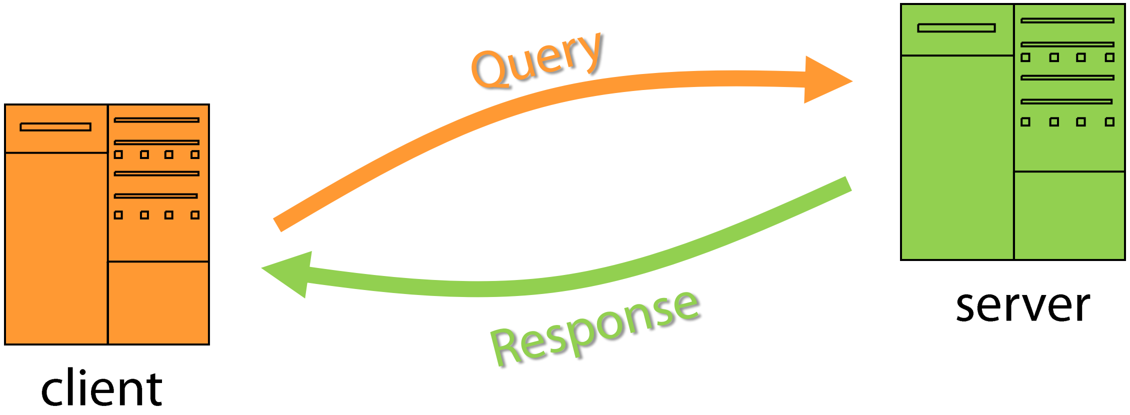
Technion Summer School in Computer Security, September 2016

Attacking the Network Time Protocol (NTP)



BOSTON UNIVERSITY

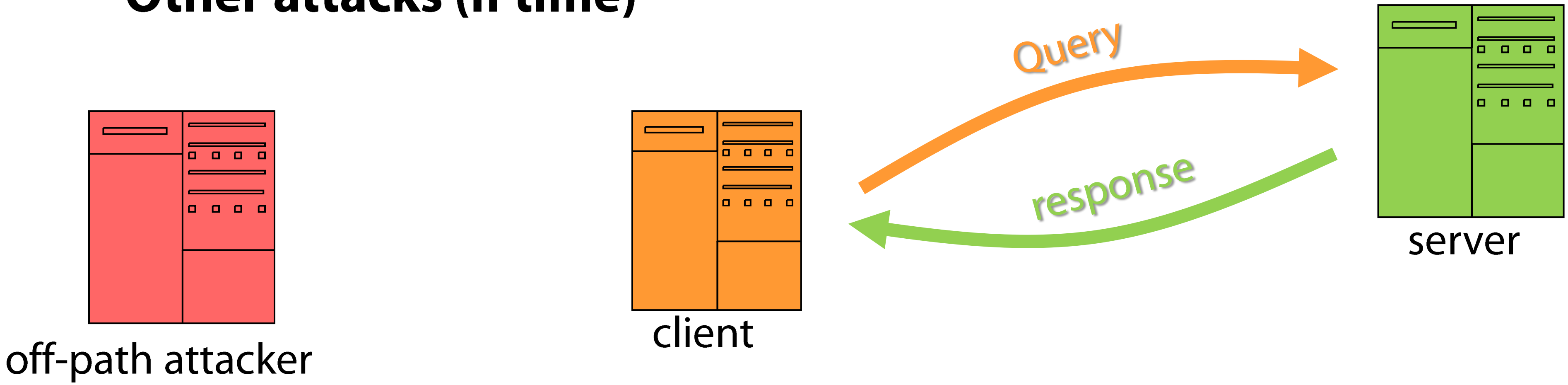
Aanchal Malhotra
Isaac E. Cohen, Erik Brakke
Sharon Goldberg

outline of the talk

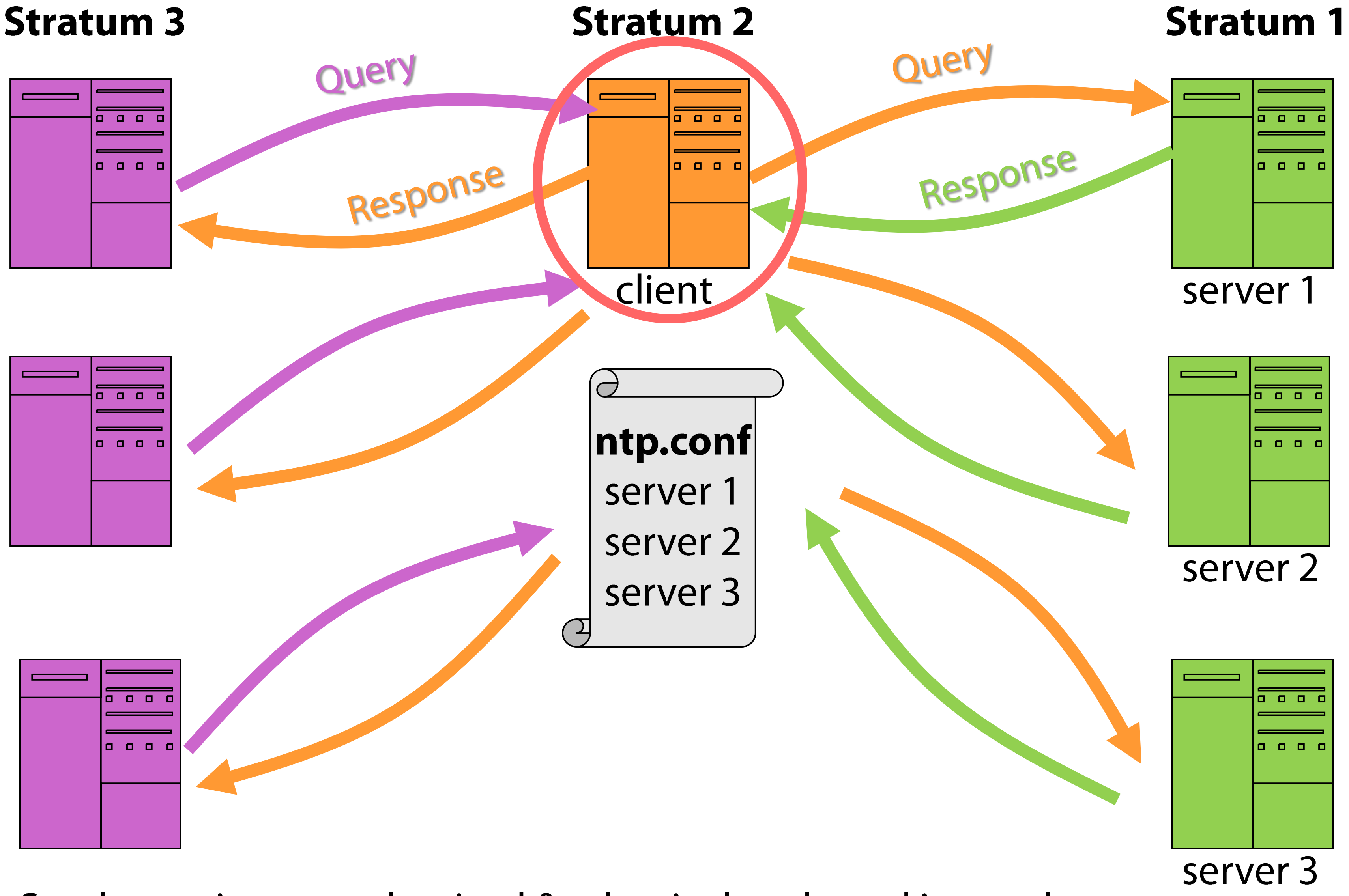
- **Background**
  - How does NTP work?
  - How does NTP client take time?

