or the contents of conversations—may leak private information. However, Yenta's use of this server is rare. Because communication with this server is strictly one-way, from Yentas in the field to the server, it seems unlikely that taking over the server could accomplish much besides inconveniencing the implementors (and, secondarily, making it impossible for brand-new Yenta users to automatically sign up for the couple of mailing lists which talk about Yenta; they could still do so manually even in this case).

5.7.6 Nontechnical risks

Yenta faces some nontechnical risks which might also impact its utility. For example, how exactly to use the attestation system—what might be useful to say about oneself, for instance—has been left deliberately underspecified. In part, this is an experiment to see how people *do* decide to use the attestation system, but it may backfire—without sufficient guidance, users may not use it at all, or they may use it in such idiosyncratic ways that attempting to use the attestation system for *automatic* filtering of incoming messages becomes very difficult. (This is especially true given the rather user-unfriendly choice in current Yentas of requiring such filtering to user regular expressions; regexps are *not* expected to be understood by most users and it is hoped that a later version of Yenta will use something friendlier. How exactly to do this is a matter of some research.) Note that, even if users cannot use regexps in any useful way to automatically reject particular Yentas—hence leading perhaps to a spam problem—they may still manually add Yentas to their rejection lists, thus killing spam from any Yenta that has sent it even once. They may also, of course, still read attestations themselves and use their own judgment about whether to accept an introduction or a message from someone based on their own reading.

It is conceivable that Yenta may run afoul of patent issues. This is generally a problem in software systems these days, and especially problematic with those employing cryptography. It is also rumored to be a method of attack from corporate interests against free software generally, given that most authors of free software do not have legal counsel and certainly to do not have the war chest of patents that large companies tend to have. This is, alas, a risk that is not unique to Yenta.

Because Yenta facilitates anonymous, private speech, it is likely to irritate many governmental and even nongovernmental actors who have vested interests is discouraging such speech. (For example, the European Union has recently proposed—though not yet adopted—prohibitions against electronic anonymous speech [48]. This issue comes up frequently in the United States as well, despite its Constitutional protection in other media [65][120].) However, barring changes in existing law, especially in the United States where Yenta is being developed and fielded, it seems unlikely that excessive coercive force could be applied either to Yenta's users or to its implementors.

5.8 Other applications of this architecture

Let us now turn our attention to a brief evaluation of how the underlying architecture employed by Yenta might be used in several other applications. The questions we are answering here are: How well does the architecture support these applications? Where might the architecture need to be extended? Compared to more traditional ways of

implementing these applications, does this architecture offer unique advantages? This discussion is necessarily speculative—none of these sample applications have been implemented, although many of them would not be difficult to do.

Web pages

One example would be an application might use the contents of web pages that have been fetched by the user as the input to the document comparison engine, rather than the contents of the user's own files or mail as is currently done in Yenta. This application bears some resemblance to the Webhound/Webdoggie system [103], although it is actually a superset—not only is it distributed, unlike Webhound, but Yenta incorporates an interpersonal communication system which Webhound lacks.

Building such an application seems relatively easy. The minimal-work approach would be to use some other external program, such as the *wget* program, to fetch all web pages in the user's bookmark list, and then simply point Yenta at the resulting collection of files. Another approach, more convenient for the user but slightly more work for the implementor, would add the necessary code to Yenta to enable it to fetch web pages directly and feed them into Savant. Whichever approach is chosen, performance would probably be improved if the Savant comparison engine was augmented to understand more about the structure of the web page—such as attempting to compare web pages by number of outbound links to foreign sites, or number of included images, and so forth—because the Webhound/Webdoggie research showed that doing so improved the performance of that system as well. (Clearly, simply importing the relevant part of Webhound's page-comparison code would be a straightforward way to go about this.)

This application seems well-adapted to the architecture described in Chapter 2. It has considerably advantages over the original Webhound implementation, because users no longer have to worry about some central site knowing which pages they browse, and also enables them to easily share information about web pages by simply talking to each other—Webhound only suggested pages, with no explanation and no easy way of contacting the other user(s) who may have seen those pages already.

