iFlame
6.853 Red Group Project Report

Brian D. Carlstrom David M. LaMacchia

May 18, 1995

1 Introduction

One of the most popular uses of the Internet has always been electronic mail. Mail, while good for communicating with off-line users is not good for instigating continuous conversations with online users. From a UNIX standpoint, there has been much development to create communications programs to solve this problem. Network-aware versions of the `write` command and various `talk` programs allow two users to communicate. The Internet Relay Chat (IRC) allows many people to talk in groups, but, like Athena Zephyr, which also supports somewhat instantaneous communications, is server-oriented and thus slow when many people are using it.

iFlame is an evolutionary step beyond systems like IRC and Zephyr in that it solves many of their problems, is convenient, and adds new features. It is a revolutionary step in that it is inherently client-oriented. Like Zephyr, iFlame offers instantaneous communication that doesn't require continuous attention, but sends messages directly between clients instead of having the server handle them.

We propose iFlame as a solution for an integrated message system. It is designed to support various interfaces depending on the user's preferences backed by authentication and encryption and a scalable model for providing Internet wide service. Below is a discussion our initial design which will be further refined as time goes on.

2 Design

This section describes the basic design of the *iflame* system, from how users organize themselves into groups based on a common topic to a simple example of how messages are sent between clients.

2.1 Forums

The heart of iFlame resides in *forums*. A forum is a group of *users* who are communicating with each other about a common subject. Underneath, a forum consists of three parts. The first part is a collection of *Access Control Lists* (ACLs) that constrain who can read a forum, who can write to a forum, and who can administer a forum. The latter parts are the lists of *locations* of active readers and writers of a forum. This general model allows the user a lot of room for customization. For example, a forum to receive private messages would allow anyone to write but only would allow one reader. A user could have a negative ACL to disallow certain users from writing to them. Login forums afford another example. A login forum would have one allowed writer, the person logging in, and multiple readers, those who wish to be identified when that person logs in or out. Similar to the first example, a negative ACL could be used to prevent others from listening to a login forum for privacy reasons. Forums with multiple readers and writers would be suitable for group communication.

2.2 Locations

When a user joins a forum, its location is added to the list of locations of active readers. A location uniquely identifies the user's client within the iFlame system. A user may join a forum multiple times from different locations. Each Internet location entry is composed of both an IP address and port number. It is important to note at this point that port numbers will be dynamically chosen by the client when started: multiple users on a single machine will all be able to have their own iFlame clients. Furthermore, it is not necessary for any part of the client to run as the root user to dynamically establish these connections.

Locations are extremely important for both the client and server. On the client end, a cache for each forum tells that client who to send *flames* (message-grams) to. The server maintains a database of which locations are reading and writing each forum and the ACLs for each. These are described in more detail under *Server Specific Design Issues*. The server has the ability to dynamically update the client's cache as its database is updated.

2.3 Users

Users are key to authentication in iFlame. Each location is associated with a user. When forum requests are made from a location, the location's associated user is used to check against the forum's ACLs. Whenever a connection is made between a client and a server, that connection is authenticated using one of various forms of authentication enabled on the client end at initialization. Authentication types can include unauthentic communication, a plain-text password system, Kerberos, or a PGP based system.

Currently, a user in the iFlame system resembles an e-mail address more than an IRC nickname or personal Zephyr instance. This is because a user is associated with an array of locations but also a home server. A user, say, *flamer@iflame.mit.edu*, is a long-term association that includes authentication information such as a password or a PGP key. Users on other servers can contact this user at this address with assurance that the person using this iFlame account is the same from session to session.

2.4 A Simple iFlame Interaction

While forums, locations, and users are important to iFlame's protocol, the advantages of the system reside in its protocol for message passing. Because messages are sent in a client-to-client fashion and servers are only used for tracking location information, transactions put much less load on the servers than a centralized-server system like IRC or Zephyr.

The main delivery path transfers a message from a user to an iFlame client (`iflamec`), then between iFlame clients, and is finally output to the user. The server is not involved at this level. One of three situations require the server to connect to the client or vice-versa. First, the client could become a new *reader* on a forum; in this case, the server must be notified so that it can update it's cache. Second, the client could become a new *writer* on a forum, in which case the server must notify the client of all current readers on that forum so it knows who to connect to. Third, if someone joins or leaves a forum that a client is a designated writer for, the server will have been notified and will subsequently relay that information to the writer client.

Of course, there are other times the client must talk to the server; when initialized, the client determines what authentication system it wishes to use with the server. This determines which forums it will be allowed to join, based on each forum's ACL.

3 Client Implementation

The client is implemented in two parts: `iflamec`, the half of the client that maintains cache information and talks to servers, and `iflame`, the half that communicates with the user and delivers messages to other `iflamec`'s. This division is the core of the client's design and gives us numerous advantages.

3.1 Basic Architecture

`iflamec` is the system's cache maintainer, display handler, and authentication overseer. It is connected to `iflame` via a Unix socket; while shared memory was also considered, Unix sockets were chosen for the client end to preserve portability and minimize complexity. `iflamec` also has a number of TCP sockets: one for incoming connection from clients, one for incoming connections from servers and one for outgoing connections to servers.

