TCP in depth

TCP is so effective because it is based on a handshake system where:

1. A Connection needs to be established – System A needs to establish the connection with system B before going further. A sends over a packet introducing itself and B returns an acknowledgement where it validates system A.
2. Byte stream service – Send over a stream of bytes, no matter the packet breakdown. Aka you can have the data stream in bytes, split up into IP packets, tagged nicely by the protocol and assembled upon receipt.
3. When the packet is received the recipient sends over an acknowledgement (ACK), which the sender expects. Otherwise the sender will re-send.

TCP header structure:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Source port | | Destination Port | | | | | | |
| Sequence Number | | | | | | | | |
| Acknowledgement Number | | | | | | | | |
| Data  Offset | Reserved | U  R  G | A  C  K | P  S  H | R  S  T | S  Y  N | F  I  N | Window |
| Checksum | | Urgent Pointer | | | | | | |
| Options | | Padding | | | | | | |
| Data | | | | | | | | |

\*Highlighted elements will be focused on

Source and Destination ports are the identifiers attached to the IP when the transaction happens

The way we identify the connection instance is through a combination of the Source Address, Destination Address, Source Port and Destination Port from both the IP header and TCP header.

Example.

Source Destination

A (Client) 🡪 B (Server)

[SourceIP]12345

[DestinationIP]22 🡪

[SourceIP]54336

[DestinationIP]22 🡪

It is also by-directional, when sending from the Server to the Client we flip the destination and source IP’s and Ports

Destination Source

A (Client) 🡨 B (Server)

🡨 [SourceIP]22

[DestinationIP]12345

🡨 [SourceIP]22

[DestinationIP]54336

When TCP packets are sent around only the destination port would be referenced in the ACL, while the source port is a reference for the receiver for where to send back to.

Single Subnet, single ACL example.

Client IP: 10.0.1.80 Server IP: 10.0.2.76

Request 1

Packet destination port: 22 🡪

Request 2

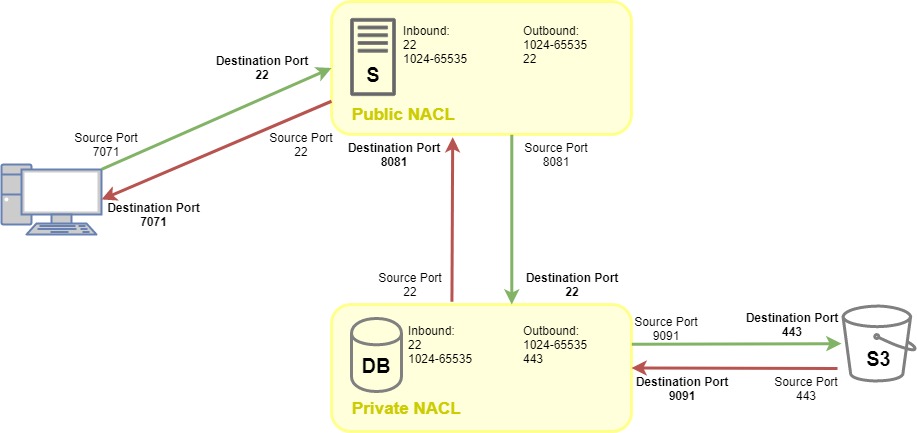
Packet destination port: 22 🡪

Response 1

🡨 Packet destination port: 12345

Response 2

🡨 Packet destination port: 54336



Port ranges and Ephemeral ports

0 – 1023 - System or Well-known ports (SSH, HTTP, …)

1024 – 49151 - User or Registered ports (indicate certain application or service)

49152 – 65535 - Dynamic or Private ports (range of free ports for wide use)

TCP Client (system sending the request) is assigned a random port in the range of 1024 – 65535, so it doesn’t clash with any System port, which would be assigned to the Server (the one fulfilling the requests).

This creates a unique connection which can be easily distinguished between different packet requests.

* Destination port is known to both systems as it is a Well-known port
* Source port is known to the Server as it was sent with the initial connection request (SYN packet)

Example.

Client IP: 10.0.1.80 🡪 Server IP: 10.0.2.76

Request 1

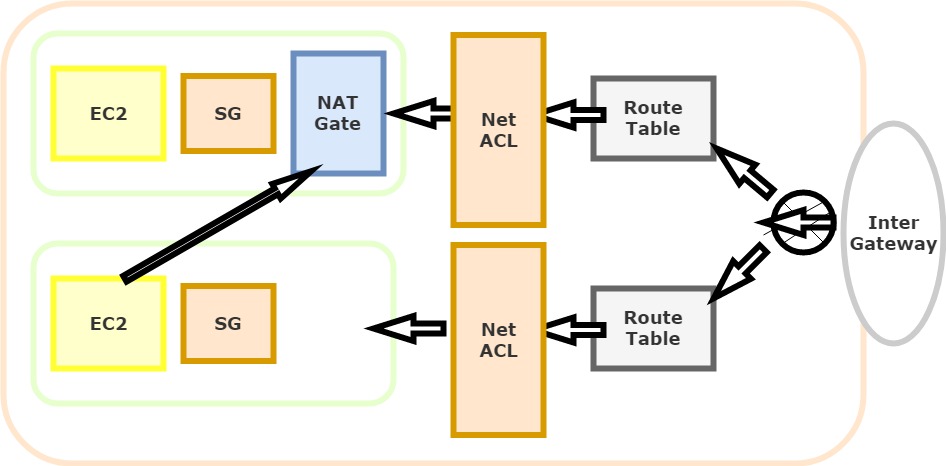
Client port: 3454 🡪 Server port: 22

Request 2

Client port: 9113 🡪 Server port: 22

Ephemeral ports

The range of ports, commonly from 1024 – 65535 (if using a system port to connect to, otherwise be careful in the Registered port range, as clashes may occur) used for a single transaction as an inbound or outbound port allowed. If you are using TCP you need to allow Ephemeral ports range in order to account for the unique Client port assigned in the handshake.



My NACL setup:

Top subnet is the public one

Bottom is the private

I would SSH through the public EC2 instance into the private one

Top NACL:

Inbound Outbound

80 80 Web Server traffic

443 443 Web Server traffic

22 Ephemeral Range SSH outside VPC / Web Server traffic

Ephemeral Range 22 SSH to/from Private (only to select EC2)

Bottom NACL:

Inbound Outbound

22 Ephemeral Range SSH return to Public (only to select EC2)

Issues and Questions:

Had an experimentation moment where I allowed a smaller range of Ephemeral ports, which blocked the SSH process a few times until the auto-allocated port was in the range allowed through the NACL.

How many request caps can we have from one IP ?

Also when I set up a Gateway Endpoint I probably have to allow something out of the ACL as it would not route through. AKA I needed to specify S3 ports to route through outbound and Ephemeral for the inbound? What is the S3 CIDR block

<https://aws.amazon.com/premiumsupport/knowledge-center/connect-s3-vpc-endpoint/>

How does my Net ACL look? Is this a good way or is there a way to improve?