Python has a number of easy to implement http proxy solutions. A convenient list is maintained at: `http://proxies.xhaus.com/python/`. Most of these proxies extend the httplib python library, which provides a simple interface for setting up stable http connections. Python also has a native socket library for TCP/UDP flows, as well as a fair count of other choices for HTTP connections.

By far the most rebust and extendible library is Twisted Proxy. Twisted provides an easy to deploy and heavily scriptable threaded http proxy server. By overrideing its many default classes, you can define a simple proxy in just a few lines. At the other end of the spectrum lies the possiblity to do pretty much everything yourself. By overriding the client processor class, you can extend the object just enough to fit your needs, while keeping all the rest of the default functionality that it guarantees.

Twisted allowed me to set up a http proxy server that issues HTTP head requets on all get content before accepting a connection. Then, depending on the content-length dictated by the server, the proxy can decide whether to process the request as usual, or attempt to communicate with other proxies in its list of peers. But what do we do when a router has decided too enlist its neighbors for help? How is the packet divided so that each chunk finishes downloading in a reasonably close fashion?

0.1 Packet Segmentation Algorithm

The first, naieve approach, was fairly straight forward. Given a peer network of n routers, give each router:

$$\frac{FILESIZE}{n} bytes$$

This has a major shortcoming however: different routers download at different rates, so the file download isn't complete until the slowest router has downloaded and transmited its chunk back to the host. So the download time becomes the download speed of the slowest router. A better approach is to divy up the file into chunks using a function of router bandwidth and total bandwidth.

A consideration to keep in mind is that the host router may not be directly connected to every n routers. It could be the case that the host has 3 peers, who each have 3 other peers in wireless communication range. When the router divies up these chunks, the advertised bandwidth of a router that he is directly connected too should reflect the average bandwidth of all of that routers peers. So if A connects to B who has 2mbps of bandwidth, but B can talk to C and D, who each have a 10mbps connection, B should advertise to A that his bandwidth is

$$\frac{B + C + D}{3}$$

mbps. This will help A more accurately decide how to manage the file segmenting. When B is passed this chunk, he can do the same with each peer in his network.

The problem of overlapping neighbors does immediately become an issue. I plan to address this at the implementation level, and assume without loss of generality that this can be accomplished. My plan so far is to have the host generate a random session key, and pass the key along when it communicates with its neighbors during the negotiation period. Each neighbor will store this key, and until a cancel request is sent from the host, the router will reject any negotiationg requests whose key matches their stored key. This way, a router will not commit its bandwidth too two different neighbors on the same file download.

The second issue is link cost. Presumably, routers will communicate with eachother over 802.11 b/g at 54 mbps. As part of the negotiating process, each router will consider their peers estimated download time with the transportation time (over the 802.11 link), in order to determine if futher segmentation would yield a significant download time increase.

Pseudo code of the algorithm:

for all peer in neighbors(host) **do**
  $netBandwidth \leftarrow netBandwidth + peer.bandwidth$
end for
for all peer in neighbors(host) **do**
  $Chunk_{peer} \leftarrow \frac{peer.bandwidth}{netBandwidth} \times fileSize$
end for
for all peer in neighbors(host) **do** Delegate $Chunk_{peer}$
  $AmountRemaining \leftarrow AmountRemaining - Chunk_{peer}$
end for
Issue HTTP GET for $AmountRemaining$