Transport Attacks

I.G.Batten@bham.ac.uk

Attacks on UDP and TCP

- UDP: datagrams, no “connection” state
- TCP: reliable, sequenced data
- Both routinely passed through firewalls
- Both routinely attacked

Filtering

- Simple filtering looks at the IP address and the port number in both the source and the destination.
- So if you only want to receive traffic to your DNS server from a particular network, you write something like:
  - **pass** traffic to port 53 from network 147.188.0.0/16
  - **block** all traffic to port 53 from elsewhere.
- Another example is “accept email only from my filtering service” or “accept remote logins only from my branch office”.

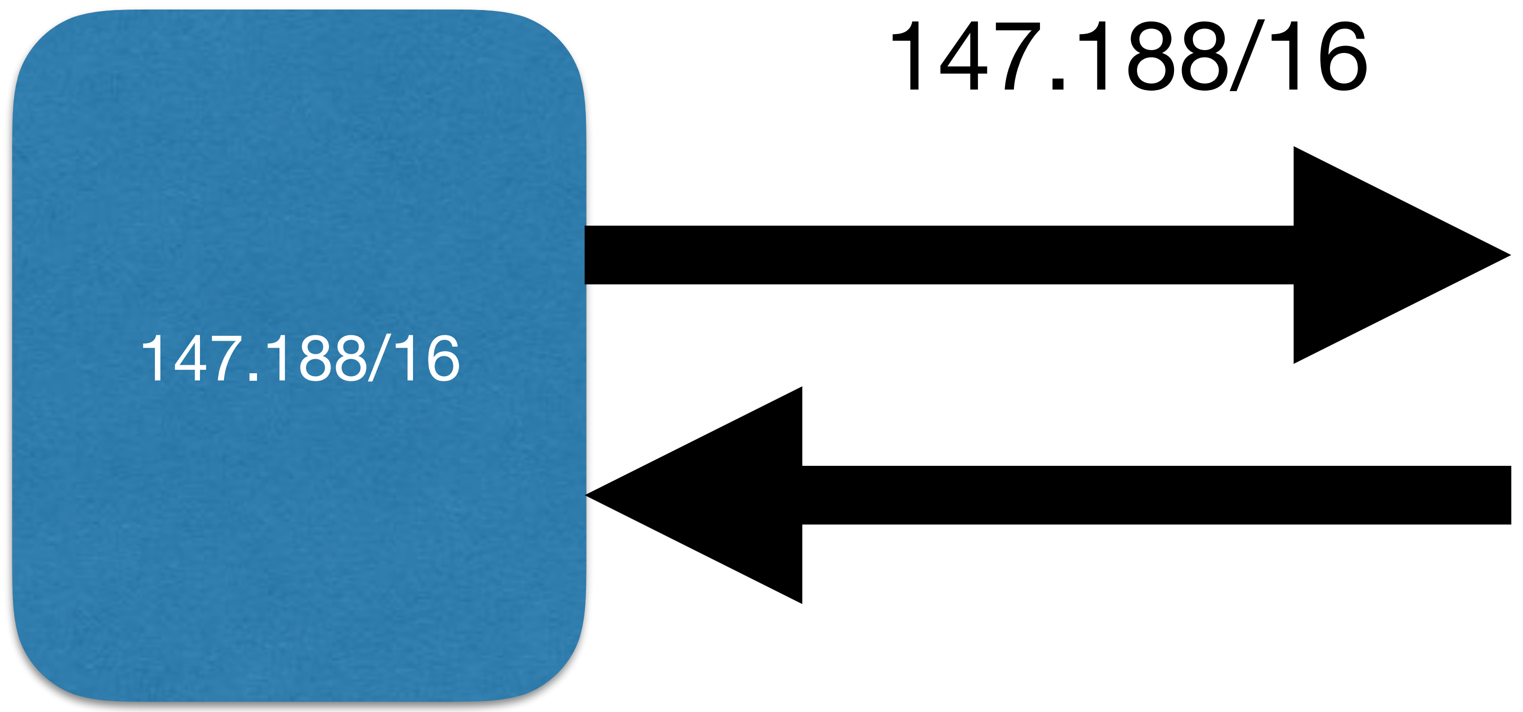
Why doesn’t this always work as well as you want?

