



P2PChat

Language Abstractions for Concurrent and Parallel Programming (1DL540)

Lucas Arnström

Department of Computer Systems
Uppsala University

2016-12-14



Introduction

■ What is P2PChat?

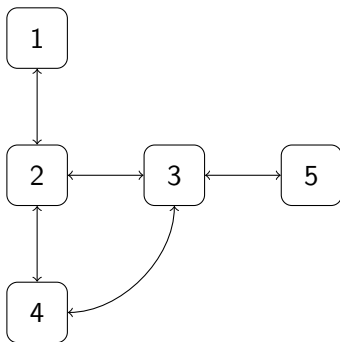
Chat program which utilizes peer-to-peer connections in order to construct a large network of clients whom all participate in a single global group chat.

■ How does it work?

- Clients connect to each other and create a network.
- Clients can then broadcast messages over the network.
- No tracker sadly; clients have to manually connect to each other.



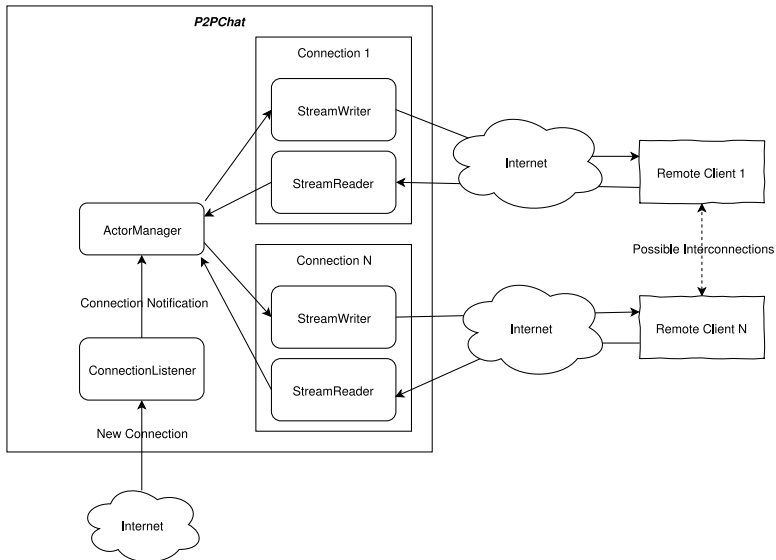
Example Network



Five clients connected to each other forming a simple network. If client 5 broadcasts a message, it will traverse the entire graph and eventually reach all nodes in the network.

