MBDD: Client Side Operation

1. Operation

The client side of MBDD runs as a socks proxy on a user’s machine. To run the proxy, enter the following at the command prompt, in the directory that contains the mbdd classes dir:

Prompt:> java mbdd.client.ClientProxy

The client relies on the file “client.props” which should be in the current working directory when running. Also, just like the server, the client must have the appropriate multicast classes in its classpath. If they are not present in the classpath already, use the included classes.jar at the prompt with this command instead:

Prompt:> java –classpath classes.jar:. mbdd.client.ClientProxy

If the client proxy is run correctly, a small window with an “exit proxy” button should appear on the desktop.

To use the proxy, set your web browser to use a socks proxy with “localhost” as the address and 1080 as the port. The browser will now operate through the proxy, and there should be no perceived difference when browsing, except when requesting from an MBDD-enabled server. In this case, the proxy will detect the multicast-enabled server and operate in MBDD-enabled mode.

2. Organization

The client side is split into two major components: the socks proxy, and the actual MBDD client. The proxy is located in the “socks” folder and the MBDD client is in “mbdd/client.” When mbdd.client.ClientProxy is run, it creates a socks.ProxyServer object, which is the main class for the socks proxy.

The socks proxy operates like a standard proxy for the most part, but it contains an mbdd.client.Client object. When ProxyServer gets a client request to a server it has not seen before, ProxyServer checks if the server is MBDD enabled by sending http HEAD requests to a number of predefined ports. If enabled, the server replies with the multicast push/pull addresses and ports. This information is stored so that subsequent requests to the same server don’t require sending a HEAD request to find multicast info. The socks
proxy then tells its Client object to connect to the multicast channels, and the proxy lets
the Client object handle requests to the enabled server. All the interaction between the
proxy and the MBDD Client class is in ProxyServer’s onConnect method. No other
methods in the socks package were changed from regular socks operation. If the server is
found to not have MBDD, then the client operates through the socks proxy as normal,
without any influence of the MBDD client.

Since the socks proxy operation has been covered, let’s now focus on the MBDD client
package (in mbdd/client). The Client class provides the high-level access to the package.
It consists of one UnicastChannel object, one MulticastPushChannel object and one
MulticastPullChannel object. The job of the Client class is to accept requests, and fetch
the requested data from the appropriate channel. It has two public methods: connect and
request. Connect is called by the socks ProxyServer when a request to a new MBDD-
enabled server is made. It simply passes the connection information to the push and pull
channel objects, which then attempt to connect.

Client.request is called by ProxyServer, after a connection has already been established.
First, Client.request sends the request to the MulticastPushChannel object. It expects the
channel to know what is being broadcast, and return null immediately if the requested
document is not on the channel. If the request fails on the push Channel, it is then sent as
a unicast request. The pull channel listens for the document, and the channel that sees the
document first returns it to the Client, which passes it back to the ProxyServer.

That is the basic operation of the client. However, the Push channel needs to be
examined in more detail. It was coded primarily by CASE, but changes were needed in
order to integrate with the MBDD client.

3. Multicast Push Channel

The MulticastPushChannel object basically wraps a CASE MulticastPushManager object.
It uses the client.props file to set up and start a new MulticastPushManager object each
time it is asked to connect to a new server. Documents are fetched by simply returning
the result of pushManager.getContentofDocument(doc). getContentofDocument creates
an AppDocRequest object and returns the result of

Significant changes had to be made to AppDocRequest.getContentofDoc. As mentioned
before, the Client expects the push channel to return if the document isn’t on the channel
broadcast index (or in cache), so the unicast channel can be used. The operation of
getContentofDoc is as follows: First, it checks the cache, if enabled. Next, it checks the
current broadcast index. If the requested doc is in the index, it waits for the doc to arrive,
or for a new index. If a new index is heard first, it checks the new index. If found on the
new index, we assume that the document was missed in the previous broadcast cycle
because the request arrived late. It then listens for one complete cycle, and finally returns null if the document wasn’t found.

This means that broadcast listening capability had to be added somewhere. In addition, AppDocRequest needed to check the contents of the current index, and check if the index was updated. This functionality was added to the RequestManager (and RequestManagerWithoutCache) class. The following methods were added: parseHeader, notifyIndexChange, and isInIndex. ParseHeader is called whenever a new broadcast index is received on the push channel. When parseHeader completes, it calls notifyIndexChange, which notifies every running AppDocRequest object that there is a new broadcast index. isInIndex is called by AppDocRequest when it needs to check for a doc in the current index.

The broadcast index is pieced together by the Receiver class. The parsePacket method catches packets with a “type” of 3, and adds the page to the incoming index. When the final page is received, parsePacket calls RequestManager.parseHeader with the new index string.

All other classes in mbdd/client/push and mbdd/client/push/cache were created by CASE and have not undergone any major changes.

4. Future Work and Questions

- The CASE push channel code does not work on Windows XP, but is fine on linux. I can’t explain why, but this needs to be resolved.

- We have not yet implemented the CASE initialization protocol option, where the MBDD server runs on the standard http port and handles communication to all clients. In our setup, the web server is unchanged and runs on the standard port. Our server reverse-proxy runs on port 8081, and the client socks-proxy attempts to contact this port whenever it hears a request to a new server. If it finds the server is not MBDD enabled, or if the client does not have the MBDD socks proxy, then requests go to the regular web server as normal.