- **Our client/server mode attacks**
  - Denial of Service by Spoofed Kiss-of-Death (off-path)
  - Denial of Service by Priming the Pump (off-path)
  - Timeshifting by IPv4 Packet Fragmentation (off-path)

- **Broadcast mode attacks**
- **Other attacks (if time)**

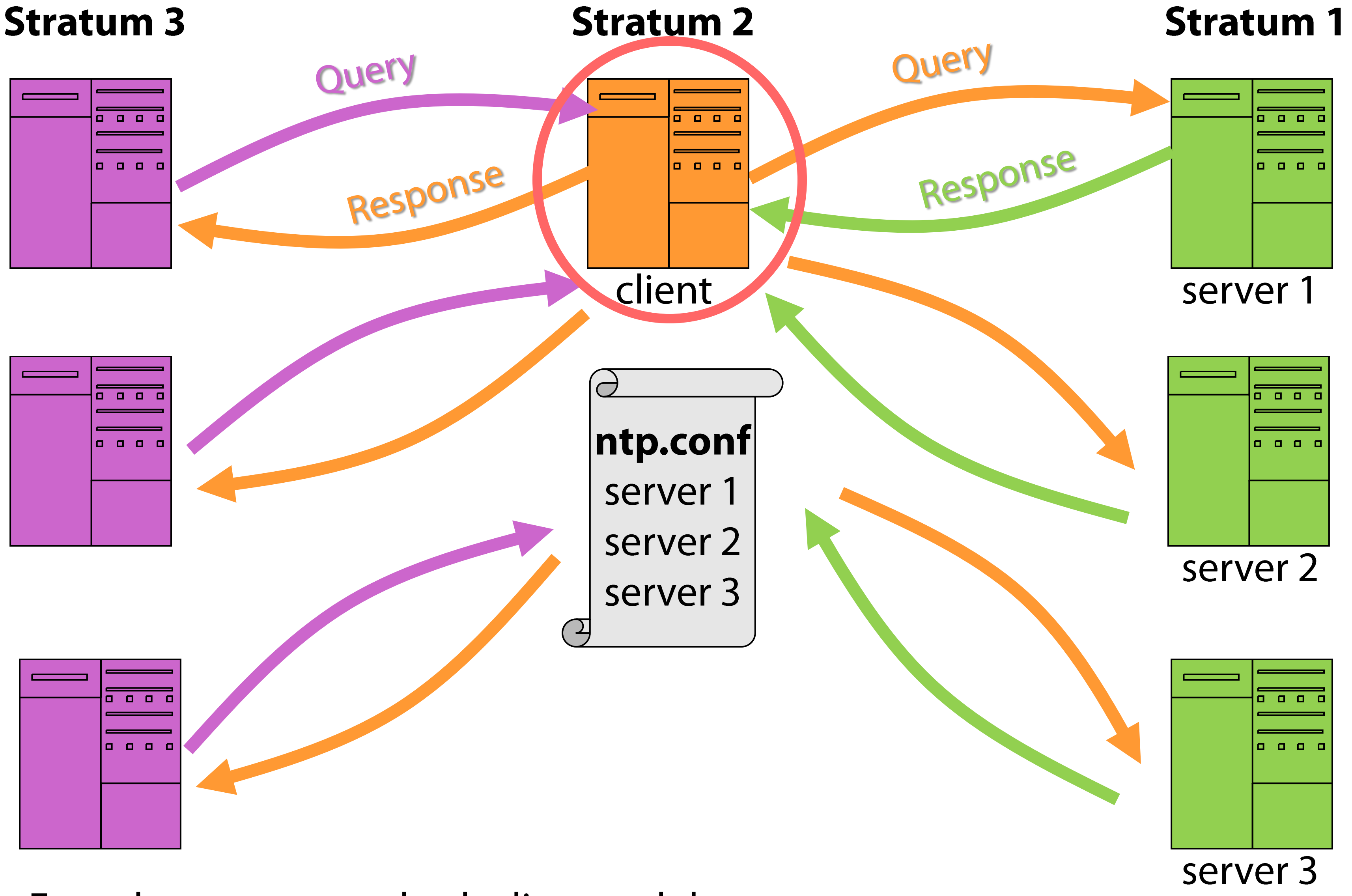


background: how does NTP work?



- Sends queries at randomized & adaptively-selected intervals
- Requires certain number of self-consistent responses to update its clock

background: how does NTP work?



- Every host can act as both client and the server
- My laptop will answer queries from public Internet

the state of crypto in NTP

NTP’s crypto is rarely used in practice

- Symmetric crypto
  - Uses *MD5(key||message*) [RFC 5905] (insecure!)
  - No in-band mechanism for key distribution
- Asymmetric crypto
  - Autokey Protocol [RFC 5906] is not a standards-track doc
  - Crypto is badly broken [S. Röttger’ 2012]

Our zmap scan (July 2016) found 3.9M IPs revealing NTP crypto state

- Only 78K systems have all associations authenticated (2%)

IETF: Lots of activity lately in IETF to develop a secure NTP

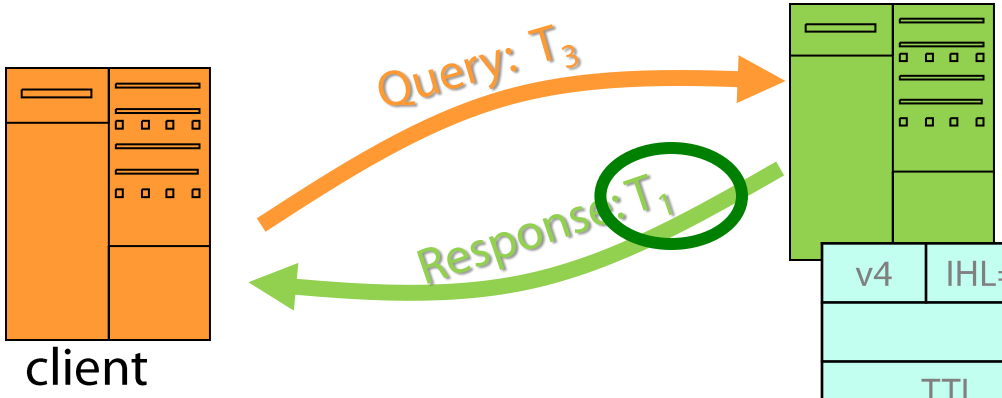
- NTS (Network Time Security)
- Very fluid right now, but potentially based on DTLS.