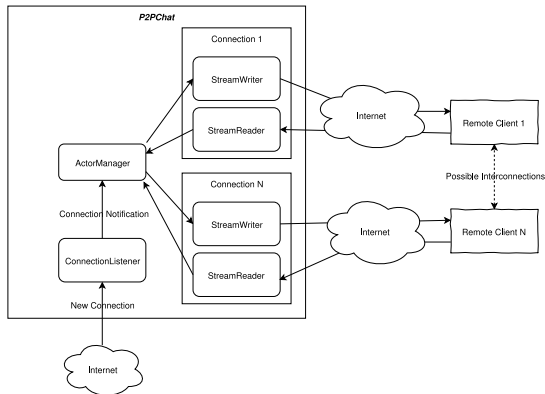


Application Structure



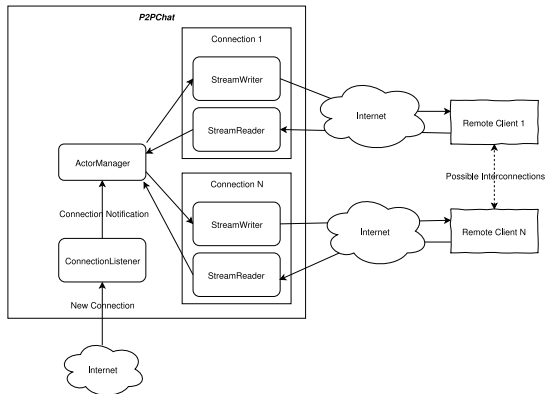


Application Structure





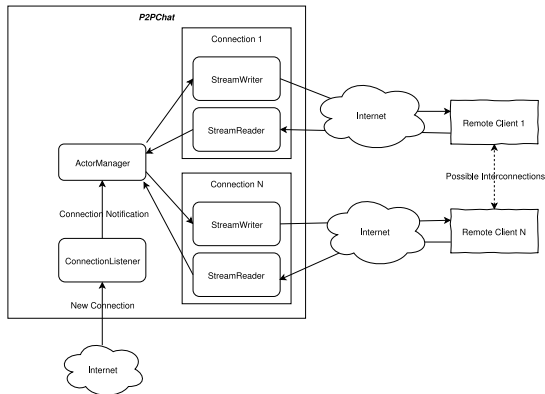
Application Structure



- Rounded rectangles represents actors.



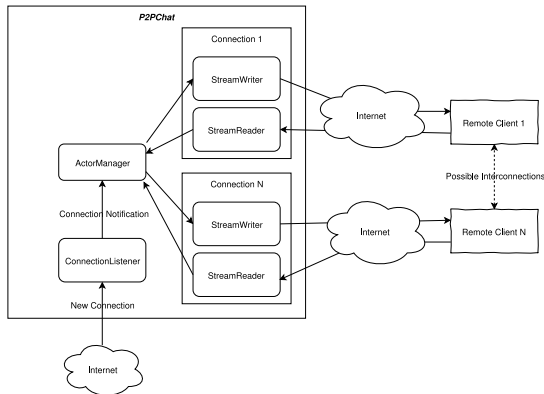
Application Structure



- Rounded rectangles represents actors.
- Arrows inside big rectangle represents channels.



Application Structure



- Rounded rectangles represents actors.
- Arrows inside big rectangle represents channels.
- Arrows outside represents TCP sockets.



Application Structure

The program consists of three main parts:



Application Structure

The program consists of three main parts:

- Actor Manager



Application Structure

The program consists of three main parts:

- Actor Manager
- Connection Listener



Application Structure

The program consists of three main parts:

- Actor Manager
- Connection Listener
- A Connection



Application Structure

The program consists of three main parts:

- Actor Manager
 - Manages a list of connected clients and their associated actors.

- Connection Listener

- A Connection



Application Structure

The program consists of three main parts:

- Actor Manager
 - Manages a list of connected clients and their associated actors.
 - Broadcasts incoming messages to all connected clients.

- Connection Listener

- A Connection



Application Structure

The program consists of three main parts:

- Actor Manager
 - Manages a list of connected clients and their associated actors.
 - Broadcasts incoming messages to all connected clients.
 - Essentially the main hub of the program.
- Connection Listener
- A Connection



Application Structure

The program consists of three main parts:

- Actor Manager
 - Manages a list of connected clients and their associated actors.
 - Broadcasts incoming messages to all connected clients.
 - Essentially the main hub of the program.
- Connection Listener
 - Listens for new TCP connections.
- A Connection



Application Structure

The program consists of three main parts:

- Actor Manager
 - Manages a list of connected clients and their associated actors.
 - Broadcasts incoming messages to all connected clients.
 - Essentially the main hub of the program.
- Connection Listener
 - Listens for new TCP connections.
 - Upon a new connection it does three things:
- A Connection



Application Structure

The program consists of three main parts:

- Actor Manager

- Manages a list of connected clients and their associated actors.
- Broadcasts incoming messages to all connected clients.
- Essentially the main hub of the program.

- Connection Listener

- Listens for new TCP connections.
- Upon a new connection it does three things:
 - 1** Spawns two new actors that separately read/write to/from the socket.

- A Connection



Application Structure

The program consists of three main parts:

- Actor Manager

- Manages a list of connected clients and their associated actors.
- Broadcasts incoming messages to all connected clients.
- Essentially the main hub of the program.

- Connection Listener

- Listens for new TCP connections.
- Upon a new connection it does three things:
 - 1 Spawns two new actors that separately read/write to/from the socket.
 - 2 Creates new channels between the Actor Manager and the two new actors.

- A Connection



Application Structure

The program consists of three main parts:

- Actor Manager

- Manages a list of connected clients and their associated actors.
- Broadcasts incoming messages to all connected clients.
- Essentially the main hub of the program.