`iflame` is the user interface to `iflamec`; users are able to type commands to `iflame` which it interprets and uses to join or leave forums or to send flames to other users. The interface is easy to use and allows users to join or leave to forums, set ACLs for forums, and send flames. On the command line, a user is able to specify a remote non-default server at any time by tagging a forum with @[remote_server_name]. Thus, similar to e-mail, sending "+r test@iflame.mit.edu" would send a +r message to the test forum on server iflame.mit.edu (of course, if iflame.mit.edu is the default, this need not be specified).

3.2 Flamage

Flamage, the sending of messages between clients, is the central purpose of the iFlame client and its implementation is straightforward. While all messages are sent from a user's `iflame` program directly to remote users' `iflamec` clients in a point-to-point fashion, there is also a significant interaction between the local `iflame` and `iflamec`.

`iflamec` has a main dispatching loop that handles incoming clients and servers. When a user desires to send a flame to a forum, `iflame` checks with `iflamec` to see if that forum exists within its local cache. If so, `iflamec` enqueues the list of locations of readers of a forum in a single package and transfers this back to `iflame`, which, as locations are dequeued, sends a copy of the message to each remote client. In the case where a forum has two hundred readers, the client bears the load of sending those messages across the net, instead of a particular server. In reality, studies of the Internet Relay Chat have demonstrated that for it's current load of 8–10,000 users at any time, most channels average less than 25 consecutive members.

On the receiving end, the remote `iflamec` accepts an incoming TCP connection from `iflame`. The local `iflamec` can continue to receive flames while `iflame` is sending. iFlame messages currently use a format similar to electronic mail with RFC822 style headers such as `To:`, `From:`, and `Signature:` followed by the message body. If the message is well-formed, `iflamec` passes the content to its display manager.

3.3 Caching

Caching occurs only in `iflamec`. Caches are maintained for the forums the local client is writing. Each cache is maintained as a balanced red-black tree for fast storage and retrieval. Nodes in the tree use a forum name as key and point to another red-black tree as data; this subtree has nodes that contain location information for each reader. This cache is updated by the `iflamed` server when readers are added or removed from forums the client is writing to. In these cases, `iflamed` sends a +/-F command to add or delete these locations from forums.

3.4 Configuration

Configuration of the `iflamec` client is easy. Three files control the environment: `.iflamec.var` contains local variables that are set at run-time (such as user name or optional variables). `.iflamec.sub` contains a script of `iflame` commands to be run at initialization. `.iflamec.tcl` contains TCL code to control the behavior of `iflamec`.

3.5 User Scripting and Graphical Display

`iflamec`, like other message programs such as Zephyr, supports the ability for users to extend the basic functionality with an embedded scripting language. Because many people already use TCL and there are lots of good examples of TCL applications, we decided using TCL as our extension language would be optimal. Furthermore, we use Tk as the default for graphical displays so this can be easily configured as well, although we also support a terminal mode. `iflamec` currently uses tcl7.3/tk3.6 and it will be upgraded to tcl7.4/tk4.0 in the future.

The use of TCL/Tk, while having many advantages, has some potential drawbacks. TCL allows a friendly configuration environment, but Tk may be slow in comparison to direct access to the X toolkit. While this performance issue will be studied in the future, we believe the extensibility of Tk is a great advantage in allowing others to write customized iFlame clients.

3.6 Authentication and Access Control

`iflamec` also handles authentication to `iflamed`. The various authentication forms are used by the server for verifying access to controlled forums. The protocol currently supports two simplistic forms of authentication, unauthentic and password-verified, and is open to support for Kerberos and PGP key exchange.

4 Server Implementation

This section describes the current implementation of the `iflamed` server. This second generation server is now relatively stable, representing significant amount of work over our original prototype iterative server. The basic architecture of the server is a monolithic process that stores the iFlame forum database as well as managing concurrent connections for incoming transactions and outgoing cache updates.

4.1 Data Structures

The `iflamed` state consists of forums, locations, and users and their relationships. They are currently manipulated using red-black tree routines. It is unclear if any type of data caching will be helpful for the performance of the server. Although the server supports simultaneous concurrent connections, it still performs all operations on the data structures using the code from the original iterative server which is atomic. This is not a problem at the moment since all operations are inexpensive. The atomic operations allow for cleaner code and reduce the complexity of the server.

4.2 Memory Management

Because the server is expected to be long lived, special emphasis was put into the memory management routines to help prevent memory leaks. For debugging purposes, counters are maintained to track the number of various server structures which can be compared to what is reachable from walking the data structures. Because of the complex relations between data structures, reference counts are used to control deallocation of structures. Although having a garbage collector walk the structures periodicly would be possible, for now routines that decrement reference counts just check locally released data for deallocation.

