

Your First P2P Application

Build a complete peer-to-peer chat application with a beautiful UI in 30 minutes.

What You'll Build

A real-time chat app where:

- Multiple users can join a chat room
- Messages are encrypted end-to-end
- No servers needed - purely peer-to-peer
- Modern, responsive UI
- Works on Desktop (Windows, Mac, Linux)



What You'll Learn

- Setting up a Pear desktop project
- Using Hyperswarm for peer discovery
- Building an interactive UI
- Managing P2P connections
- Best practices for P2P apps

Prerequisites

- Node.js 18+ installed
 - Pear CLI installed (`npm install -g pear`)
 - 15-30 minutes of focused time
 - Basic JavaScript knowledge
-
-

Step 1: Create Your Project

Create a new directory and initialize:

```
1 mkdir p2p-chat
2 cd p2p-chat
3 pear init -y -t desktop
```

bash

This creates:

```
1 p2p-chat/
2   └── package.json      # Project config
3   └── index.html        # Your UI
4   └── app.js            # Application logic
```

Step 2: Install Dependencies

```
1 npm install hyperswarm hypercore-crypto b4a
```

bash

What these do:

- **hyperswarm** - Peer discovery and networking
- **hypercore-crypto** - Cryptographic utilities
- **b4a** - Buffer/string conversions

Step 3: Build the UI

Replace `index.html` with:

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>P2P Chat</title>
7      <style>
8          * {
9              margin: 0;
10             padding: 0;
11             box-sizing: border-box;
12         }
13
14     body {
15         font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, 0
16         background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
17         height: 100vh;
18         display: flex;
19         align-items: center;
20         justify-content: center;
21     }
22
23     #titlebar {
24         -webkit-app-region: drag;
25         position: fixed;
26         top: 0;
27         left: 0;
28         right: 0;
29         height: 30px;
30         background: rgba(0, 0, 0, 0.2);
31         display: flex;
32         align-items: center;
33         padding: 0 10px;
34         z-index: 1000;
35     }
36
```

```
30
31
32
33
34
35
36
37     .container {
38         width: 90%;
39         max-width: 800px;
40         height: 80vh;
41         background: white;
42         border-radius: 20px;
43         box-shadow: 0 20px 60px rgba(0, 0, 0, 0.3);
44         display: flex;
45         flex-direction: column;
46         overflow: hidden;
47     }
48
49
50     /* Setup Screen */
51     #setup {
52         flex: 1;
53         display: flex;
54         flex-direction: column;
55         align-items: center;
56         justify-content: center;
57         padding: 3rem;
58         text-align: center;
59     }
60
61     #setup h1 {
62         color: #667eea;
63         margin-bottom: 1rem;
64         font-size: 2.5rem;
65     }
66
67     #setup p {
68         color: #666;
69         margin-bottom: 2rem;
70         font-size: 1.1rem;
71     }
72
73     .btn {
74         background: #667eea;
75         color: white;
76         border: none;
77         padding: 1rem 2rem;
```

```
78     font-size: 1.1rem;
79     border-radius: 10px;
80     cursor: pointer;
81     transition: all 0.3s;
82     font-weight: 600;
83   }

84   .btn:hover {
85     background: #5568d3;
86     transform: translateY(-2px);
87     box-shadow: 0 5px 15px rgba(102, 126, 234, 0.4);
88   }

89   .divider {
90     margin: 1.5rem 0;
91     color: #999;
92   }

93   #join-form {
94     display: flex;
95     gap: 0.5rem;
96     width: 100%;
97     max-width: 500px;
98   }

99   #join-form input {
100    flex: 1;
101    padding: 1rem;
102    border: 2px solid #e0e0e0;
103    border-radius: 10px;
104    font-size: 1rem;
105    transition: border 0.3s;
106  }

107  #join-form input:focus {
108    outline: none;
109    border-color: #667eea;
110  }

111  /* Chat Screen */
112  #chat {
113    font-size: 1.1rem;
114    border-radius: 10px;
115    cursor: pointer;
116    transition: all 0.3s;
117    font-weight: 600;
118  }
```