- Connection Listener

- Listens for new TCP connections.
- Upon a new connection it does three things:
 - 1 Spawns two new actors that separately read/write to/from the socket.
 - 2 Creates new channels between the Actor Manager and the two new actors.
 - 3 Notifies the Actor Manager by sending it a message containing a new Client object.

- A Connection



Application Structure

The program consists of three main parts:

- Actor Manager

- Manages a list of connected clients and their associated actors.
- Broadcasts incoming messages to all connected clients.
- Essentially the main hub of the program.

- Connection Listener

- Listens for new TCP connections.
- Upon a new connection it does three things:
 - 1 Spawns two new actors that separately read/write to/from the socket.
 - 2 Creates new channels between the Actor Manager and the two new actors.
 - 3 Notifies the Actor Manager by sending it a message containing a new Client object.

- A Connection

- An abstract concept. Each connection is represented by the two actors spawned by the Connection Listener.



Demo

DEMO



Why Rust?

- Why Rust instead of any other language?

¹Threads can share data, but under very strict conditions.



Why Rust?

- Why Rust instead of any other language?
 - Forces me to reason about memory ownership.

¹Threads can share data, but under very strict conditions.



Why Rust?

- Why Rust instead of any other language?
 - Forces me to reason about memory ownership.
 - ▶ Only one scope can own and use a reference at any given time.

¹Threads can share data, but under very strict conditions.



Why Rust?

- Why Rust instead of any other language?
 - Forces me to reason about memory ownership.
 - ▶ Only one scope can own and use a reference at any given time.
 - Automatic memory management without the need of a garbage collector.

¹Threads can share data, but under very strict conditions.



Why Rust?

- Why Rust instead of any other language?
 - Forces me to reason about memory ownership.
 - ▶ Only one scope can own and use a reference at any given time.
 - Automatic memory management without the need of a garbage collector.
 - Guarantees memory safety.

¹Threads can share data, but under very strict conditions.



Why Rust?

- Why Rust instead of any other language?
 - Forces me to reason about memory ownership.
 - ▶ Only one scope can own and use a reference at any given time.
 - Automatic memory management without the need of a garbage collector.
 - Guarantees memory safety.
 - Guarantees no data races.

¹Threads can share data, but under very strict conditions.



Why Rust?

- Why Rust instead of any other language?
 - Forces me to reason about memory ownership.
 - ▶ Only one scope can own and use a reference at any given time.
 - Automatic memory management without the need of a garbage collector.
 - Guarantees memory safety.
 - Guarantees no data races.
 - ▶ Does so by not allowing any threads to share data.¹

¹Threads can share data, but under very strict conditions.



Why Rust?

■ Why Rust instead of any other language?

- Forces me to reason about memory ownership.
 - ▶ Only one scope can own and use a reference at any given time.
- Automatic memory management without the need of a garbage collector.
- Guarantees memory safety.
- Guarantees no data races.
 - ▶ Does so by not allowing any threads to share data.¹
 - ▶ Need to use channels in order to pass data between threads.

¹Threads can share data, but under very strict conditions.



Why Rust?

- Why Rust instead of any other language?
 - Forces me to reason about memory ownership.
 - ▶ Only one scope can own and use a reference at any given time.
 - Automatic memory management without the need of a garbage collector.
 - Guarantees memory safety.
 - Guarantees no data races.
 - ▶ Does so by not allowing any threads to share data.¹
 - ▶ Need to use channels in order to pass data between threads.
 - Compiles to LLVM, and has a minimal runtime.

¹Threads can share data, but under very strict conditions.



Why Rust?

- Why Rust instead of any other language?
 - Forces me to reason about memory ownership.
 - ▶ Only one scope can own and use a reference at any given time.
 - Automatic memory management without the need of a garbage collector.
 - Guarantees memory safety.
 - Guarantees no data races.
 - ▶ Does so by not allowing any threads to share data.¹
 - ▶ Need to use channels in order to pass data between threads.
 - Compiles to LLVM, and has a minimal runtime.
 - ▶ Can essentially run on any hardware.

¹Threads can share data, but under very strict conditions.