Paper Analysis

Anycast-Aware transport for content delivery networks

Anycast is a routing scheme where multiple hosts are mapped to a single IP address and the data is routed to the nearest host as viewed by the routing topology of the network. This routing scheme is well suited for connectionless service (UDP) and less so for TCP. However, with the increase in the multimedia data and size of files being transferred over the networks, such a scheme can be highly useful in content delivery networks providing such large content. But the main hurdle to this is the connection oriented nature of TCP (especially for large files). In an anycast network, the destination keeps on changing frequently (because of change in network conditions) and thus would lead to frequent TCP connection disruptions. In this paper, the authors have targeted this problem so that a client can continue to download the content from where it left off due to a server change.

The paper is very well organized and starts off by discussing the current architecture of CDNs, which rely on DNS to redirect client requests to an edge server of the CDN. As pointed out by the authors, the scheme has several drawbacks, for example, routing will be done according to the proximity of an edge server to the requesting DNS server instead of the client. Second, caching at DNS servers will deflect from the goal of routing to the least loaded server. So why not route the request on the basis of proximity of the client to the server? There are 2 major concern of such a network: 1. lack of any load balancing mechanism and 2. connection disruptions in downloads.

The second problem is the crux of this paper and the authors start off by telling us why a static binding (binding the client to a particular edge server for a download) creates issues, after which they propose their scheme to solve it. If a TCP connection is broken, the client responds by a new HTTP range request for the remaining bytes. After receiving all the pieces of the file, the client reconstructs the whole file. The scheme, although simple, raises some important questions like:

1. How does the client learn its a connection disruption and not a network loss?
2. What to do with the dormant TCP state at the old server?
3. How much does TCPs slow start affect the throughput?
4. Server performance
5. How secure is the system?

Then the authors give provide us with a very relevant yet detailed background on the problems. They go deeper into the issues with TCP over anycast networks where they basically re-iterate the two problems that they mentioned earlier. I was however caught on one point where they say: “IP routing does not use application level data, thus the routing decision can lead to server load imbalance” It would have been better had they explained it a bit more in the background section (they reference this to their own previous work, however a small explanation would have been helpful). I pretty much liked the way they describe the potential disruptions which can be classified into internal and external routing changes (w.r.t. the CDN network) and the use of diagrams to explain them. Also, the TCP behavior due to disruptions is very well depicted in figure 2 and helps us understand the process well.

Next the authors seamlessly lead us into a discussion of some ways of solving the connection disruption problem where they effectively bring out the flaws in each solution. However, I wasn't very convinced with the idea of utilizing the mobility support in the IPv6 to intentionally redirect the client to a different server. How would the client or the server know if it is about to be redirected? What if upon redirecting to a particular edge server, a better path to a different edge server is discovered? However, this discussion is not very close to the topic of the paper and thus not addressed in detail. One important point mentioned here is the fact that static binding CDNs are more susceptible to DoS attacks and server performance degradation. I think that this area is particularly interesting and I would like to know how Akamai, which uses static binding, avoids or prevents these attacks.

In the next section, the actual mechanism of anycast aware transport is described. It has been implemented by modifying the TCP at the client as well as the server side. Upon being disrupted, the client issues an HTTP range request. At the server side, the proposal is to use a very small maximum retry count. This helps the server close the TCP connection quickly (upon a disruption) and hence avoid the problem of too many dormant TCP connections.

This might create problems when the connection is disrupted due to network delays. However, the authors mention (and later show in their results) that this does not affect performance much which I found a bit surprising. Even if the networks nowadays are less lossy, there are losses always (we saw a fair amount of losses when we worked on the Planetlab network!) and it should be up on the results. An analysis on this observation could have thrown up some interesting things.

The performance evaluation of the system is done on the basis of three aspects:

1. Impact of connection disruptions:
   This is basically the analysis of the throughput of the system and as mentioned before, the impact of the disruptions is not marked at all. The authors used a dummynet node between a server and client to emulate the network conditions. Though the set up doesn't look like it will give erroneous results, an evaluation on a real network could have given us more reliable results.
2. Impact of closing the dormant TCP connections quickly:
   This experiment was a much more reliable one as it used Keynote clients to establish a real world scenario and again it gave a surprising result by registering only 0.17 percent dropped connections when the modified 1 retry TCP was used. Does this mean that the networks these days are highly reliable?
3. Synchronization of requests (performance of the server):
   The scheme raises concerns over synchronization effects when many requests routing to a server A are simultaneously shifted to server B. Will the performance of the server go down with the sudden bombardment of requests due to synchronization? The methodology was particularly interesting here where the authors monitored the queue size at a bottleneck in the network. A synchronization problem would yield a highly oscillating queue size. However, there was no evidence of the same. I think this is a good estimate of as the evaluation environment was the most conducive to synchronization (real world environment would introduce more randomness).

Overall, I think the authors did quite a good job coming up with a scheme which is quite simple and looks highly effective. The scheme is very elegant in solving some security loopholes (DoS attacks) which exist in the current CDN setup by reducing the dormant TCP connections. The evaluation methodologies give us a good idea of the performance of the scheme, though a more real world analysis of the scheme is necessary due to the presence of various routing variations out there which could alter the behavior of the scheme. The authors even acknowledge the fact by mentioning the effect of Policy based routing. Since the CDN traffic today is mostly large multimedia files (as shown in the paper “An analysis of Internet Content Delivery Systems”), this simple scheme can contribute a lot in reducing the load on edge servers as well as routers. One drawback that might affect the scheme could be its implementation. All edge servers as well as participating clients will have to modify their protocol stacks which is quite a challenging task and could hinder the implementation of the scheme. But of course, most of the network architecture oriented new schemes suffer from the same limitations.