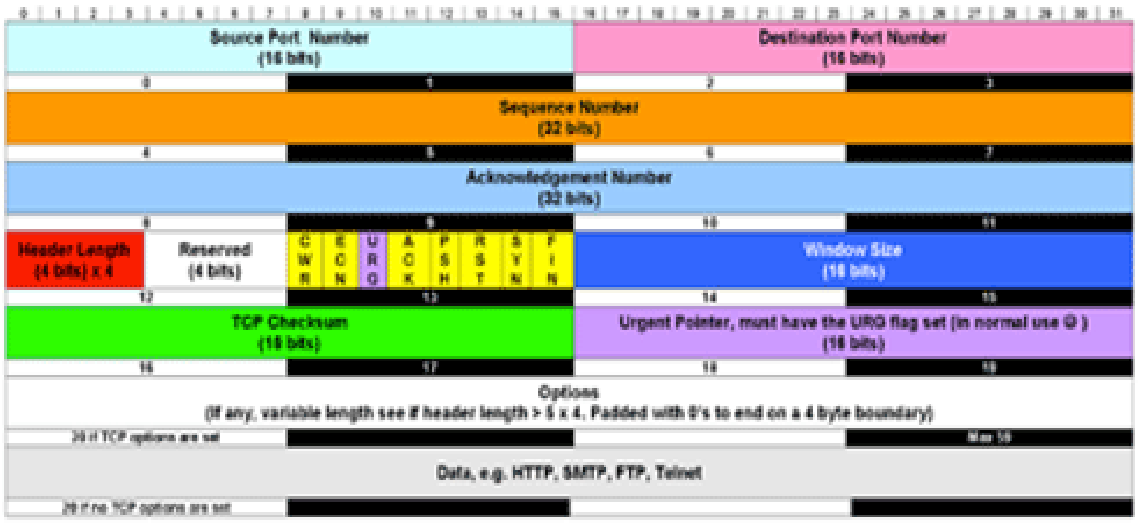
How to Remember Your TCP Flags



Source Port Number (16 bits)
Destination Port Number (16 bits)
Sequence Number (32 bits)
Acknowledgement Number (32 bits)
Header Length (4 bits) x 4
Reserved (4 bits)
CWR
ECN
URG
ACK
PSH
RST
SYN
FIN
Window Size (16 bits)
TCP Checksum (16 bits)
Urgent Pointer, must have the URG flag set (in normal use 0)
(16 bits)
Options
(If any, variable length see if header length > 5 x 4, Padded with 0's to end on a 4 byte boundary)
20 if TCP options are set
Max 59
Data, e.g. HTTP, SMTP, FTP, Telnet
20 if no TCP options are set

Many people are familiar with the concept of a mnemonic [nəˈmɑnɪk] — a memory device that uses a phrase based on the first letter of words in a series. Perhaps the most popular of these in the field of networking is the one for the OSI Model (All People Seem To Need Data Processing).

Well, for those that deal with TCP/IP a lot, I thought it might be helpful to have a mnemonic for the TCP flags as well. What I've come up with and use regularly is:

__U__nskilled __A__ttackers __P__ester __R__eal __S__ecurity __F__olks

Unskilled = URG
Attackers = ACK
Pester = PSH
Real = RST

Security = SYN
Folks = FIN

USES

TCP flag information is most helpful to me when looking for particular types of traffic using Tcpdump. It's possible, for example, to capture only SYNs (new connection requests), only RSTs (immediate session teardowns), or *any* combination of the six flags really. As noted in my own little Tcpdump primer, you can capture these various flags like so:

Find all SYN packets

```
tcpdump 'tcp[13] & 2 != 0'
```

Find all RST packets

```
tcpdump 'tcp[13] & 4 != 0'
```

Find all ACK packets

```
tcpdump 'tcp[13] & 16 != 0'
```

Notice the SYN example has the number 2 in it, the RST the number 4, and the ACK the number 16. These numbers correspond to where the TCP flags fall on the binary scale. So when you write out:

U A P R S F

…that corresponds to:

32 16 8 4 2 1

EXAMPLE

So as you read the SYN capture `tcpdump 'tcp[13] & 2 != 0'`, you're saying find the **13th** byte in the TCP header, and only grab packets where the flag in the **2**nd bit is not zero. Well if you go from right to left in the UAPRSF string, you see that the spot where 2 falls is where the S is, which is the SYN placeholder, and that's why you're capturing only SYN packets when you apply that filter.

```
# tcpdump 'tcp[13] & 2 != 0'
```

```
12:40:04.649404 IP 10.5.1.42.51584 >
64.233.187.99.http: S1524039069:1524039069(0)
win 65535
```

```
12:40:04.708459 IP 64.233.187.99.http >
10.5.1.42.51584: S 1416742397:1416742397(0)
ack 1524039070 win 8190
```

You'll notice that when I `netcat`'d to Google on port 80 from another terminal, `tcpdump` shows only two out of the three steps involved in the three-way handshake. It didn't show the third because the final step is simply an ACK from my side, i.e. no SYN flag set.

CONCLUSION

Remembering these flags and how to make use of them can go a long way in helping low-level network troubleshooting/security work by isolating what it is you want to see and/or capture. And of course the better you can isolate the problem, the faster you can solve it.: