Media Computing Group

RWTH AACHEN UNIVERSITY

AsyncNetwork

Cocoa & iOS Networking Evolved

Open Source (MIT License)
github.com/jdiehl/async-network

Jonathan Diehl
jonathan.diehl@rwth-aachen.de

Client/Server

One Server

Listen to incoming connections

↳ Listening Socket

Many Clients

Initiate a connection to the server

↳ Connection Socket

Peer-to-Peer

Initiate a connection to another peer

Listen to incoming connections

↳ Connection Sockets + Listening Socket

Many Peers

Networking Basics

- Sockets are endpoints for communication
- All sockets require a socket address (IP & port)
  - Listening sockets listen on the interface & port
  - Connection socket connect to the host & port

Port	IP Address
1...65535	IPv4, IPv6
many reserved ports, > 10.000 is fairly save	loopback: 127.0.0.1 can listen to all

Networking Basics

- A connected socket has two data-streams:
  - Input stream & output stream
- Stream data must be encoded
  - Data format (string or binary)
  - Data delimiter (designated character or data length)

Networking and the Web

- WebSocket: HTTP Extension for Socket Networking
- Supported by most web browsers and web backends
  - Chrome, Safari, Firefox, Node, Ruby on Rails, Python, PHP...
- Cocoa / iOS WebSocket Client
  - https://github.com/square/SocketRocket
- Cocoa / iOS WebSocket Server

Network Support

Built-in

GameCenter

Foundation

CFNetwork

BSD Sockets

Third Party

AsyncSocket

ThoMoNetworking

AsyncNetwork

...

GameCenter

- Real-Time Matches
  - Peer-to-Peer
  - Must use GameCenter to initiate connections
- Hosted Matches
  - Client/Server
  - Must implement own networking
- Turn-Based Matches
  - No networking (shared data store)

Foundation

- Distributed Objects
  - Objects are synchronized across processes (possibly across remote devices)
  - Must use Bonjour, high overhead
- URL Loading System
  - Connect to and load data from HTTP servers
- Bonjour
  - Discover peers

CFNetwork / BSD Sockets

```
    // Create the listening Socket
CFSocketContext context = {0, (__bridge void *) self, NULL, NULL, NULL};
socket = CFSocketCreate(kCFAllocatorDefault, AF_INET,  SOCK_STREAM, 0, kCFSocketAcceptCallBack, &CallBack, &context);
static const int yes = 1;
(void) setsockopt(CFSocketGetNative(socket), SOL_SOCKET, SO_REUSEADDR, (const void *) &yes, sizeof(yes));

// Bind the socket socket to an IP address and port
struct sockaddr_in addr;
memset(&addr, 0, sizeof(addr));
addr.sin_len = sizeof(addr);
addr.sin_family = AF_INET;
addr.sin_port = htons(0);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
if (kCFSocketSuccess != CFSocketSetAddress(socket, (__bridge CFDataRef) [NSData dataWithBytes:&addr length:sizeof(addr)]))
return NO;

// Set up the run loop for the socket
CFRunLoopSourceRef source = CFSocketCreateRunLoopSource(kCFAllocatorDefault, socket, 0);
CFRunLoopAddSource(CFRunLoopGetCurrent(), source, kCFRunLoopCommonModes);
CFRelease(source);

// This function is called by CFSocket when a new connection comes in
static void CallBack(CFSocketRef socket, CFSocketCallBackType type, CFDataRef address, const void *data, void *info) {
    CFSocketNativeHandle nativeSocketHandle = (CFSocketNativeHandle)data;
    CFReadStreamRef readStream = NULL; CFWriteStreamRef writeStream = NULL;

    // Create the Input and Output Streams
    CFStreamCreatePairWithSocket(kCFAllocatorDefault, nativeSocketHandle, &readStream, &writeStream);
    if (!readStream || !writeStream)
           return (void) close(nativeSocketHandle);
    CFReadStreamSetProperty(readStream, kCFStreamPropertyShouldCloseNativeSocket, kCFBooleanTrue);
    CFWriteStreamSetProperty(writeStream, kCFStreamPropertyShouldCloseNativeSocket, kCFBooleanTrue);

    // Create a connection object
    MyConnection *connection = [[MyConnection alloc] initWithInputStream:(__bridge NSInputStream *)readStream
    [connection open];
    if (readStream) CFRelease(readStream);
    if (writeStream) CFRelease(writeStream);
}
```

from Mac Developer Library: Example "CocoaEcho"