We attack the NTPv4 spec [RFC 5905]

and its reference implementation
(ntpd v4.2.8p2 & ntpd v4.2.6p5)

We assume NTP messages are
not cryptographically authenticated.

non-crypto authentication with origin timestamp (T_1)



Analogous to
- UDP source port randomization
- TCP sequence no randomization

<table>
<tr><td colspan="2">v4</td><td>IHL=20</td><td>TOS</td><td colspan="5">Total length = 76</td></tr>
<tr><td colspan="4">IPID</td><td>x</td><td>DF</td><td>MF</td><td colspan="2">Frag Offset</td></tr>
<tr><td colspan="3">TTL</td><td>Protocol = 17</td><td colspan="5">IP Header Checksum</td></tr>
<tr><td colspan="9">Source IP</td></tr>
<tr><td colspan="9">Destination IP</td></tr>
<tr><td colspan="4"><b>Source Port = 123</b></td><td colspan="5">Destination Port = 123</td></tr>
<tr><td colspan="4">Length = 76</td><td colspan="5">UDP Checksum</td></tr>
<tr><td>LI</td><td>v4</td><td>Response</td><td>Stratum</td><td colspan="4">Poll</td><td><b>Precision</b></td></tr>
<tr><td colspan="9">Root Delay</td></tr>
<tr><td colspan="9">Root Dispersion</td></tr>
<tr><td colspan="9">Reference ID</td></tr>
<tr><td colspan="9">Reference Timestamp</td></tr>
<tr><td colspan="9"><b>T_1 = Origin Timestamp</b></td></tr>
<tr><td colspan="9">T_2 = Receive Timestamp</td></tr>
<tr><td colspan="9">T_3 = Transmit Timestamp</td></tr>
</table>

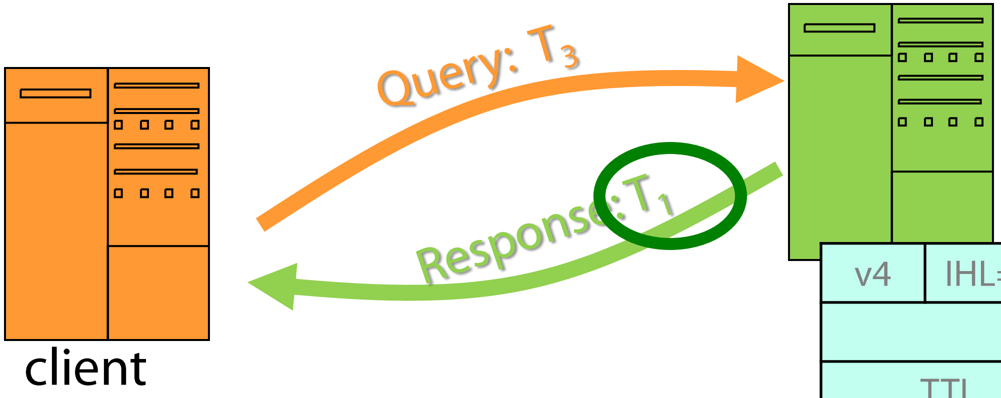
TEST2: Match
T3 in Query to **T1 in Response**.

seconds
fractional seconds

64 bits

*ntpd does not randomize source port!

non-crypto authentication with origin timestamp (T_1)



Analogous to
- UDP source port randomization
- TCP sequence no randomization

TEST2: Match
T3 in Query to **T1 in Response**.

<table>
<tr><td colspan="2">v4</td><td>IHL=20</td><td colspan="2">TOS</td><td colspan="5">Total length = 76</td></tr>
<tr><td colspan="5">IPID</td><td>x</td><td>DF</td><td>MF</td><td colspan="2">Frag Offset</td></tr>
<tr><td colspan="3">TTL</td><td colspan="2">Protocol = 17</td><td colspan="5">IP Header Checksum</td></tr>
<tr><td colspan="10">Source IP</td></tr>
<tr><td colspan="10">Destination IP</td></tr>
<tr><td colspan="5"><b>Source Port = 123</b></td><td colspan="5">Destination Port = 123</td></tr>
<tr><td colspan="5">Length = 76</td><td colspan="5">UDP Checksum</td></tr>
<tr><td>LI</td><td>v4</td><td colspan="2">Response</td><td>Stratum</td><td colspan="4">Poll</td><td><b>Precision</b></td></tr>
<tr><td colspan="10">Root Delay</td></tr>
<tr><td colspan="10">Root Dispersion</td></tr>
<tr><td colspan="10">Reference ID</td></tr>
<tr><td colspan="10">Reference Timestamp</td></tr>
<tr><td colspan="10"><b>T_1 = Origin Timestamp</b></td></tr>
<tr><td colspan="10">T_2 = Receive Timestamp</td></tr>
<tr><td colspan="10">T_3 = Transmit Timestamp</td></tr>
</table>

<table>
<tr><td colspan="2"><b>seconds</b></td></tr>
<tr><td><b>precision</b></td><td><b>randomness</b></td></tr>
</table>

64 bits

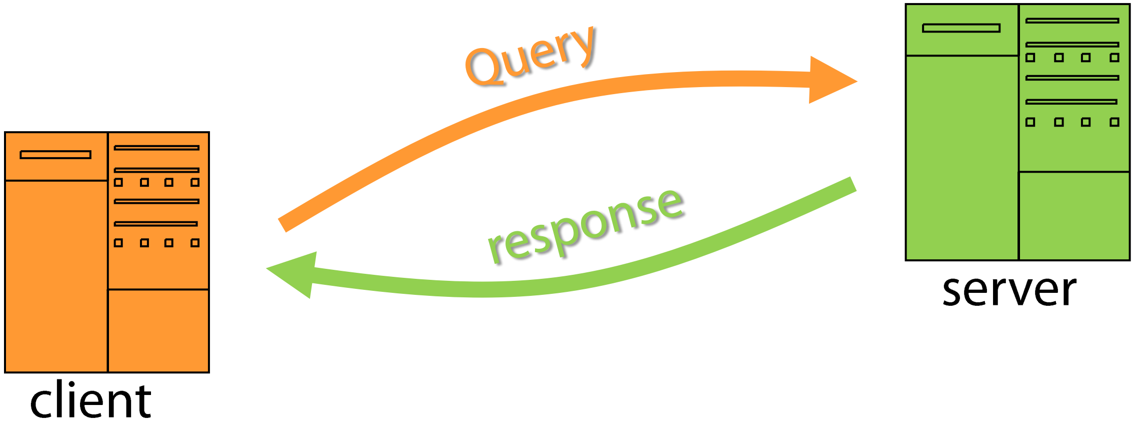
*ntpd does not randomize source port!