Database queries

Another example is an application that attempts to build groups of people who do similar database searches—a sort of community-builder that might be used within a single company that does database mining. Such an application could help inform those working in this hypothetical company about other groups or divisions which seem to be duplicating work, or which allow people doing similar searches to pool their resources. Implementing this application requires removing Yenta's existing document comparison engine—Savant—and implementing some new comparator which, given two database queries, can compute some similarity metric between them. It would also require some trivial modifications to change the printed representation that Yenta uses to describe an interest from a short vector of keywords to, perhaps, the actual database query that was issued.

Assuming that it is possible to develop some metric that can suitably compare database searches, then this application, too, should work well given the framework of Chapter 2. If used only within a company, the anonymity and privacy features presented could well be overkill, but perhaps not—intracompany politics and competition can sometimes be ugly. And the attestation system might be used to ensure that no information is somehow shared with rivals—consider a system in which the agents only talk to others which display attestations that have been properly signed by some well-known entity in the company, such as its human-resources department. This turns the web-of-trust architecture presented by the attestation system into the substrate for a more hierarchical, certificate-authority-based system, and enables a high degree of trust that any given agent really *does* belong to a user who works for the company. Properly done, this assurance can be much strong that trusting to a firewall

or to the domain-name system, both of which tend to be easy to subvert in practice [128][161].

Romantic matchmaking

What changes would be involved in making Yenta a true *romantic* matchmaker—an application that was explicitly designed for dating? On the surface, this seems both obvious, simple, and well-adapted to Yenta—for example, the attestation system might serve very well in helping to controllably share certain crucial information between prospects, while the underlying cryptographic security and nym system can help to control undesired information flows before partners commit to a physical meeting, if they ever do.

But a closer look shows that this is not quite the problem that the original Yenta was designed to solve, for a number of reasons. Yenta assumes that shared interests are sufficient to bring people together, but romantic matchmakers cannot make that assumption—indeed the phrase opposites attract may be quite relevant for many users. In part because of this, romantic matchmakers often require each user to specify a profile which describes an intended match, and this profile may bear little resemblance to the user creating it. This lack of self-similarity—we may be attempting to match users based on characteristics they do not share—breaks a naive implementation of the clustering algorithm described in Section 2.8. In addition, a handmade profile may lack the ability to do hillclimbing, because such profiles often consist of very few words (e.g., 10 or 20) and not the large number of words—and hence long vectors—that document summaries such as Savant tend to generate. One possible solution to this might be to instruct potential users to instead pick, say, online works of one sort of another—web pages, book chapters, and so forth—that could be of interest to a potential mate. A profile-creation step which requests large amounts of information in ways that a comparison engine can partially order may also help; this would require careful thought and correctly-structured data. But how do we deal with the opposites-attract problem?

For concreteness, let us name our potential users *Harry* and *Sally*, and consider some ways out of this dilemma if Harry is looking for a mate who is *unlike* him, and Sally is also. Assume that Harry wants someone who is outgoing and friendly, but is himself curmudgeonly; likewise, assume that Sally *is* outgoing and friendly, but wants a curmudgeon. They would be perfectly matched, if only they could find each other. However, if Harry creates a profile that looks for outgoing, friendly people, and Sally finds it, she will incorrectly assume that Harry *himself* is not a curmudgeon, and will reject the proposed match.

One way out of this might be to modify Yenta such that it understands explicitly the connection between *pairs* of interest clusters, such that Harry Yenta can cluster itself into a clump of other curmudgeons, while simultaneously clustering itself into an outgoing-and-friendly cluster. Sally's Yenta could presumably do the same. If both Yentas then understood the *meaning* of each finding themselves in both clusters simultaneously, and if each Yenta kept track of which cluster represented a profile of its *own* user and which cluster represented a profile of the mate being sought, then it is possible that the opposites-attract problem might be solved. It does not seem, at least at first glance, that this is a prohibitively difficult programming project, although it does seem to be the sort of thing that might require extensive tuning and a careful user interface so as not to confuse its users.

