Stateless tcp

1/11

Stateless tcp Doesn't have a concept of tcp states or connections. It Determine what to do Based on an incomming packet. hence if a ACK came in we would look at the ACK's Parametars and Determine what to Do. No socket or connetion State would be kept on the Stateless tcp host.

This method would Be Ideal for small Webservers and a Proper Design could handle an Amazing Load since No Connection State needs to Be maintained

There are limitations however. Since No Connetion State is maintained IF AN Ack From the Peer is dropped along the network path. the Connection would hang until the Peer sent another ACK, this could Be along time or Never.

So how Does it Work?

Basically Every thing is ACK number Pumped when we Receive a SYN we get to pick our Sequence # we send Back to the Peer.

So If we Recv SYN.

Send SYN/Ack with a Seq # of (0-1) Zero-1

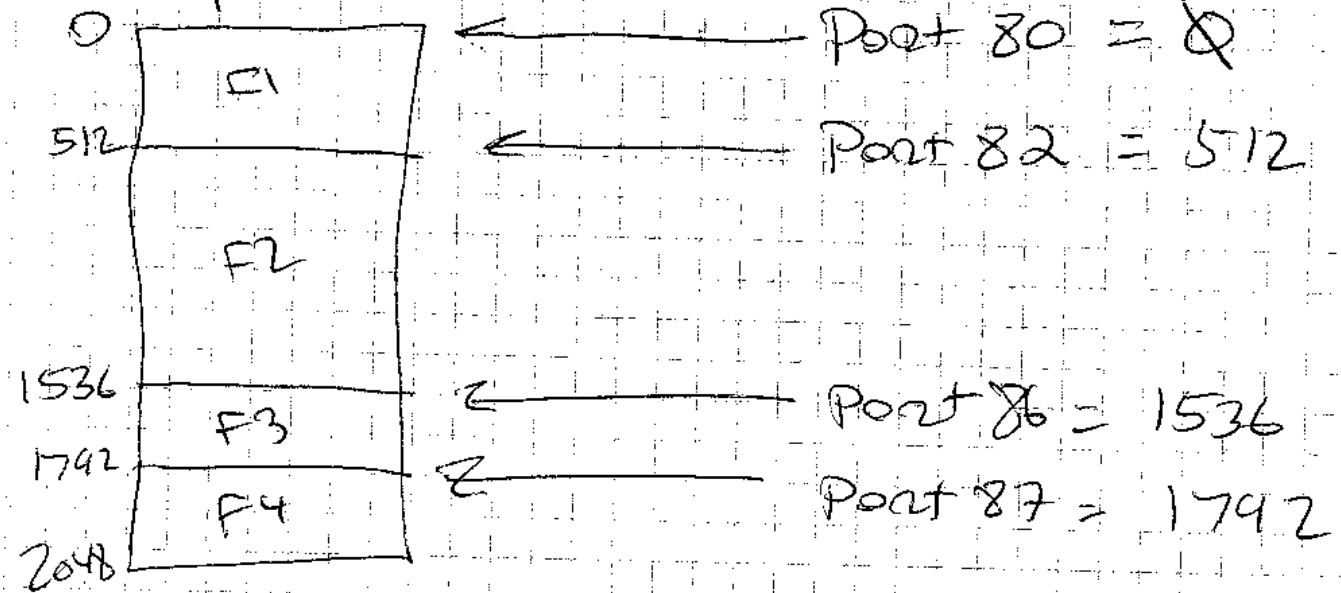
Inventor NQ Date
Read and understood by
Witness RK Date Dec 95

Stateless TCP

Webserver example

Assume 256 Block Size.

Port 80 is File Zero!



Mapping of Ports to files.

Block INDex or File INDex is the PORT #, Sequence # is the INDex into a file.

Fig 1

Stateless TCP

After we have sent our SYN/ACK we should get an ACK back from the Peer. The ACK # should be zero. So since there are no other flags set besides the ACK (and possibly PSH which we ignore.). We send DATA in response.

Now the DATA format is important here. We will assume that we only send 2^N sized chunks. Where N can be any reasonable value. Lets say that we have $2^8 = 256$ or a 256 Block size.

So in the above example we would send a 256 Byte segment from the beginning of our file. So now we would expect an ACK of 256 where we would send the next block of our file to the Requester, this would go on up to the end of the file has been reached where we would send a FIN when we receive a segment that ACKs the END of our file.

our FIN would get ACKed, AND we would finally ACK our FINAL ACK.

If we drop a packet we could be in trouble, AND we have to wait for the Delayed ACK timer on the peer. Before we will get our ACK. So performance on the connection will be low because of the SRP operation (no windows, 1 segment in flight), But we can fix this.

