R and Websockets

Bryan W. Lewis

blewis@paradigm4.com

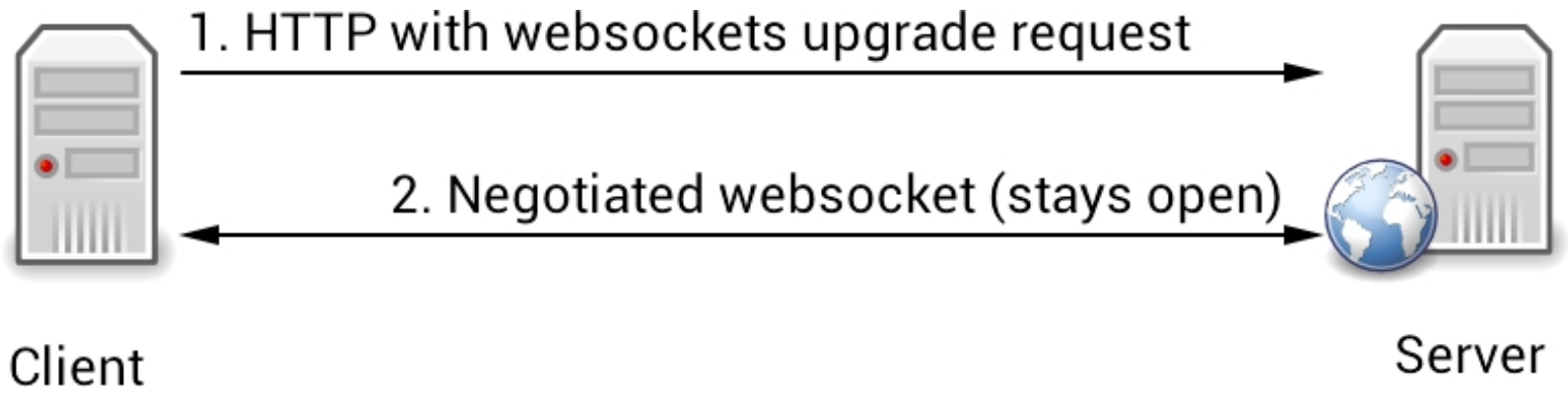
http://paradigm4.com

http://illposed.net

JSM 2012

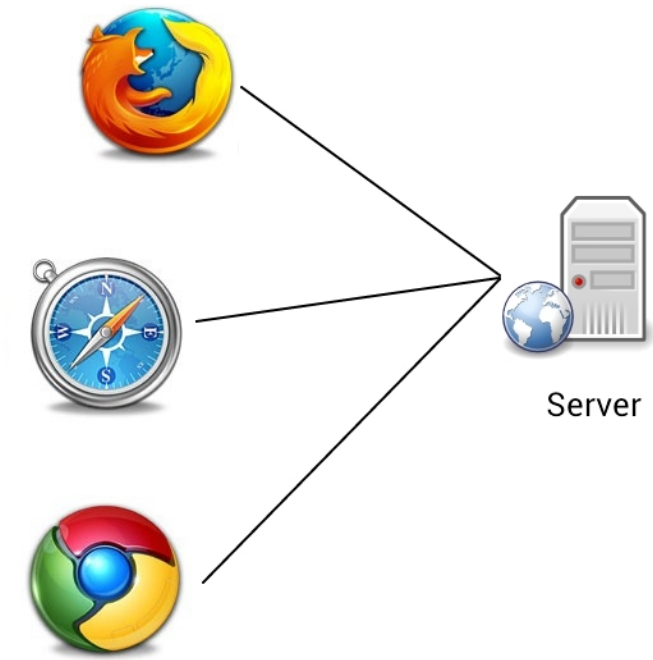
What are Websockets?

Websockets are normal bidirectional TCP sockets that begin life as a web (HTTP) connection.



What are Websockets?

Websocket servers are HTTP servers. They can serve mixtures of regular HTTP requests and websocket connections to one or more clients concurrently.