4.3 Concurrency

A large part of engineering the server was building the I/O system. The server maintains a table for all the file descriptors that associates open file descriptors with a connection structure. The connection structure consists of the associated user and location,, input and output buffers, timeout information, and the current state in the connections finite state machine. This system seems complex compared to using something like a threads package but it was felt that in this case portability was more important than code simplicity. In retrospect, this seems like a good decision. The system was easier to build and debug than had been expected. Time was also saved by not having to seek out, evaluate, and comprehend the various issues in current user-level threads systems.

4.4 Scheduling Client Cache Updates

One last interesting design note about the server is its cache update queue. Instead of actively opening multiple outgoing connections for updating client caches, the server works through a single queue this is an attempt to minimize server load for a slight tradeoff in forum consistency. Since it would be difficult to guarantee how quickly updates are distributed, this doesn't seem to be a major issue.

The benefit of the queue system is that multiple updates for a single client can queue themselves together and reduce the number of required connections. Also, if a client's transactions are complete and an outgoing queue exists for that client, the connection can be turned around so that the client immediately receives cache updates without establishing a new connection. This is very helpful, especially because many updates can be generated when a client's cache is empty when first joining a forum. One last feature of the update system is that when the server detects that a client has disappeared from the net while attempting an update, it can free up the resources and notify other clients to stop writing to the lost client.eared from the net while attempting an update, it can free up the resources and notify other clients to stop writing to the lost client.

5 Performance

At the current time, we don't have any performance measures for the iFlame system. This is because we don't have a large enough test-bed. Once a full release is prepared, however, we hope to have enough people using it that we can examine performance and identify more optimizations. Having a large number of users will also give us more information about iFlame's scalability.

6 Future Work

The systems has started to become fairly usable, enough so that we are planning on making an initial test release soon. However, there are many more issues to be addressed before the system will really be usable.

6.1 Client

The iFlame protocol supports encapsulation of all messages; this will be included in a later release. Currently, clients can pass arbitrary binary data. It should be easy to MIME-encapsulate different types and handle images, audio, video, or other MIME messages. This would also provide an easy means for supporting encrypted messages, especially if iFlame moves to a PGP authentication mechanism: the system could easily use public authentication keys for encryption as well.

The client is also moving to be concurrent. This will the allow the client to send outgoing data while queuing incoming data. It also means that when sending data to many other clients `iflame` will not have to wait for a recipients to accept connections.

6.2 Server

There are several features that need to be added to the server to make operation a little smoother. One important item on the agenda is a set of tools for administrators to easily maintain the permanent user and forum state. Another useful feature will be the ability of client's to give hints to the server when they think another client may have disappeared from the network. This will allow the server to quickly inform other clients of their peers demise.

The main goal for a next generation server is reliability. This might be achieved in the short term with the ability to checkpoint server state as well as using a log mechanism to allow recovery from server crashes. The real end goal is to have replicated servers to spread server load as well to improve server availability.

6.3 Global Forum Resolution

We recognize the need for global forum resolution; users inherently don't want to consider which servers their favorite forums reside on. We intend to create a meta-level resolver system that will be able to transparently direct clients to the servers of forums, thus making their appearance global to the user.

6.4 Authentication

In the future, iFlame will support an authentication system that will be transparent to the user. This relies on the user previously having established some authentication information with the server, just like in a password-based system. One idea involves the use of PGP keys; each server will have an available public key and each user will also have a public key established with that server. These can be used to authenticate the client. Because these public keys would already be in the system, they could be used for transparently encrypting messages as well. One interesting idea is having a servers authenticate clients for each other: the default server could sign a message with its secret PGP key. When received by another server (that would already have the default server's public key), this message and the client's authentication could be verified.

7 Conclusion

The iFlame system hopes to eliminate many of the problems of its predecessors as well as provide new functionality. The primary potential is because of its unique client heavy design with a server maintained cache. It will be interesting to see if performance costs of maintaining the writers' cache of readers will be low enough for the system to be truly scalable. In any case, the system will provide a unique opportunity to combine many systems software ideas in new and interesting ways.

References

[1] D. Crocker, *RFC822: Standard for the format of ARPA Internet text messages*, 08/13/1982.

[2] C. A. DellaFera, M. W. Eichin, R. S. French, D. C. Jedlinsky, J. T. Kohl, and W. E. Sommerfeld, Athena Technical Plan *Section E.4.1: Zephyr Notification Service*, M.I.T. Project Athena, Cambridge, Massachusetts (June 5, 1989.

[3] C. A. DellaFera, M. W. Eichin, R. S. French, D. C. Jedlinsky, J. T. Kohl, and W. E. Sommerfeld, *The Zephyr Notification Service*, M.I.T. Project Athena, Cambridge, Massachusetts (December 21, 1987)

[4] B. Kantor, P. Lapsley, *RFC0977: Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News*, (February 1, 1986).

[5] J. Oikarinen, D. Reed, *RFC1459: Internet Relay Chat Protocol* (May 26, 1993).

[6] T. Rinne, *RFC1756: Remote Write Protocol - Version 1.0* (January 19, 1995).