Electronic commerce

Let us now consider implementing some sort of ecommerce system, in which buyers and sellers wish to find each other in order to exchange goods. The first order of business involves creating some sort of comparison metric that can translate some description of goods or services into something upon which a partial order of similarity may be imposed; otherwise, clustering cannot use hillclimbing. One potential

approach to this problem, depending on the domain, might be to embed products in a hierarchy of related products, and to measure similarity between two types of products by comparing the distance between points in the hierarchy. This might allow clustering to build groups of buyers and sellers which are interested in similar products.

The next issue concerns what to do once a cluster has been formed. One approach might be to have buyers and sellers simply broadcast messages, using the messaging system described in Section 2.10. The message sent may either be human-to-human, as in Yenta, or algorithmically generated, for example some sort of open-outcry bidding system—there are no doubt a large number of potential algorithms which could take advantage of such an architecture. Once a buyer/seller pair are aware of each other's existence, they may of course also simply send messages directly to each other, again as human-to-human or in some sort of automated bidding algorithm.

The attestation system could be used to excellent effect in such a system. For example, buyers who are happy with the seller's performance may volunteer to sign attestations from the seller which verify that the seller is trustworthy. (Recall that attestations, being kept by their owner, will presumably not be kept if uncomplimentary, and hence buyers will be unlikely to be able to say negative things about sellers because sellers will not offer such attestations in the first place; see Section 2.11.) Buyers might themselves have attestations which sellers can sign—perhaps as part of some other element of the transaction—indicating that the buyer has paid for prior purchases.

The support of this architecture for private, authenticated message exchange, combined with the attestation system, makes the architecture described in Chapter 2 an attractive choice as the substrate for an ecommerce system. The most difficult part of the design which is unique to the architecture—as opposed to, say, which bidding strategies to use and so forth—is likely to be support for forming clusters in the first place. If there is no natural landscape of similarity which can be used to support the hillclimbing, this approach may not be acceptably efficient.

5.9 Motivating adoption of the technology

Designing, implementing, and fielding a decentralized application has many challenges. While doing so can have important benefits for users, it can be significantly more difficult for implementors than a centralized system, for a number of reasons:

Challenges

- It is much more difficult to update a large number of applications in the field than a single, central application. This implies that the system must be closer to production quality—not alpha or beta—before first ship.
- Because the application must be significantly more robust at first release than is often observed, it may take longer for a given development staff to field such a system than many centralized systems. For uses, such as in businesses, where time to market is the dominant factor, such a delay may be a major liability.
- The application may have to be more complex to correctly handle the inevitable mix of versions that will be present in the field.
- There exists a significant issue of *education*, of the user base and of others who must talk about or interact with the system, because truly decentralized systems are still unusual. During deployment of Yenta, for example, even sophisticated users continued to ask, "Where's the server?" repeatedly until they understand how the system operates.

Solutions

These issues need not be fatal. For example, many centralized systems such as web-based ecommerce sites rely on an already-implemented toolkit, for example the Apache web server [7], and do a relatively small amount of additional work to add whatever functionality is required to make their sites into a business. A similar,

widely-available toolkit for building decentralized systems—such as a commercial-strength infrastructure that implements the basic architecture described in Chapter 2 and Chapter 3—could go a long way towards enabling rapid implementation and deployment of such decentralized systems. In addition, such a prebuilt toolkit might help to avoid some of the more common errors in the implementation and use of cryptographic algorithms and protocols, since the work required to design and verify them can be spread across multiple applications and multiple reviewers. While it is still possible for some other part of the deployed system to compromise the otherwise-good crypto, at least there is that much less of the design and implementation that must be written and checked.

Motivating businesses

A more serious concern, at least for business users, is the large value of data-mining to many businesses and their reliance upon such data as a revenue stream. Indeed, with the falling price of computers, some businesses are willing to give away a computer valued at several hundred dollars *for free* in return for the ability to collect vast amounts of detailed personal information from their users as the computer is being used [85]. In this case, even if a decentralized system offers technical advantages, such as robustness and insulation from the labor of answering subpoenas, and even if the system is viewed favorably by users, the business must forgo a revenue source in order to be socially responsible. This is a tradeoff that few businesses are apparently willing to undertake.