outline of the talk

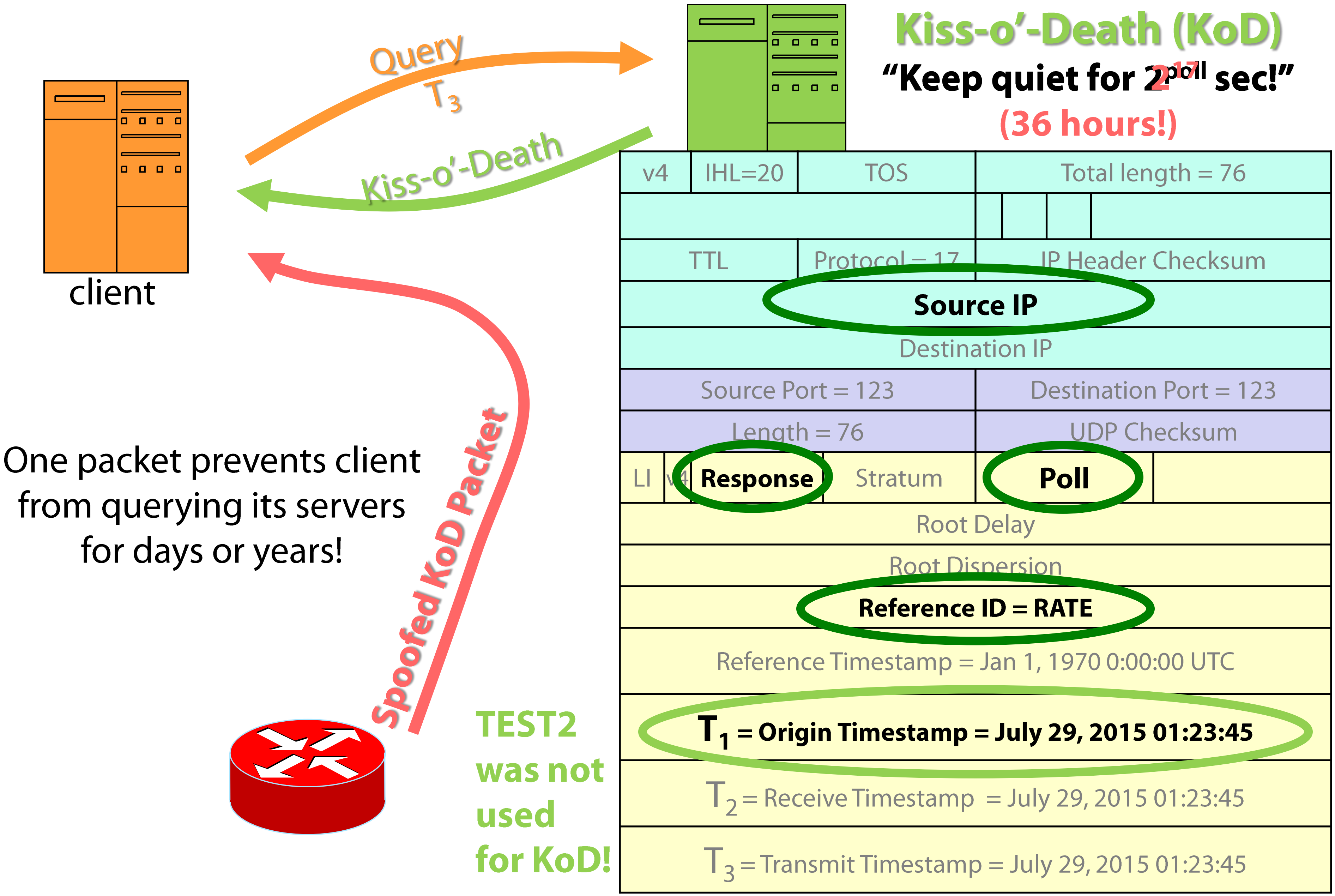
- Background
  - ➢ How does NTP work?
  - ➢ How does NTP client take time?

- Our client/server mode attacks
  - ➢ **Denial of Service** by Spoofed Kiss-of-Death (off-path)
  - ➢ **Denial of Service** by Priming the Pump (off-path)
  - ➢ Timeshifting by IPv4 Packet Fragmentation (off-path)

- Broadcast mode attacks



denial of service via spoofed kiss-o-death (KoD)



Kiss-o'-Death (KoD)

“Keep quiet for 2^{poll} 2^{17} sec!”

(36 hours!)

One packet prevents client from querying its servers for days or years!

TEST2 was not used for KoD!

v4 | IHL=20 | TOS | Total length = 76

TTL | Protocol = 17 | IP Header Checksum

Source IP

Destination IP

Source Port = 123 | Destination Port = 123

Length = 76 | UDP Checksum

LI | v4 | **Response** | Stratum | **Poll**

Root Delay

Root Dispersion

Reference ID = RATE

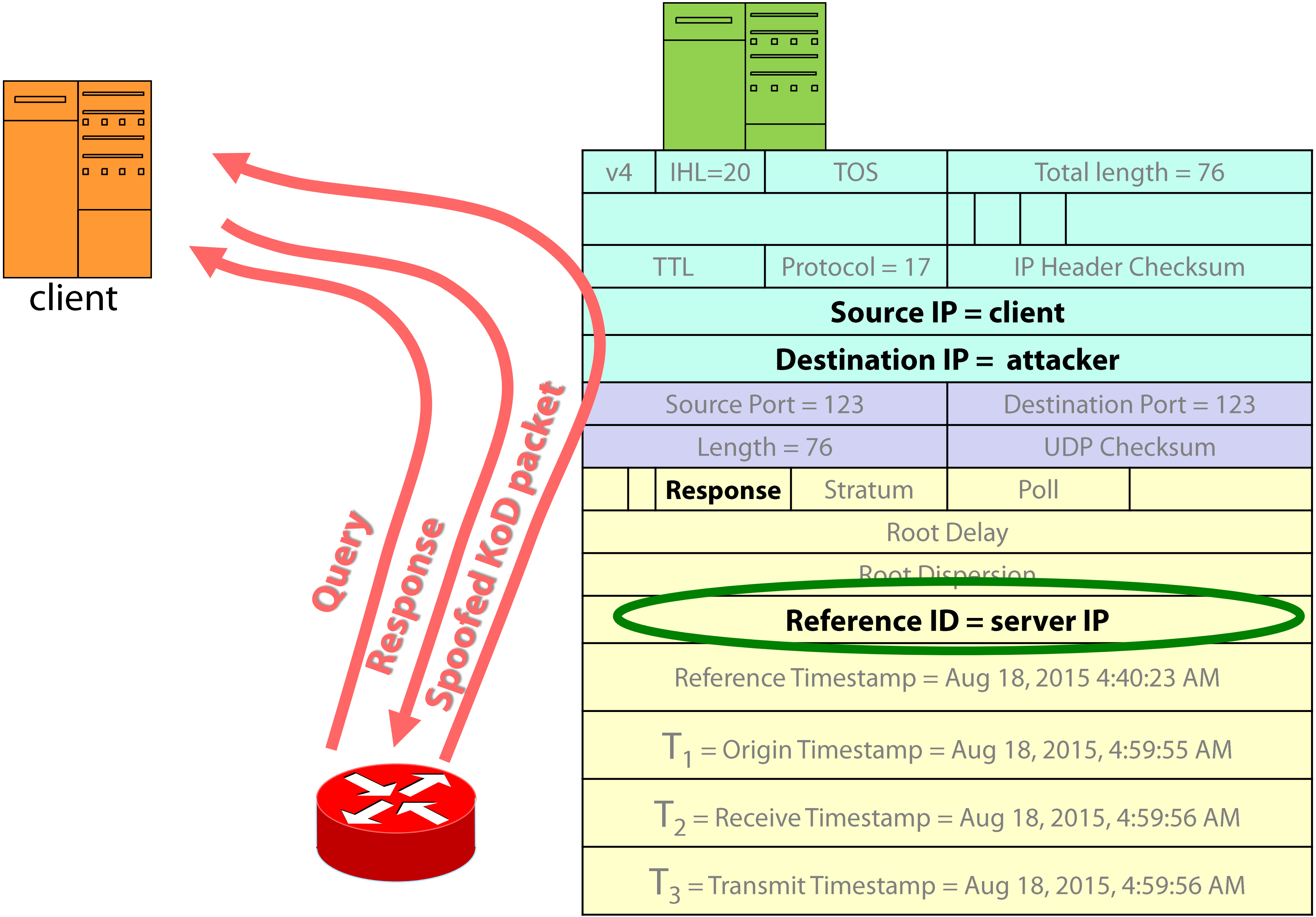
Reference Timestamp = Jan 1, 1970 0:00:00 UTC

T_1 = Origin Timestamp = July 29, 2015 01:23:45

T_2 = Receive Timestamp = July 29, 2015 01:23:45

T_3 = Transmit Timestamp = July 29, 2015 01:23:45

how to learn the server’s IP for the spoofed KoD?