All modern web browsers on mobile and desktop systems, and many programming languages, support websockets.

Why are Websockets useful?

When there are frequent back and forth data exchanges between clients and a server, websockets:

- Significantly cut network overhead.
- Reduce server processing overhead.
- Simplify fast asynchronous updating of web clients (push).
- Simplify stateful couplings between servers and clients.

The `websockets` R package

The `websockets` package provides websocket client and server functions for R.

- The `websocket` function lets R talk to websocket servers.
- The `create_server` function lets R talk to websocket clients.

An R websocket server

is a TCP server, which listens on a port, together with four functions:

1. `webpage`
   Invoked for normal HTTP requests.
2. `established`
   Invoked after a new websocket connection is established.
3. `receive`
   Invoked after data is received from a websocket connection.
4. `closed`
   Invoked after a websocket connection is closed.

R websocket server

Multuple client connections: the `established, receive` and `closed` functions each have a parameter called `WS` that identifies the websocket connection.

Binary and text connections.

A server can be polled for activity:

- in an R loop with the `service` function, or
- serviced asynchronously from the main R event loop with the `daemonize` function.

R websocket server

Use the `websocket_write` function to send data to a specific websocket.

Use the `websocket_broadcast` function to send data to all the websockets connected to a server.

Multiple servers on different ports may run from one R session.

Hello World

```
text =
  "<html><body><script> try {
   var s = new WebSocket(\"ws://\" + window.location.host, \"R\");
   s.onopen = function(){socket.send(\"HELLO R\");};
   s.onmessage = function(msg){document.write(\"<h1>\"+msg.data);};
  } catch(ex) {} </script></body></html>"

html = function(socket,...) http_response(socket,content=text))
recv = function(DATA,WS,...)
       websocket_write(paste("You sent:",rawToChar(DATA)),WS)

w = create_server(webpage=html)
set_callback('receive', recv, w)

daemonize(w)
```

Crazy Simple R Chat (R part*)

```
w = create_server(webpage=static_text_service(htmldata))
f = function(DATA,WS,...)
{
  d = tryCatch(rawToChar(DATA),error=function(e) "")
  jpeg(file=itmp, width=400,height=400,quality=100)
  sink(ctmp,split=TRUE)
  tryCatch(eval(parse(text=d)), error = function(e) print(e))
  sink()
  dev.off()
  ans = paste(readLines(ctmp),collapse="\n")
  p = base64encode(readBin(itmp,what="raw",n=1e6))
  p = paste("\n",p,sep="")
  msg = sprintf("<b>Client %d says: </b>%s\n%s",WS$socket,d,ans)
  websocket_broadcast(toJSON(list(msg=msg, fig=p)),WS$server)
}
set_callback("receive",f,w)

ctmp = tempfile()
itmp = tempfile()
daemonize(w)
```

* The HTML part of the chat example, the R character variable above called “htmldata,” is on a later slide.

R Chat Example in Action

Multiple web clients chatting through a basic R console in the web.