- The attacker is assumed to have full control over the entire contents of the packet.
    - This is a good assumption because it’s true. Anything which assumes the attacker cannot set any byte of the packet to any chosen value is unlikely to be correct.
- The attacker can make traffic appear to come from anywhere, including your “whitelisted” network.

Seeing the responses

- It is trivial to forge packets coming from a particular address. Indeed, it’s routinely done by accident.
- There are proposals to make it harder (BCP38) and RFCs discussing it (most recently RFC 6959) but most ISPs do not do effective egress control (“don’t let packets out with addresses I don’t control”) or effective ingress control (“don’t permit inbound packets with source addresses that are already inside my network”).
- It is in general harder to get hold of packets sent to an address you don’t control (requires alteration of routing tables, or strong interception capability). Different class of attacker.
- So interesting to see what can be done by an attacker who can send you arbitrary packets, but cannot see traffic not destined for them.

What should be blocked, but isn't?



Discard all traffic outbound **not from** 147.188/16

Discard all traffic inbound **from** 147.188/16

This is all worth doing yourself, by the way

What does this matter

- Attacker can bombard your UDP services with arbitrary traffic (but cannot see the responses)
- Attacker can bombard your TCP services with arbitrary traffic (but cannot see the responses)
- And other stuff, like ICMP, but that's much more likely to be filtered these days.

UDP Attacks

- Can send packets in the hope of (for example) buffer overruns
- Can send packets in hope of DoS
- Can send packets to use services that are authenticated by source
- And can send packets as part of amplification attacks.

Source Authentication

- One offender is syslog, UDP port 514.
- Send a packet to port 514, contents are appended to a log
- Sounds harmless, but swamps logging service so legitimate messages are dropped or ignored.
- Non-local syslog (eg, passing port 514 through a firewall from a remote site) is madness.