v4 | IHL=20 | TOS | Total length = 76

TTL | Protocol = 17 | IP Header Checksum

Source IP = client

Destination IP = attacker

Source Port = 123 | Destination Port = 123

Length = 76 | UDP Checksum

Response | Stratum | Poll

Root Delay

Root Dispersion

Reference ID = server IP

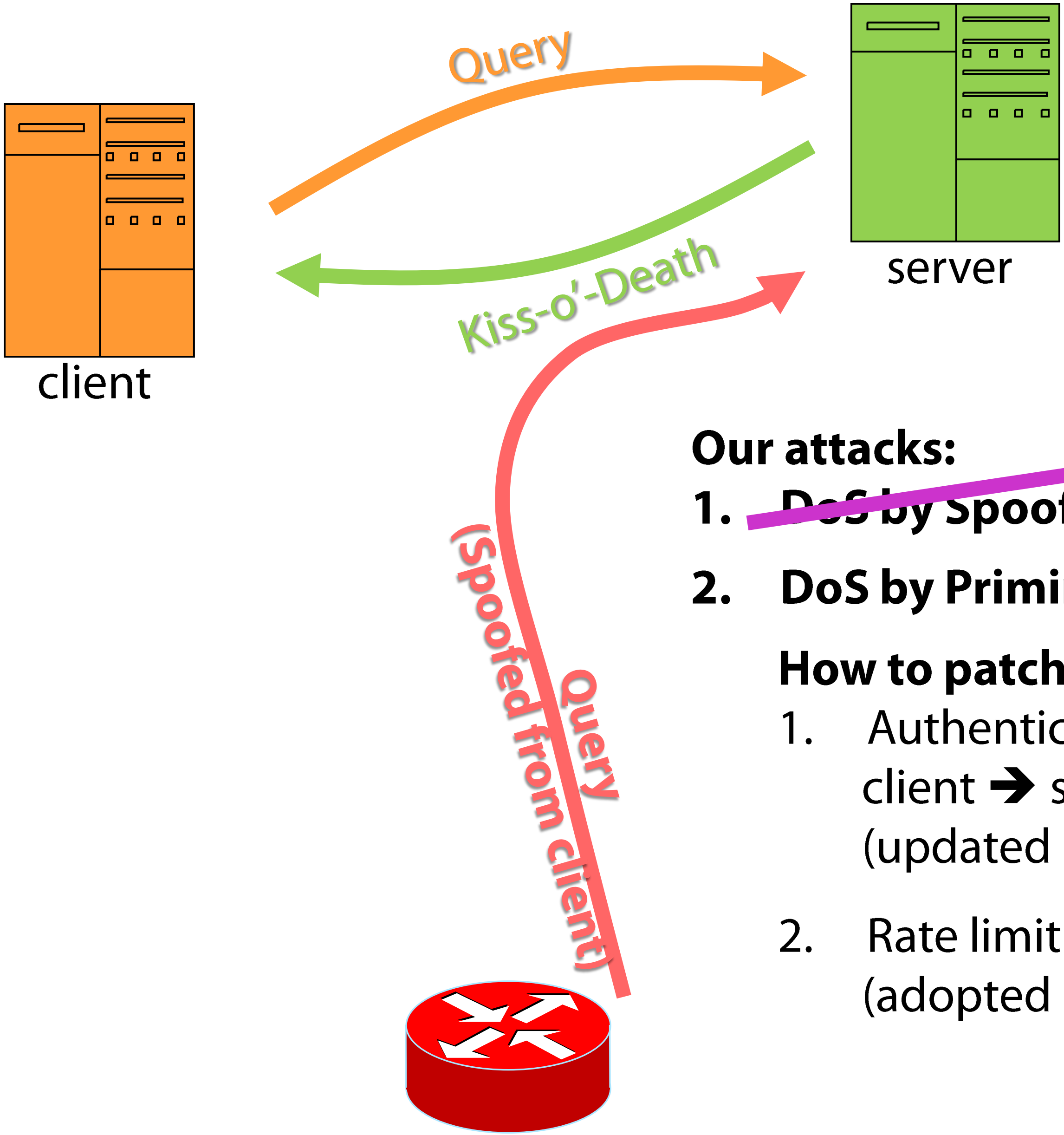
Reference Timestamp = Aug 18, 2015 4:40:23 AM

T_1 = Origin Timestamp = Aug 18, 2015, 4:59:55 AM

T_2 = Receive Timestamp = Aug 18, 2015, 4:59:56 AM

T_3 = Transmit Timestamp = Aug 18, 2015, 4:59:56 AM

denial of service by priming the pump



Patched!

TEST2 for KoD

ntpd 4.2.8p4

Our attacks:

1. ~~**DoS by Spoofed KoD (off-path)**~~
2. **DoS by Priming the Pump (off-path)**

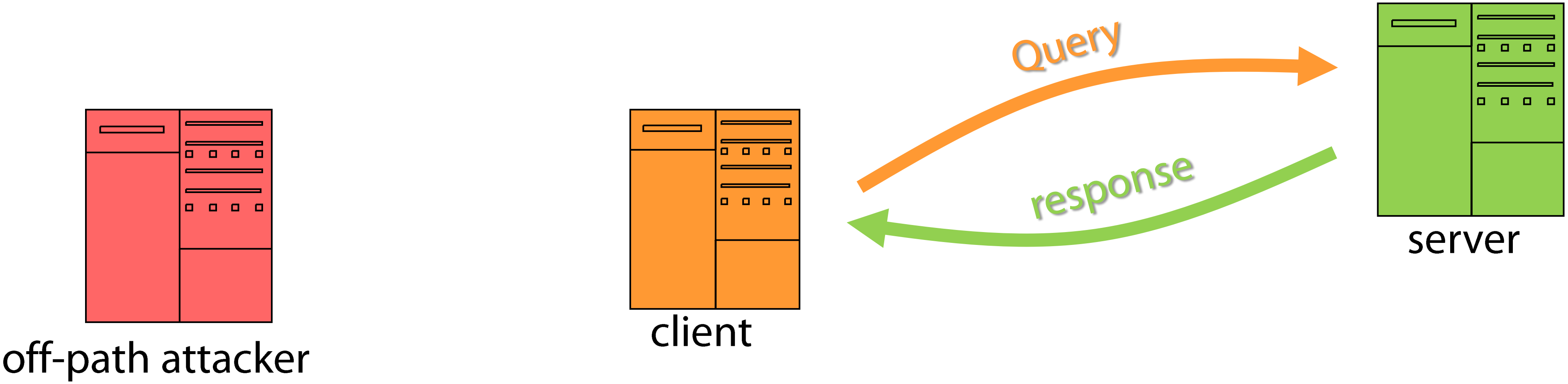
How to patch?