```
119     flex: 1;
120     display: none;
121     flex-direction: column;
122 }
123
124 #chat.active {
125   display: flex;
126 }
127
128 .chat-header {
129   background: #667eea;
130   color: white;
131   padding: 1.5rem;
132   display: flex;
133   justify-content: space-between;
134   align-items: center;
135 }
136
137 .chat-header h2 {
138   font-size: 1.3rem;
139 }
140
141 .chat-info {
142   font-size: 0.9rem;
143   opacity: 0.9;
144 }
145
146 .topic-display {
147   background: rgba(255, 255, 255, 0.2);
148   padding: 0.5rem 1rem;
149   border-radius: 5px;
150   font-family: monospace;
151   font-size: 0.85rem;
152   margin-top: 0.5rem;
153   word-break: break-all;
154 }
155
156 #messages {
157   flex: 1;
158   padding: 1.5rem;
159   overflow-y: auto;
```

```
160     background: #f5f5f5;
161 }
162
163 .message {
164     margin-bottom: 1rem;
165     animation: slideIn 0.3s;
166 }
167
168 @keyframes slideIn {
169     from {
170         opacity: 0;
171         transform: translateY(10px);
172     }
173     to {
174         opacity: 1;
175         transform: translateY(0);
176     }
177 }
178
179 .message-author {
180     font-weight: 600;
181     color: #667eea;
182     margin-bottom: 0.25rem;
183 }
184
185 .message-content {
186     background: white;
187     padding: 0.75rem 1rem;
188     border-radius: 10px;
189     box-shadow: 0 2px 5px rgba(0, 0, 0, 0.05);
190 }
191
192 .message.own .message-author {
193     color: #764ba2;
194 }
195
196 .message.own .message-content {
197     background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
198     color: white;
199 }
200
201
```

```
202     #message-form {
203         padding: 1.5rem;
204         background: white;
205         border-top: 1px solid #e0e0e0;
206         display: flex;
207         gap: 0.5rem;
208     }
209
210     #message-input {
211         flex: 1;
212         padding: 1rem;
213         border: 2px solid #e0e0e0;
214         border-radius: 10px;
215         font-size: 1rem;
216     }
217
218     #message-input:focus {
219         outline: none;
220         border-color: #667eea;
221     }
222
223     .hidden {
224         display: none !important;
225     }
226 
```

</style>

```
227     <script type="module" src="./app.js"></script>
228 </head>
229 <body>
230     <div id="titlebar">
231         <pear-ctrl></pear-ctrl>
232     </div>
233
234     <div class="container">
235         <!-- Setup Screen -->
236         <div id="setup">
237             <h1>🍐 P2P Chat</h1>
238             <p>Secure, serverless, peer-to-peer messaging</p>
239
240             <button class="btn" id="create-btn">Create Chat Room</button>
241
242             <div class="divider">or</div>
243     </div>
244 
```

```
243      <form id="join-form">
244        <input
245          type="text"
246          id="topic-input"
247          placeholder="Paste chat room topic..."
248          required
249        >
250        <button type="submit" class="btn">Join</button>
251      </form>
252    </div>
253
254
255    <!-- Chat Screen -->
256    <div id="chat">
257      <div class="chat-header">
258        <div>
259          <h2>Chat Room</h2>
260          <div class="chat-info">
261            <span id="peer-count">0</span> peer(s) connected
262          </div>
263          <div class="topic-display" id="topic-display"></div>
264        </div>
265      </div>
266
267      <div id="messages"></div>
268
269      <form id="message-form">
270        <input
271          type="text"
272          id="message-input"
273          placeholder="Type a message..."
274          autocomplete="off"
275        >
276        <button type="submit" class="btn">Send</button>
277      </form>
278    </div>
279  </div>
</body>
</html>
```

Key UI features:

- Clean, modern design
 - Draggable window (using `pear-ctrl`)
 - Smooth animations
 - Responsive layout
 - Clear visual hierarchy
-
-

Step 4: Add Application Logic

Replace `app.js` with:

```
1  /* global Pear */
2  import Hyperswarm from 'hyperswarm'
3  import crypto from 'hypercore-crypto'
4  import b4a from 'b4a'
5
6  // Pear utilities
7  const { teardown, updates } = Pear
8
9  // Initialize swarm
10 const swarm = new Hyperswarm()
11
12 // Cleanup on exit
13 teardown(() => swarm.destroy())
14
15 // Enable hot reload during development
16 updates(() => Pear.reload())
17
18 // Track connections and messages
19 const connections = new Set()
20
21 // DOM elements
22 const setupScreen = document.querySelector('#setup')
23 const chatScreen = document.querySelector('#chat')
24 const createBtn = document.querySelector('#create-btn')
```