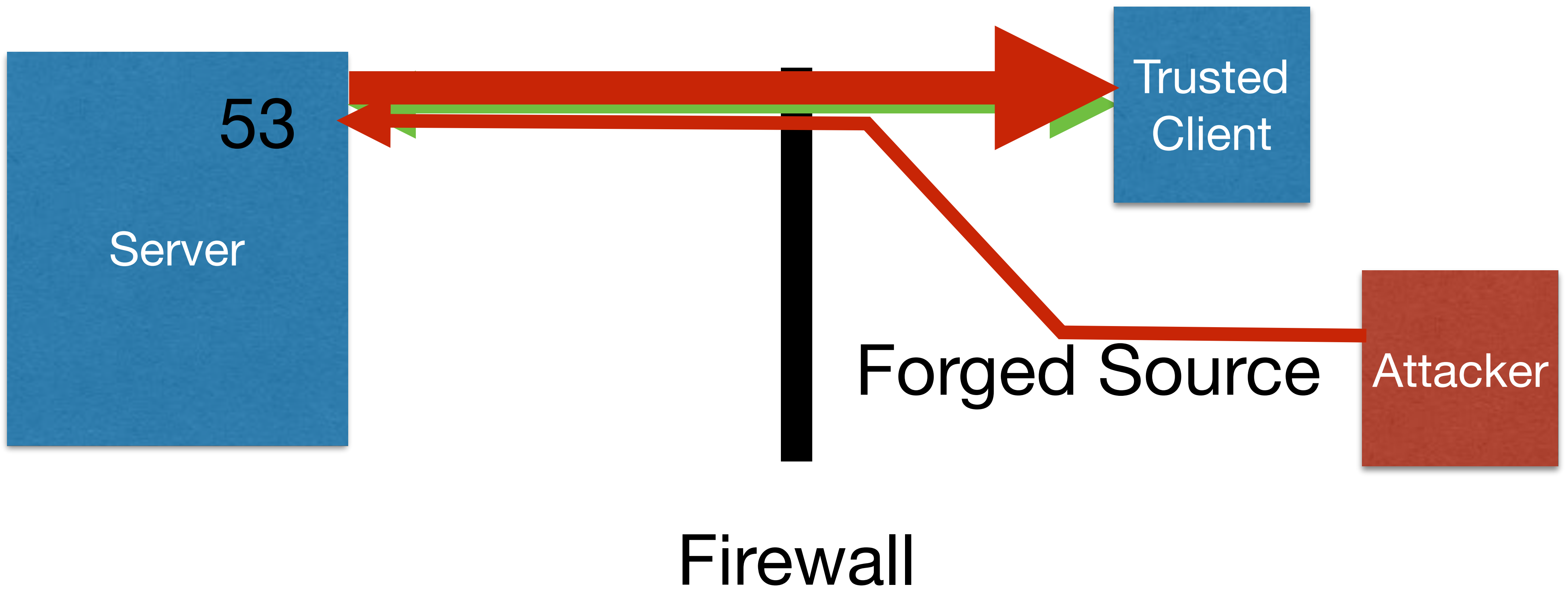
Source Authentication

- Other serious offender (and it gets worse, see next page) are DNS, SNMP and NTP. In both cases, people restrict access to servers by source address because of concerns about information leakage, load on vital infrastructure and “pay for your own damned servers”.   But…

Amplification Attacks

- Nasty even without the amplification.
- Send a packet to an innocent intermediary, using a public service, with the source address spoofed to be a victim’s address.
- Each packet elicits one or more response directed to the victim, coming from the intermediary.
- Some protocols (worst case was an NTP management protocol at around 1:10) amplify hugely.
- Original attacker cannot be traced without help of intermediary.

Amplification Attacks



Broadcast Attacks

- Wide range of attacks (“Smurf”, for example) available if you can send “directed broadcast” packets: packets which when they arrive on the final router are then broadcast, rather than unicast.
  - Send ICMP echo request packet to 147.188.0.0 with forged source address of 1.2.3.4, and in principle 1.2.3.4 will receive an ICMP echo response from every machine on campus.
- In practice, even the most dilatory firewall administrator blocks broadcast traffic.

Splicing Attacks, etc

- TCP sequence numbers distinguish segments
- Need to be unique over long-ish periods of time to avoid reuse within segment lifetime (ie, if all connections started at 0, a flurry of short-lived connections between same ports on same machines would be problematic)
- TCP checksum is per-segment, so will not detect segments being swapped

What if I can predict segment numbers?

- I can send a stream of packets which acknowledge data I can't see
- This sounds useless, but in fact is very, very useful

TCP handshake

Client Address+Port	Client Address+Port
SYN	**X**

Anyone can send this

Server will send this to Client

Server Address+Port	Client Address+Port
SYN	**Y**
ACK	**X+1**

Client Address+Port	Server Address+Port
ACK	**Y+1**

Requires knowledge of Y

Subsequent traffic requires knowledge of X and Y

Given X and Y, all sequence numbers can be predicted

```
i.g.batten@bham.ac.uk... Connecting to smart1.bham.ac.uk. via relay...
220 smart1.bham.ac.uk ESMTP Exim 4.84 Tue, 16 Jan 2018 10:24:26 +0000
```

71 bytes

```
>>> EHLO mail.batten.eu.org
250-smart1.bham.ac.uk Hello mail.batten.eu.org [147.188.192.250]
250-SIZE 104857600
250-8BITMIME
250-PIPELINING
250 HELP
```