1. Authenticate both directions
   client ➔ server & server ➔client
   (updated Network Time Security IETF ID)

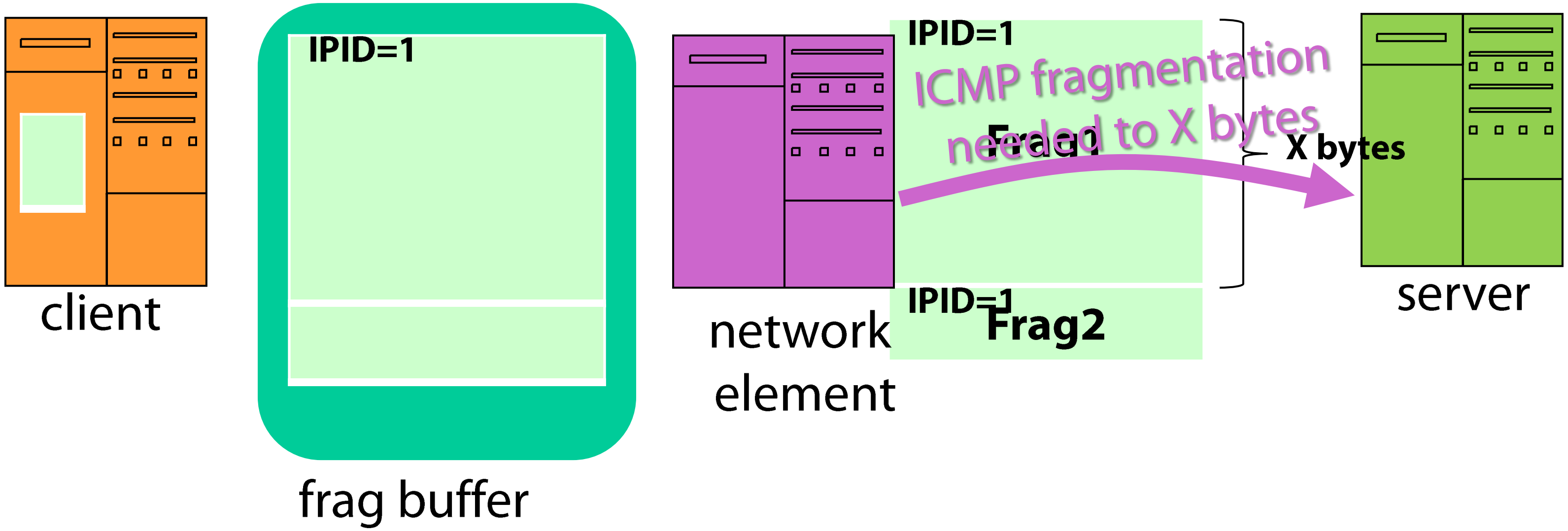
2. Rate limit like DNS Response Rate Limit'g
   (adopted by chrony, ntpd)

outline of the talk

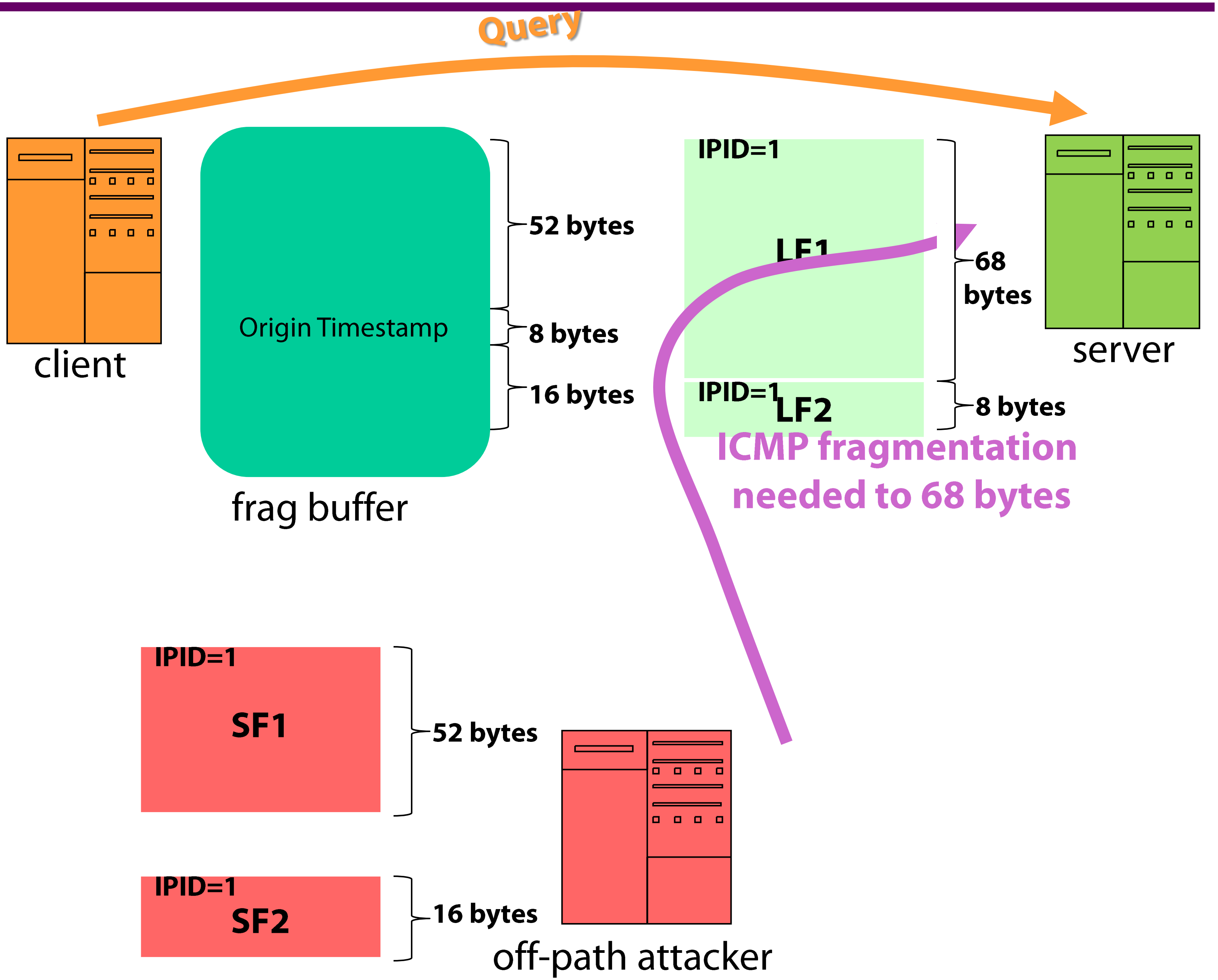
- Background
  - How does NTP work?
  - How does NTP client take time?
- Our client/server mode attacks
  - Denial of Service by Spoofed Kiss-of-Death (off-path)
  - Denial of Service by Priming the Pump (off-path)
  - **Timeshifting** by IPv4 packet fragmentation (off-path)
- Broadcast mode attacks