```
25 const joinForm = document.querySelector('#join-form')
26 const topicInput = document.querySelector('#topic-input')
27 const topicDisplay = document.querySelector('#topic-display')
28 const messagesContainer = document.querySelector('#messages')
29 const messageForm = document.querySelector('#message-form')
30 const messageInput = document.querySelector('#message-input')
31 const peerCount = document.querySelector('#peer-count')
32
33 // Event Listeners
34 createBtn.addEventListener('click', createRoom)
35 joinForm.addEventListener('submit', (e) => {
36   e.preventDefault()
37   const topic = topicInput.value.trim()
38   if (topic) joinRoom(topic)
39 })
40 messageForm.addEventListener('submit', (e) => {
41   e.preventDefault()
42   const message = messageInput.value.trim()
43   if (message) sendMessage(message)
44 })
45
46 // Handle new peer connections
47 swarm.on('connection', (peer) => {
48   connections.add(peer)
49   updatePeerCount()
50
51   // Get peer identifier (first 6 chars of public key)
52   const peerId = b4a.toString(peer.remotePublicKey, 'hex').slice(0, 6)
53
54   // Handle incoming messages
55   peer.on('data', (data) => {
56     const message = data.toString()
57     addMessage(peerId, message, false)
58   })
59
60   // Handle disconnections
61   peer.on('close', () => {
62     connections.delete(peer)
63     updatePeerCount()
64   })
65
66
```

```
67     peer.on('error', (err) => {
68         console.error('Peer error:', err)
69         connections.delete(peer)
70         updatePeerCount()
71     })
72 }
73
74 // Update peer count display
75 swarm.on('update', updatePeerCount)
76
77 /**
78 * Create a new chat room
79 */
80 async function createRoom() {
81     const topicBuffer = crypto.randomBytes(32)
82     const topicString = b4a.toString(topicBuffer, 'hex')
83     await joinRoom(topicString)
84 }
85
86 /**
87 * Join an existing chat room
88 */
89 async function joinRoom(topicString) {
90     const topicBuffer = b4a.from(topicString, 'hex')
91
92     // Join the swarm
93     const discovery = swarm.join(topicBuffer, {
94         server: true, // Accept connections
95         client: true // Search for peers
96     })
97
98     // Wait for the topic to be announced
99     await discovery.flushed()
100
101    // Update UI
102    setupScreen.classList.add('hidden')
103    chatScreen.classList.add('active')
104    topicDisplay.textContent = topicString
105    messageInput.focus()
106
107    addSystemMessage(`🎉 Joined chat room! Share this topic with friends:`)
```

```
108     addSystemMessage(topicString)
109 }
110
111 /**
112  * Send a message to all connected peers
113 */
114 function sendMessage(message) {
115     // Display locally
116     addMessage('You', message, true)
117
118     // Send to all peers
119     for (const peer of connections) {
120         try {
121             peer.write(message)
122         } catch (err) {
123             console.error('Error sending message:', err)
124         }
125     }
126
127     // Clear input
128     messageInput.value = ''
129 }
130
131 /**
132  * Add a message to the chat
133 */
134 function addMessage(author, content, isOwn = false) {
135     const messageEl = document.createElement('div')
136     messageEl.className = `message ${isOwn ? 'own' : ''}`
137
138     messageEl.innerHTML =
139         `

${author}</div>
140         <div class="message-content">${escapeHtml(content)}</div>
141         `
142
143     messagesContainer.appendChild(messageEl)
144     messagesContainer.scrollTop = messagesContainer.scrollHeight
145 }
146
147 /**
148  * Add a system message


```

```
149  /*
150   function addSystemMessage(content) {
151     const messageEl = document.createElement('div')
152     messageEl.className = 'message'
153     messageEl.innerHTML =
154       <div class="message-content" style="background: #f0f0f0; color: #666; f
155         ${escapeHtml(content)}
156       </div>
157     `
158
159     messagesContainer.appendChild(messageEl)
160     messagesContainer.scrollTop = messagesContainer.scrollHeight
161   }
162
163 /**
164 * Update peer count display
165 */
166 function updatePeerCount() {
167   peerCount.textContent = connections.size
168 }
169
170 /**
171 * Escape HTML to prevent XSS
172 */
173 function escapeHtml(text) {
174   const div = document.createElement('div')
175   div.textContent = text
176   return div.innerHTML
177 }
```

Step 5: Test Your App

Launch the App

```
1 pear run --dev .
```

bash

The app opens with the setup screen:



Create a Room

1. Click "**Create Chat Room**"
2. You'll see a hexadecimal topic displayed
3. Copy this topic

Join from Another Instance

Open a new terminal and run:

```
1 cd p2p-chat  
2 pear run --dev .
```

bash

1. Paste the topic into the input field
2. Click "**Join**"
3. The peers will connect automatically!

Start Chatting

Type messages in either window and watch them appear in both!



Understanding the Code

Key Components

1. Swarm Initialization

javascript

```
1 const swarm = new Hyperswarm()  
2 teardown(() => swarm.destroy())
```

Creates the networking instance and ensures cleanup when the app closes.

2. Creating a Room

javascript

```
1 const topicBuffer = crypto.randomBytes(32)  
2 swarm.join(topicBuffer, { server: true, client: true })
```

Generates a random 32-byte topic and joins the swarm as both server (accepting connections) and client (searching for peers).

3. Connection Handling

javascript

```
1 swarm.on('connection', (peer) => {  
2   connections.add(peer)  
3   peer.on('data', (data) => {  
4     // Handle incoming messages  
5   })  
6 })
```

Listens for new peer connections and handles their messages.

4. Broadcasting Messages

javascript

```
1 for (const peer of connections) {  
2   peer.write(message)  
3 }
```

Sends messages to all connected peers.

Enhancements to Try

1. Add Username Support

```
1 // Store username
2 let username = prompt('Enter your username:')
3
4 // Send as JSON
5 peer.write(JSON.stringify({
6   type: 'message',
7   author: username,
8   content: message,
9   timestamp: Date.now()
10 }))
```

javascript

2. Persistent Chat History

Use Hypercore to store messages:

```
1 import Hypercore from 'hypercore'
2
3 const messageLog = new Hypercore('./chat-history')
4
5 // Append messages
6 await messageLog.append(JSON.stringify({
7   author,
8   content,
9   timestamp: Date.now()
10 }))
11
12 // Read history on startup
13 for (let i = 0; i < messageLog.length; i++) {
14   const msg = JSON.parse(await messageLog.get(i))
15   addMessage(msg.author, msg.content)
16 }
```

javascript

3. File Sharing

Add drag-and-drop file sharing:

```

1  document.addEventListener('drop', async (e) => {
2      e.preventDefault()
3      const file = e.dataTransfer.files[0]
4      const buffer = await file.arrayBuffer()
5
6      // Send file to peers
7      for (const peer of connections) {
8          peer.write(JSON.stringify({
9              type: 'file',
10             name: file.name,
11             data: Buffer.from(buffer).toString('base64')
12         }))
13     }
14   })

```

javascript

4. Typing Indicators

```

1  messageInput.addEventListener('input', () => {
2      for (const peer of connections) {
3          peer.write(JSON.stringify({ type: 'typing' }))
4      }
5  })

```

javascript

Troubleshooting

Peers Won't Connect

Problem: Two instances can't find each other

Solutions:

1. Verify both instances use the exact same topic (copy-paste carefully)
2. Wait 10-15 seconds for DHT discovery
3. Check firewall settings (UDP must be allowed)
4. Try on the same network first

Messages Not Appearing

Problem: Messages sent but not received

Debug:

```
1  peer.on('data', (data) => {  
2      console.log('Received:', data.toString())  
3      // Your message handling  
4  })
```

javascript

Check both sender and receiver logs.

App Crashes on Startup

Problem: Module import errors

Solution:

```
1  # Reinstall dependencies  
2  rm -rf node_modules package-lock.json  
3  npm install hyperswarm hypercore-crypto b4a
```

bash

What You Learned

- ✓ Setting up a Pear desktop application
- ✓ Using Hyperswarm for peer discovery
- ✓ Managing P2P connections and state

- Building an interactive UI with vanilla JS
 - Broadcasting messages to multiple peers
 - Handling connection lifecycle events
-
-

Next Steps

Add Persistence

Learn to store messages using Hypercore
[Data Structures Tutorial →](#)

Deploy Your App

Package and share your chat app [Releasing a Pear App →](#)

Use React/Vue

Build with modern frameworks [React with Pear →](#)

Explore Examples

See more advanced P2P apps [Examples Gallery →](#)

Challenge: Extend Your App

Try adding these features:

- [] Message timestamps
- [] Emoji support
- [] Dark mode toggle
- [] Sound notifications

- [] Message persistence with Hypercore
- [] Private messaging (direct peer connections)
- [] Rich text formatting
- [] Image sharing

Share your creations in our [Discord community!](#)

 [Edit this page on GitHub](#)

Previous page

[Quick Start](#)

Next page

[Chat with Persistence](#)