121 bytes

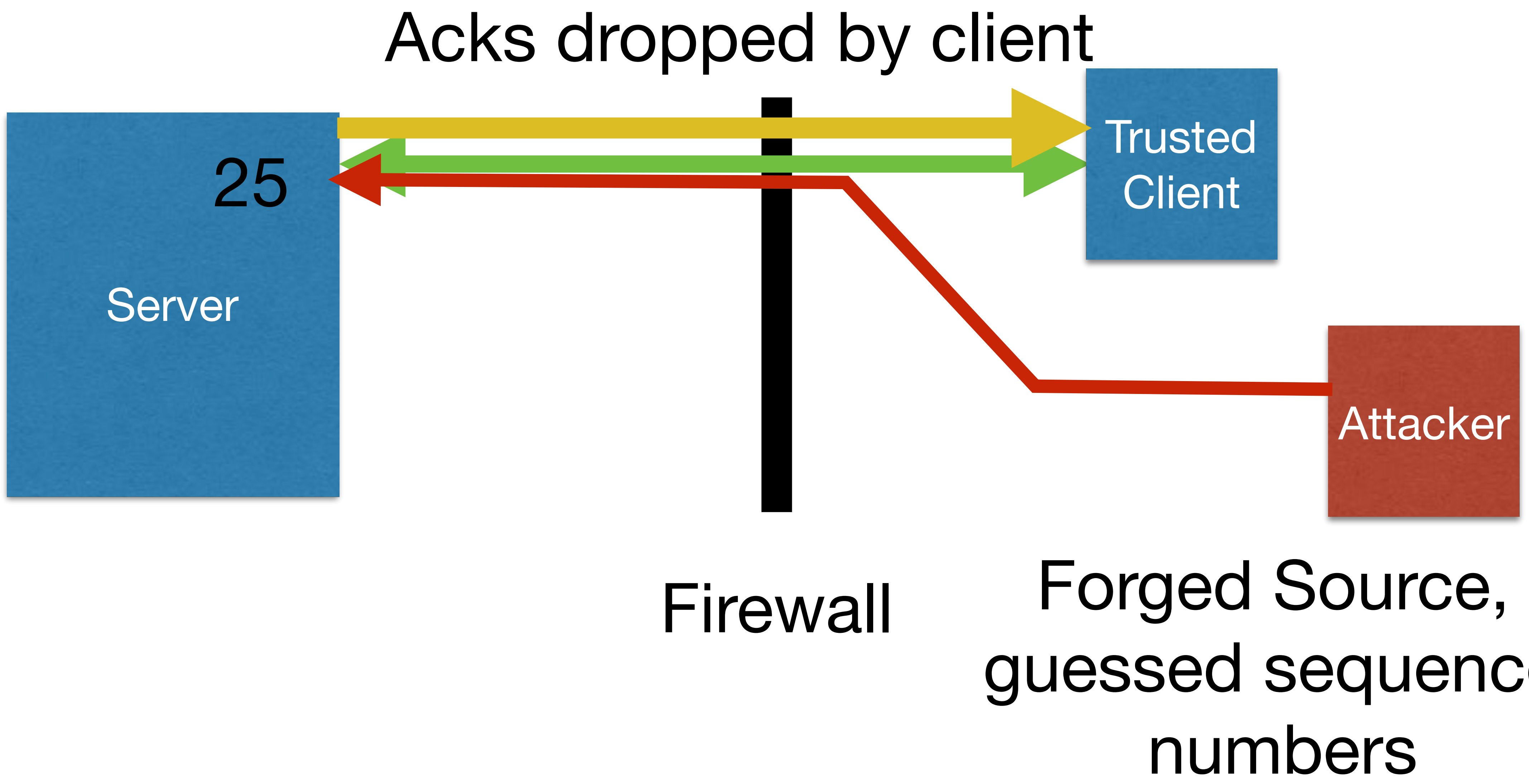
```
>>> MAIL From:<igb@batten.eu.org> SIZE=26
250 OK
```

8 bytes

```
>>> RCPT To:<i.g.batten@bham.ac.uk>
>>> DATA
250 Accepted
354 Enter message, ending with "." on a line by itself
>>> .
250 OK id=1ebOPq-0007zK-EJ
i.g.batten@bham.ac.uk... Sent (OK id=1ebOPq-0007zK-EJ)
Closing connection to smart1.bham.ac.uk.
```

Even if I cannot see the bold text (Server➔Client) I can often predict the length, and therefore send appropriate acknowledgements. If uncertain, “correct” acknowledgements will work, “incorrect” ones ignored.