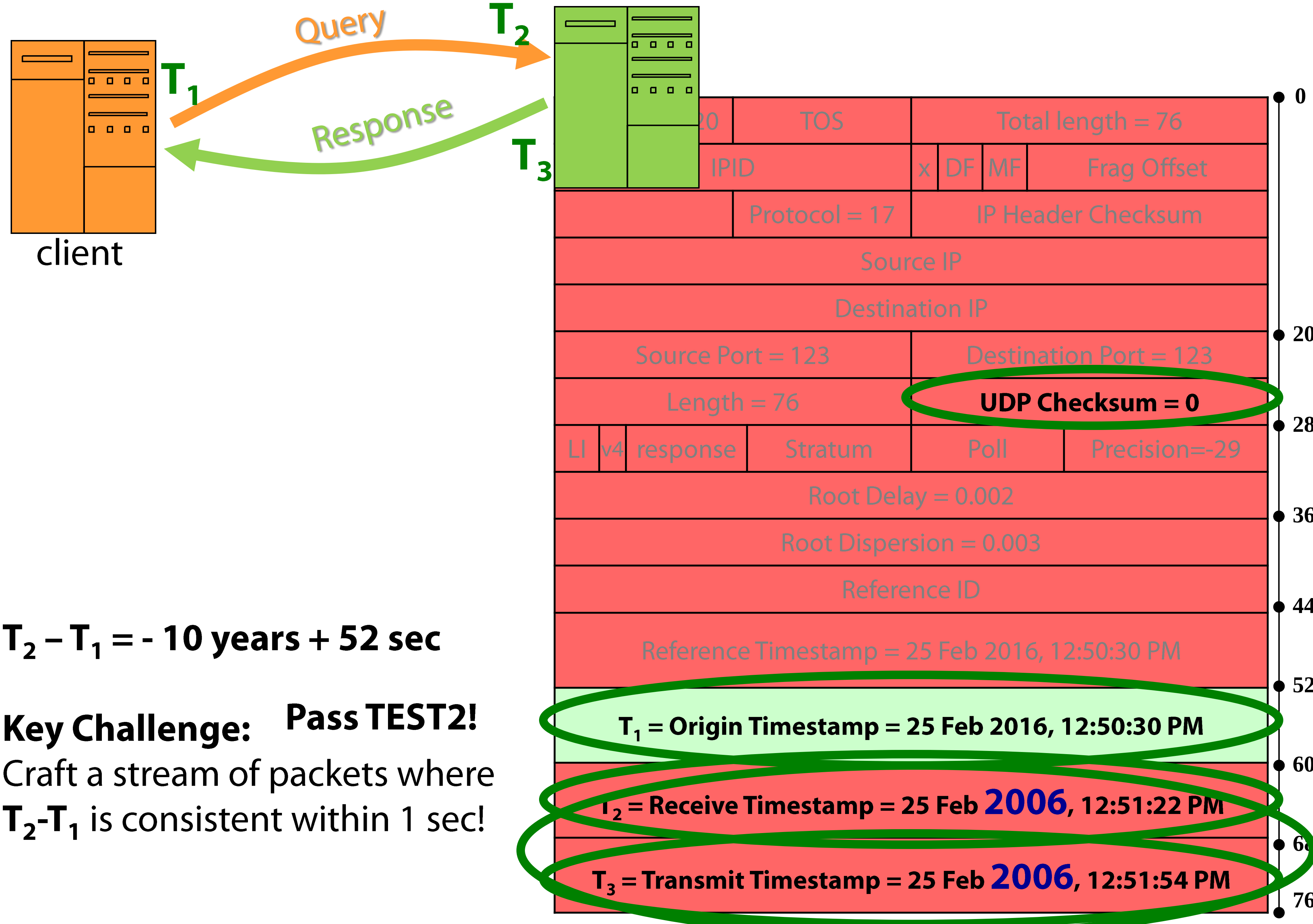
background: IPv4 fragmentation



exploiting IPv4 fragmentation to attack NTP



what does the reassembled packet look like?



20 | TOS | Total length = 76

IPID | x | DF | MF | Frag Offset

Protocol = 17 | IP Header Checksum

Source IP

Destination IP

Source Port = 123 | Destination Port = 123

Length = 76 | **UDP Checksum = 0**

LI | v4 | response | Stratum | Poll | Precision=-29

Root Delay = 0.002

Root Dispersion = 0.003

Reference ID

Reference Timestamp = 25 Feb 2016, 12:50:30 PM

T$_1$ = Origin Timestamp = 25 Feb 2016, 12:50:30 PM

T$_2$ = Receive Timestamp = 25 Feb 2006, 12:51:22 PM

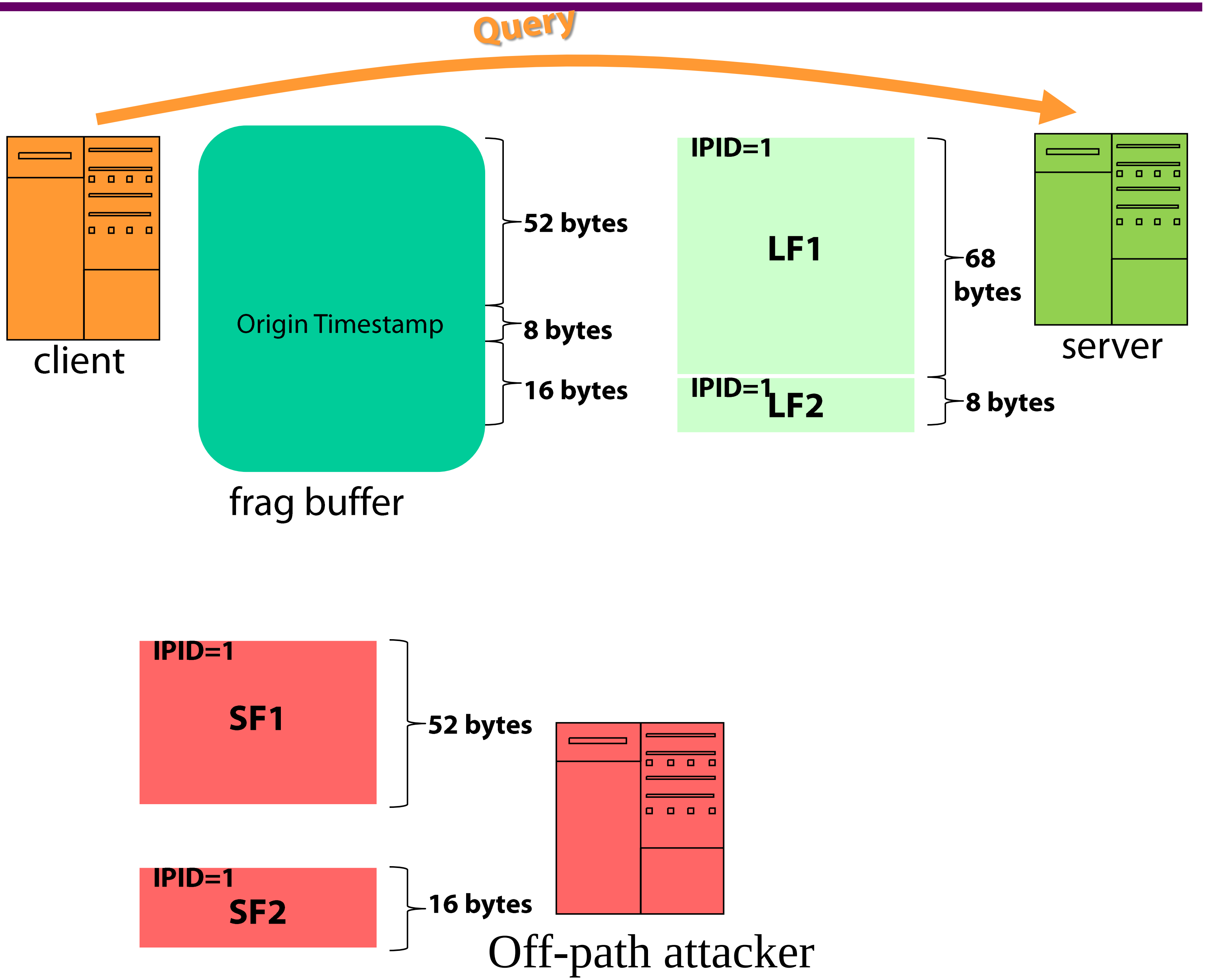
T$_3$ = Transmit Timestamp = 25 Feb 2006, 12:51:54 PM

T$_2$ – T$_1$ = - 10 years + 52 sec

Key Challenge: **Pass TEST2!**

Craft a stream of packets where T$_2$-T$_1$ is consistent within 1 sec!