R Chat | R Chat

localhost:7681

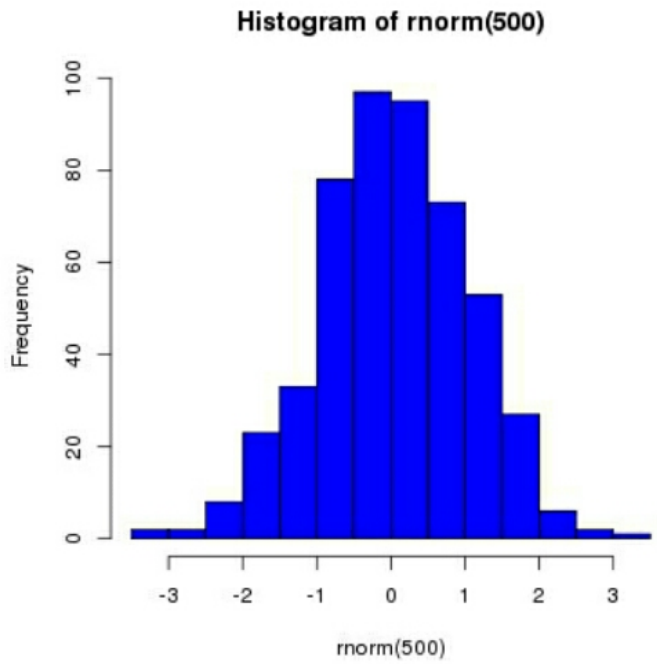
R Chat

Websocket connection established

Client 4 says: `hist(rnorm(500),col=4)`

Client 5 says: `print(head(iris))`

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```



Console:

10:02

Shiny R Web Framework

Shiny is new web application framework for R from RStuido.

Shiny makes it easy to produce beautiful, useful web applications from R without requiring coding in HTML/Javascript.

Shiny uses a reactive programming model that intuitively ties inputs and outputs together.