So


Acks dropped by client
25
Server
Trusted Client
Attacker
Firewall
Forged Source, guessed sequence numbers

What do I send?

- All packets forged Client➔Real Server
  - SYN X
  - ACK Y+1 (having guessed Y)
  - `EHLO mail.batten.eu.org\r\n`, ACK Y+72 (71 bytes)
  - `MAIL From:<igb@batten.eu.org> SIZE=26\r\n`, ACK Y+193 (121 bytes)
  - `RCPT To:<i.g.batten@bham.ac.uk>\r\n`, ACK Y+201 (8 bytes)

Relies on guessing ISN

- Turns out to be scarily easy
- Particularly when a machine is providing a public server you can connect to (http, DNS) and a private server available only to select clients (mail, telnet).
- Predicting ISN now given ISN a known time ago originally very easy (increments 128 per second and 64 per connection)
- Has got better, but not as better as it should…

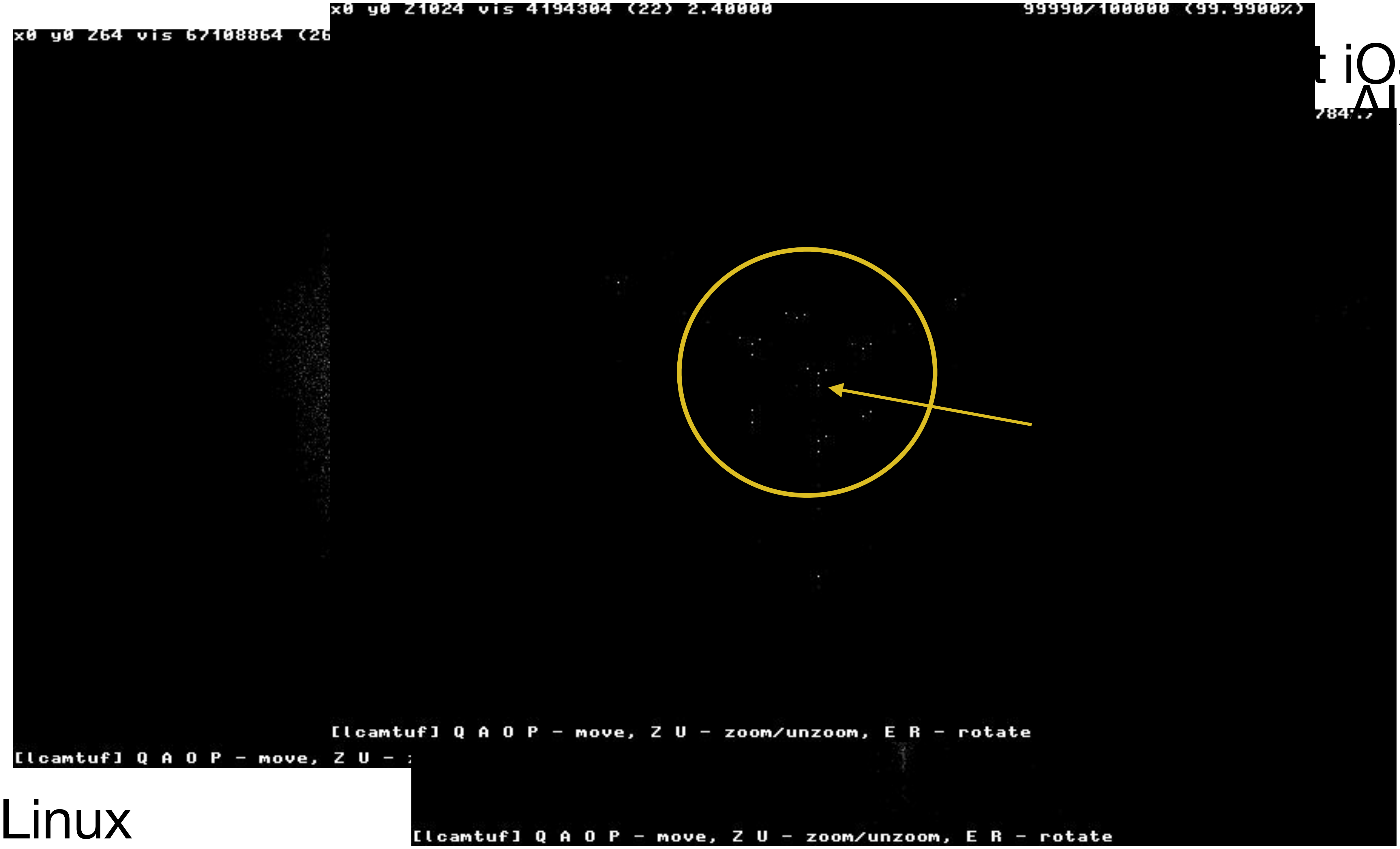
Phase Space Diagrams

- http://lcamtuf.coredump.cx/oldtcp/tcpseq/print.html
- Generates points in three-space based on window of three values (which should yield a uniform cloud):

```
 x[n] = s[n-2] - s[n-3]
y[n] = s[n-1] - s[n-2]
z[n] = s[n] - s [n-1]
```

But real world not very random

t iOS)
AIX


```
x0 y0 Z64 vis 67108864 (26
```
```
x0 y0 Z1024 vis 4194304 (22) 2.40000                99990/100000 (99.9900%)
```
```
784%)
```
```
[lcamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate
```
```
[lcamtuf] Q A O P - move, Z U -
```
```
[lcamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate
```

Linux

Remember…

- I don’t need to succeed every time, or anything like it.
- Broken attempts to connect probably won’t appear in any operating-system logs
- The example I have shown are old, but there are lots of old systems in real networks.
- pf firewall (and commercial products like Cisco ASA) will now replace sequence numbers with good randoms, on a network-wide basis. How? Well, one way follows…