(cont)

Inventor Date

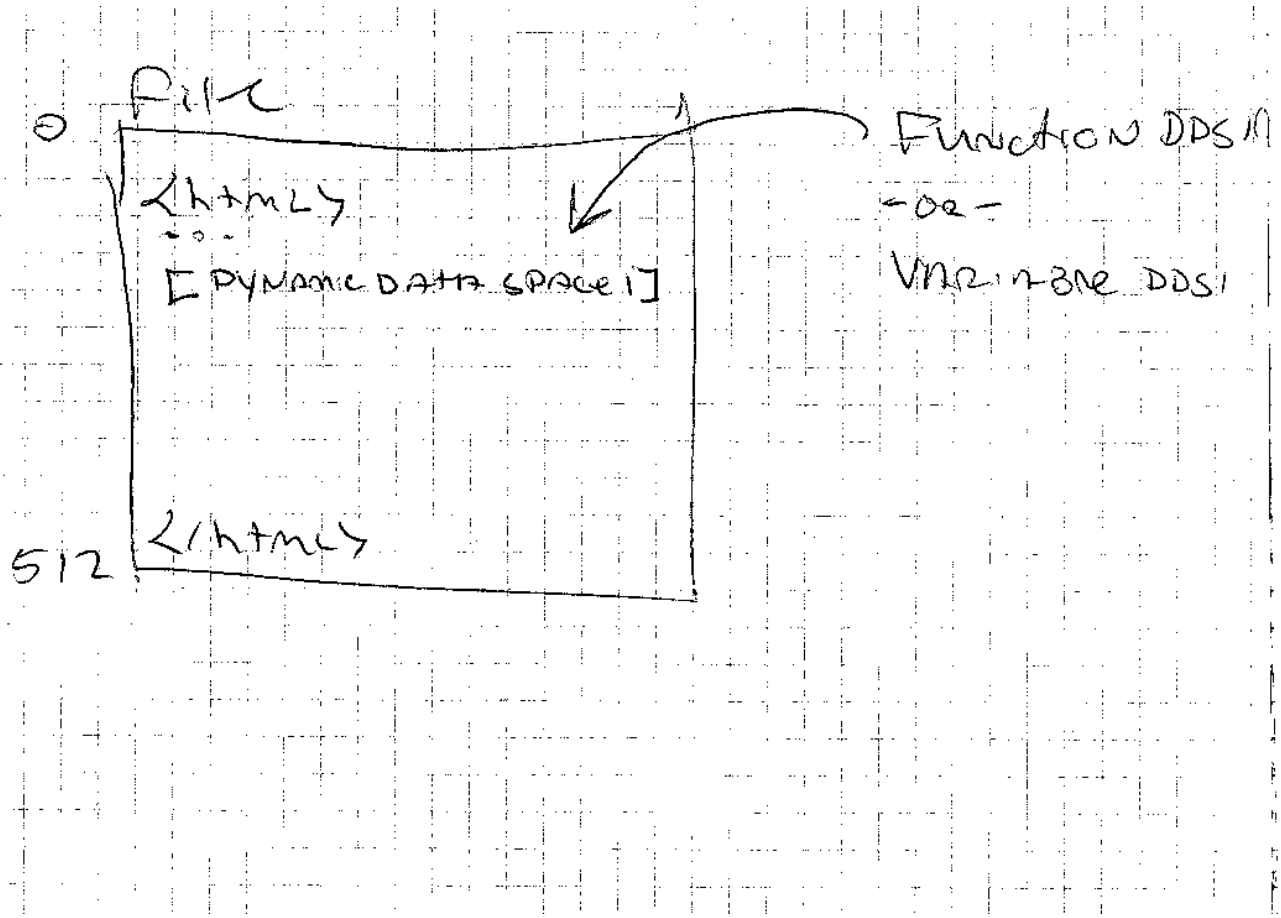
Stateless TCP

If we send not one segment per ACK received but two we reduce the possibility of one dropped packet hanging the connection and we don't have to wait for the peer to do a delayed ack timeout before sending an ACK back to us. (Delayed ack specifies that we should send an ACK every 2 segments received. Not only will sending 2 segments per ACK speed up the throughput it will increase reliability of the connection.

So what about multiple files?

The above makes it seem like you may only send 1 file since you ~~cannot~~ don't keep state you cannot parse a file retrieval command unless a full reply is going to be sent as a response (which is possibly). So looking at figure 1 you see that we add port #s into the mix. The main page would be at port 80 and each port number ~~would~~ above 80 would increment the block index returned by the stateless TCP device. So any place in the file system can be found by a port # + sequence # operation (ie file 2 starts at location 512 which could be retrieved by ACK of 513 port 80 or an ACK of zero @ port 82). This allows us from our main tcp page to specify a page below our main page with:

<A href="oneIP:port">linkLA>


0
File
<html>
...
[DYNAMIC DATA SPACE]
512
</html>
FUNCTION DDS
-OR-
VARIABLE DDS

Stateless tcp

What About Dynamic Data?

Dynamic Data Presents a Problem. Since each incoming segment has no state other than what it brings with it you cannot serve Per connection Dynamic Data. But the Real Question is can you serve Dynamic Data at all? Yes! we can.

in each place where we want to serve Dynamic Data we need a placeholder in the file system exactly the same size as the Dynamic Data. It is possible that the whole file is Dynamic Data. It could be a frame buffer of a camera. or it could be mixed text and Dynamic Data the server must know where the Dynamic Data starts and