Clearly, if the financial motivation is sufficiently large, almost all businesses will ignore any scheme that protects their users' privacy. Such a motivation would have to factor in the possible loss of goodwill from customers, the time and effort required to answer subpoenas, and the possibility of enforcement action from government actors.

Thus, the solution for motivating businesses to do the right thing—in this case, protecting the privacy of their users—must eventually come down to making it too expensive for them to violate their privacy instead. This is very unlikely to be a purely technical solution, given the example above where it is obvious that detailed information about particular users may be quite valuable commercially. Instead, businesses must either lose customers, and hence revenue stream, to others which are more protective of their customers' civil rights, or they must be forced to be more protective via legal action.

Given a scenario in which a privacy-protecting system gets to market at approximately the same time, and costs a business less—for the various robustness reasons mentioned elsewhere, for example—it is also in the interest of that business to *educate its customer base* about why they are getting a superior deal in terms of their civil rights. Such an education and advertising campaign, if properly handled, may pay off by discouraging customers from using competing systems that are not so protective. While it is historically true that such campaigns are difficult and often do not motivate a large proportion of users, it is always possible that such attitudes will change—for example, if well-publicized privacy disasters continue to emerge.

Legal remedies are another option. At the moment, the United States in particular is in a poor position in protecting privacy rights, as discussed in Chapter 1. One reason is certainly the lack of public awareness of the problem. Another may be the sense that the situation is in some way inevitable—that the use of computers to handle personal information *must necessarily* lead to reduced privacy. It is hoped that the results of this research will serve as an example that this need not always be so. If this example becomes widely known, it may influence legislative attitudes by making it obvious that many businesses have no technological justification for their actions. This may thus lead to legal pressure on businesses and other actors for the protection of their users' rights.

5.10 Future work

Having a large number of Yentas in operation provides several intriguing opportunities for further study. We shall investigate some of these here.

5.10.1 Sociological study

Any new technology can benefit from studying the way in which people use it. Yenta, in particular, is an unusual combination of matchmaking service, mail system, collection of newsgroups, document summarizer, and reputation system, among others.

One obvious approach involves exploring the sorts of groups that form, and whether users find that they deliberately include or exclude certain types of documents to try to find particular such groups. Since there is no toplevel ontology of which groups exist, the prevailing social structure is more like the one that exists in everyday, non-networked life—one cannot simply ask, "What are all the interest groups in the world that I might possibly become a member of?" because there is no such central registry. Yenta shares the same characteristics. Yet users who hear through channels outside of Yenta about particular groups may be tempted to try to join them. If Yenta does not support this explicitly, users will likely find a way—but how?

Yet another possibly-fruitful direction concerns the reputation system. What will people say about themselves? What will (and won't) get signed by others? What social signalling systems will evolve? Will these systems span clusters or not? What sorts of filters will people write to take advantage of the reputation system—or will they use it only to evaluate potential conversational partners? What are the patterns of signatures—can we infer anything about social organization by who signs whose attestations? The range of possible questions is very large, but could be sociologically interesting to answer.

5.10.2 Political evaluation

Yenta also has a political dimension. Will it change the way people tend to think of privacy and computer-based processing of personal information? Will it influence systems designers to take civil liberties more into account? Will the decentralized nature of the architecture lead to more such architectures, even in cases where it is, for example, robustness, and not privacy, that is most at issue? Will the transparency goals for vetting its source code—particular Yvette—lead to other projects being easier to evaluate collaboratively?

All of these questions are good ones, and it is hoped that they will be the subject of future research.

5.11 Summary

In this chapter, we have evaluated the architecture via simulation, and demonstrated that it scales to realistic sizes and performs well. We then described how to instrument the sample application so it could be analyzed, and discussed qualitative and quantitative data from a pilot deployment, which show that the application as fielded performs acceptably, and provides guidance on how to improve it. We then investigated some residual risks of the architecture and the application, including some exploration of how to defend against attacks that we declared to be outside of our original threat model. We have speculated on the methods that might be required to motivate business users to adopt the technology, despite current financial incentives to the contrary. Finally, we briefly mentioned some intriguing directions for future work.