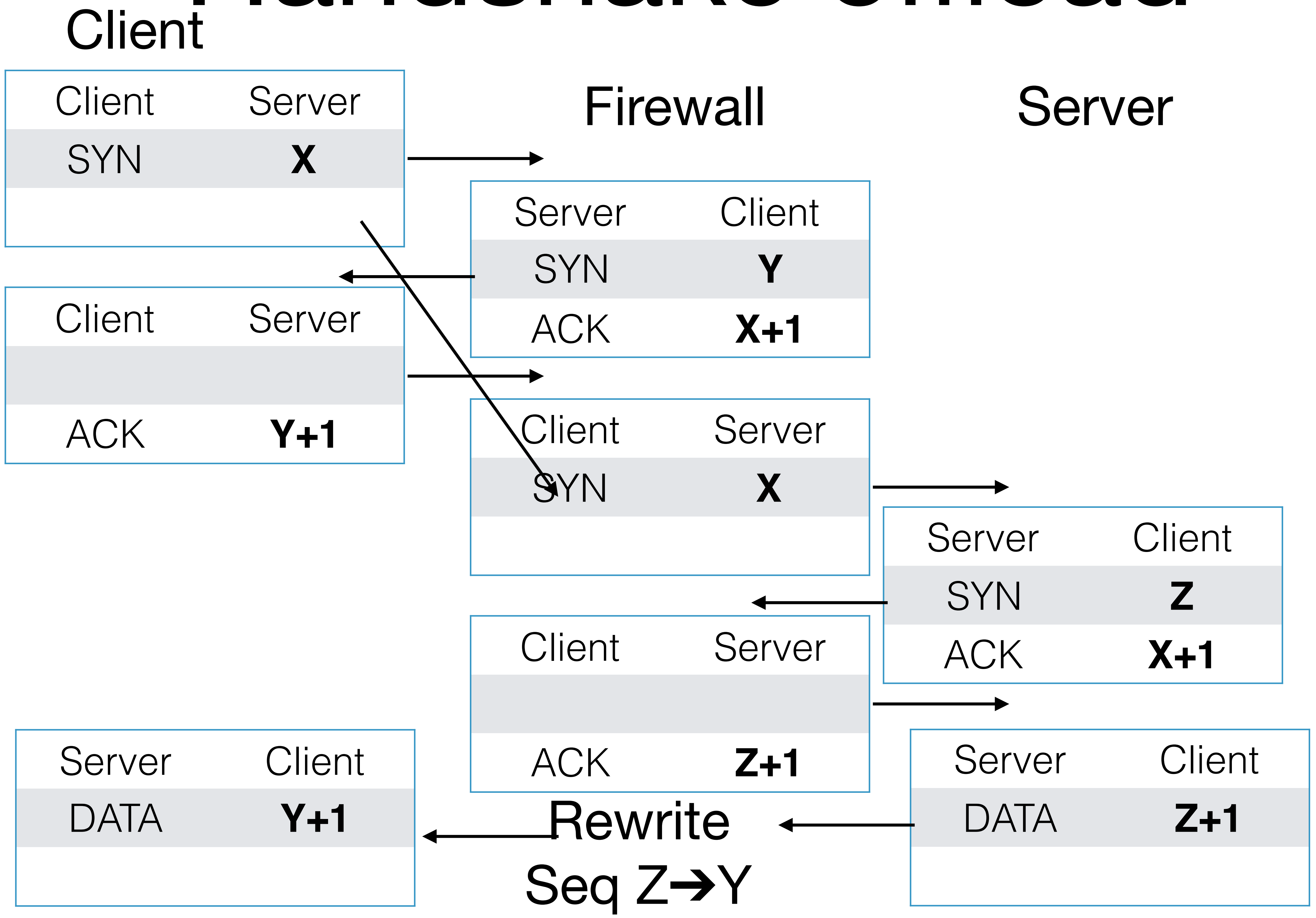
Handshake offload

- Attacker can “SYN Flood” servers by just sending lots of SYNs, and then not responding further.
- For public services, attacker can insert a random source address into each request: hard to filter, hard to track.
- Modern OS should survive, but it’s surprisingly unpleasant: most mitigations slow legitimate traffic as well. Less modern OSes, who knows?
- Solution: do the three-way handshake in the firewall.

Handshake offload

- How can a firewall deal with the three-way handshake?
- Client sends SYN
- **Firewall** sends SYN/ACK, with Firewall-chosen ISN
- **Firewall** waits for ACK from Client
- **Firewall** now passes initial SYN to server, waits for SYN/ACK with server-chosen ISN.
- **Firewall** calculates offset between ISN it chose and ISN the server chose, and applies that to all subsequent packets on the connection.

Handshake offload



Why do this?

- Modern OSes should be able to cope with a “SYN Flood”
- But there are lots of legitimate (ish) reasons why you might be running old software which can’t.
  - Being held to ancient OS ransom by application vendors and your own developers is a thing.
- Border protection is wise.

Handshake offload also…

- Rewrites the sequence number to guard against it being guessed, because the firewall can use a strong RNG.
- BOGOF: Two protections for the price of one.
-

Other TCP Entertainment

- Send SYN (possibly using forged source) and then do nothing: consumes resources.
- Send RST (possibly using forged sources) to break connections
- The list is endless
- In short, punching small holes in firewalls to permit selected TCP connections through doesn’t work in 2018.

Tasks for a firewall

- Drop packets from “outside” which have “inside” IP numbers
- Drop packets from “outside” which have “odd” IP numbers (RFC1918, loopback)
- Drop packets from “inside” whose source is not an inside network
- Rate limit stuff susceptible to abuse
- If possible, do SYN/SYN-ACK/ACK in the firewall
- If possible, re-randomise ISN
- If possible, deal with fragments, dropped packets, etc.
- Expose the services you need