AsyncSocket

- High-Level socket abstraction
- Very simple to use
- Supports Grand Central Dispatch

AsyncSocket

```
// Create listening socket
socket = [[GCDAsyncSocket alloc] initWithDelegate:self
          delegateQueue:dispatch_get_main_queue()];

// Bind socket to port
NSError *error;
if (![self.listenSocket acceptOnPort:self.port error:&error])
   return;

// New connection
- (void)socket:(GCDAsyncSocket *)sock
   didAcceptNewSocket:(GCDAsyncSocket *)newSocket;
{
   // Create connection object
   MyConnection *connection;
   connection = [MyConnection connectionWithSocket:newSocket];
}
```

ThoMoNetworking

- Automatic server discovery and connection (Bonjour)
- Automatic Object Encoding (NSCoding)
- Simple communication protocol (read & write)

ThoMoNetworking

```
// Setup the server
- (void)setupServer
{
   myServer = [[ThoMoServerStub alloc] initWithProtocolIdentifier:@"helloThoMo"];
   [myServer start];
}

// Send a string to all our clients
- (void)sendString:(NSString *)aString
{
   [myServer sendToAllClients:aString];
}

// Setup the client
- (void)setupClient
{
   myClient = [[ThoMoClientStub alloc] initWithProtocolIdentifier:@"helloThoMo"];
   [myClient setDelegate:self];
   [myClient start];
}

// This is our delegate method where we receive data from our server
- (void)client:(ThoMoClientStub *)theClient didReceiveData:(id)theData
   fromServer:(NSString *)aServerIdString;
{
   NSLog(@"%@ sent me a message: %@", aServerIdString, theData);
}
```

AsyncNetwork

- Automatic server discovery and connection (Bonjour)
- Automatic Object Encoding (NSCoding)
- Simple communication protocol (read, write, request, command byte)
- Broadcasting
- Based on GCDAsyncSocket

AsyncNetwork

```
// Setup the server
- (void)setupServer {
   // Create and start the server
   self.server = [AsyncServer new];
   self.server.serviceName = @"My Service";
   self.server.delegate = self;
   [self.server start];
}

// Client did connect
- (void)server:(AsyncServer *)theServer didConnect:(AsyncConnection *)connection {
   // Message the client
    [self.server sendObject:@"Hello Client"];
}

// Setup client
- (void)setupClient {
   // Create and start the client
   self.client = [AsyncClient new];
   self.client.delegate = self;
   [self.client start];
}

- (void)client:(AsyncClient *)theClient didReceiveCommand:(AsyncCommand)command
   object:(id)object connection:(AsyncConnection *)connection {
   // Log the message
   NSLog(@"%@", object);
}
```

AsyncNetwork Requests

```
// send a request to the server
[AsyncRequest fireRequestWithHost:@"192.168.0.1"
                             port:10000
                          command:0
                           object:@"Hello"
                     responseBlock:^(id response, NSError *error)
{
   if (error) return;
   NSLog(@"%@", response);
}];
```

AsyncNetwork Broadcasting

```
- (void)setupBroadcaster {
   // Create and start the broadcaster
   self.broadcaster = [AsyncBroadcaster new];
   self.broadcaster.port = 10000;
   self.broadcaster.delegate = self;
   [self.broadcaster start];
}

- (void)broadcast {
   // Encode the string message
   NSString *message = @"Hello World\n";
   NSData *encodedMessage = [message dataUsingEncoding:NSUTF8StringEncoding];

   // Broadcast the encoded message
   [self.broadcaster broadcast:encodedMessage];
}

- (void)broadcaster:(AsyncBroadcaster *)theBroadcaster
     didReceiveData:(NSData *)data
            fromHost:(NSString *)host {
   // Decode the message
   NSString *message;
   message = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
   NSLog(@"%@", message);
}
```