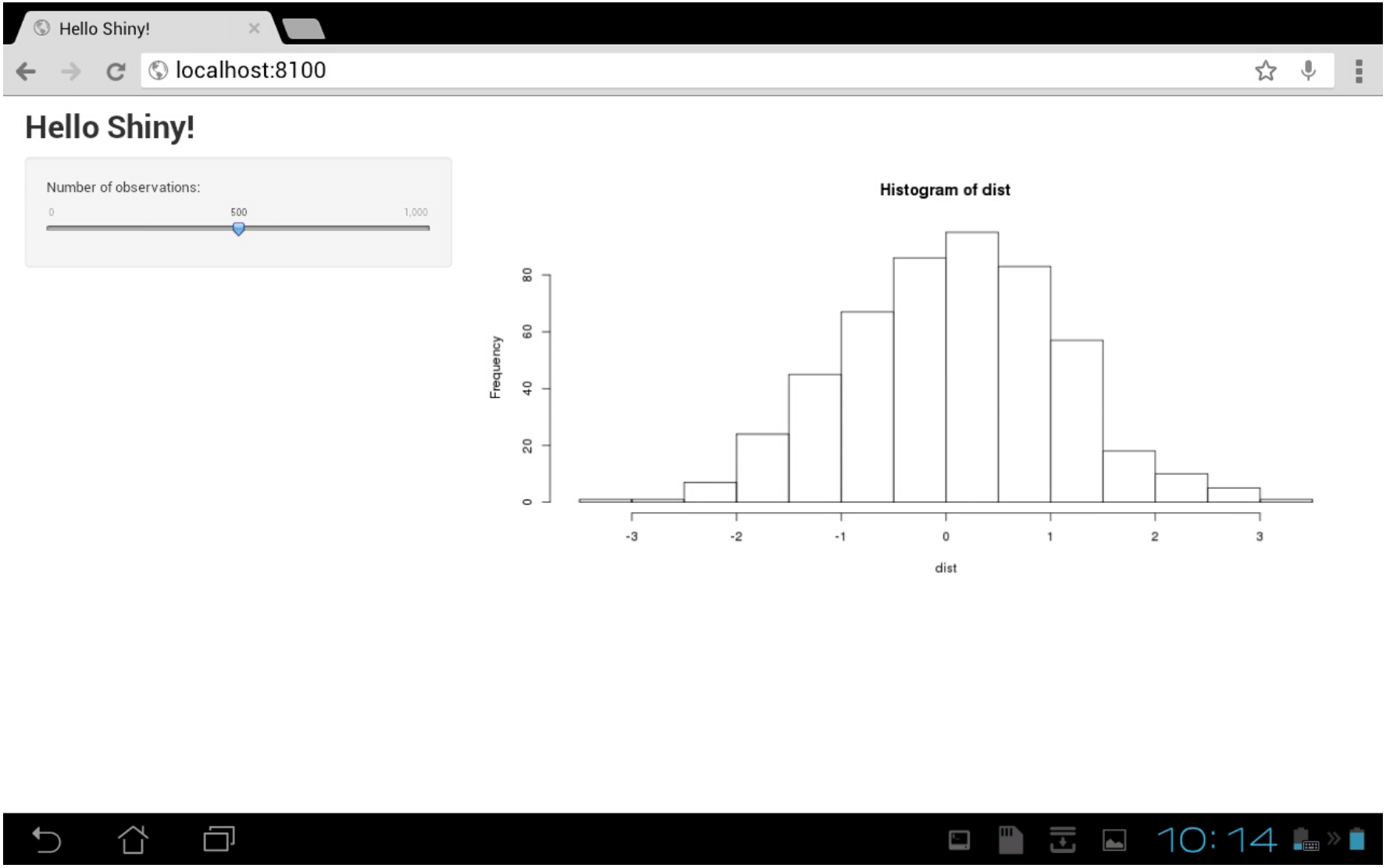
Reactive Model

The reactive plot depends on the `obs` variable below. Changes "just work" like you'd expect.

```
shinyUI(pageWithSidebar(
  headerPanel("Hello Shiny!"),
  sidebarPanel( sliderInput("obs",
                    "Number of observations:",
                    min = 0, max = 1000, value = 500)
  ),
  mainPanel( plotOutput("distPlot") )
))
```

```
shinyServer(
  function(input, output)
  {
    output["distPlot"] <- reactivePlot(
      function() dist <- rnorm(input["obs"]) hist(dist) )
  })
```

Shiny Example in Action



Takeaway

The websockets package is a good foundation for speedy, collaborative, cross-platform web applications in R.

The websockets package is on CRAN now.

Contact RStudio for preview access to the shiny package. Do it.

Appendix:HTML for R chat example...slide 1

```
<html><head><title>R Chat</title></head>
<body>
<style type="text/css">
.fig {
  float: right; margin: 0 0 10px 10px; clear: right;
}
div.scroll {
  height: 300px; width: 500px; border: 1px solid #666;
  padding: 8px; overflow-y: scroll; word-wrap: break-word;
}
input[type="text"], textarea {
  padding: 0; margin: 0; width: 518px; border: 1px solid;
}
</style>
```

HTML for R chat example...slide 2

```
<script>
var socket = new WebSocket("ws://"+window.location.host, "Rchat");
var last = "";
try {
  socket.onopen = function() {
    document.getElementById("chat").innerHTML = "<b>Websocket
        connection established</b>";
  }
  socket.onmessage = function got_packet(msg) {
    var ans = JSON.parse(msg.data);
    var log = document.getElementById("chat").innerHTML;
    log = log + "<pre>" + ans.msg + "</pre>";
    document.getElementById("chat").innerHTML = log;
    if(ans.fig != null) document.getElementById("plot").src = ans.fig;
  }
  socket.onclose = function(){
    document.getElementById("chat").innerHTML = "<b>Websocket
        connection closed</b>";
  }
} catch(ex) {document.getElementById("chat").textContent = "Error: " +
    ex;}
```

HTML for R chat example...slide 3

```
function checkKey(evt)
{
  if(evt.keyCode == 13) {
    last = document.getElementById("msg").value;
    socket.send(last);
    document.getElementById("msg").value = "";
    document.getElementById("chat").scrollTop = 10 + document.
        getElementById("chat").scrollHeight;
    return false;
  }
  else if(evt.keyCode == 38) {
    document.getElementById("msg").value = last;
    return false;
  }
  return true;
}
</script>
```

HTML for R chat example...slide 4 (last one)

```
<h2>R Chat</h2>
<div>
<img class="fig" id="plot" alt="R Plot" width="400" height="400" />
<div id="chat" class="scroll"/>
</div>
<h3>Console:</h3>
<input type="text" id="msg" value="" size="80" maxlength="150" onkeyup
    ="return checkKey(event);"/>
</body>
</html>
```