challenge: construct a stream of consistent packets



Query

client

Origin Timestamp

frag buffer

52 bytes

8 bytes

16 bytes

IPID=1

LF1

IPID=1

LF2

68 bytes

8 bytes

server

IPID=1

SF1

52 bytes

IPID=1

SF2

16 bytes

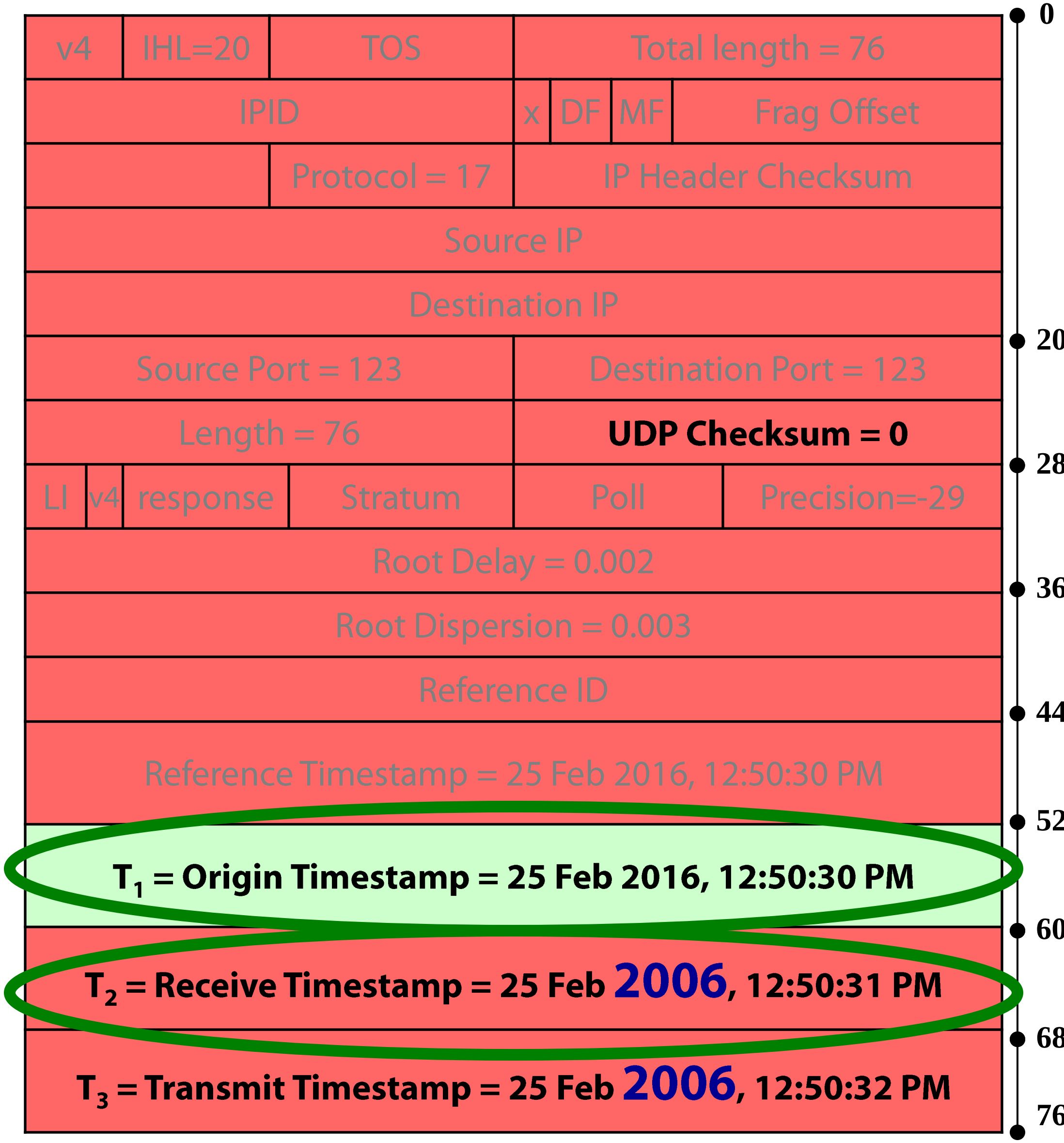
Off-path attacker

challenge: construct a stream of consistent packets

Why does this help?

The second spoofed fragment sits in the fragment buffer for no longer than 1 sec

$T_2 - T_1 = -10 \text{ years} + 1\text{sec}$



v4 | IHL=20 | TOS | Total length = 76

IPID | x | DF | MF | Frag Offset

Protocol = 17 | IP Header Checksum

Source IP

Destination IP

Source Port = 123 | Destination Port = 123

Length = 76 | **UDP Checksum = 0**

LI | v4 | response | Stratum | Poll | Precision=-29

Root Delay = 0.002

Root Dispersion = 0.003

Reference ID

Reference Timestamp = 25 Feb 2016, 12:50:30 PM

T_1 = Origin Timestamp = 25 Feb 2016, 12:50:30 PM

T_2 = Receive Timestamp = 25 Feb 2006, 12:50:31 PM

T_3 = Transmit Timestamp = 25 Feb 2006, 12:50:32 PM

attack surface for our NTP fragmentation attack

Conditions:

1. Server fragments NTP packets to 68 bytes
   - Out of 13M scanned NTP servers, 24K servers do this

2. Client reassembles overlapping fragments by the ”First” policy
   - Cannot safely measure due to teardrop [CA-1997-28]

3. Server uses incrementing IPID
   - Inferring globally-incrementing IPID is trivial (most vuln servers)
   - Infer per-destination IPID with **[Gilad-Herzberg’13]** and **[Knockell-Crandall’14]**

Recommendations:

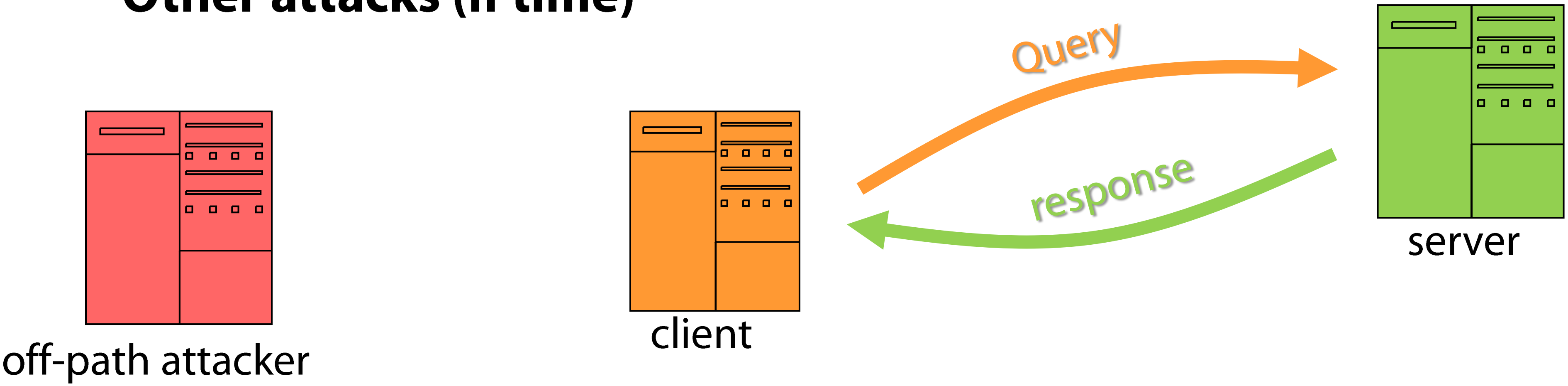
- Servers should not fragment to 68 bytes (Test servers on our site!)
- Drop overlapping IPv4 fragments!

outline of the talk

- **Background**
  - How does NTP work?
  - How does NTP client take time?

- **Our client/server mode attacks**
  - Denial of Service by Spoofed Kiss-of-Death (off-path)
  - Denial of Service by Priming the Pump (off-path)
  - Timeshifting by IPv4 Packet Fragmentation (off-path)

- **Broadcast mode attacks**
- **Other attacks (if time)**



background: broadcast mode

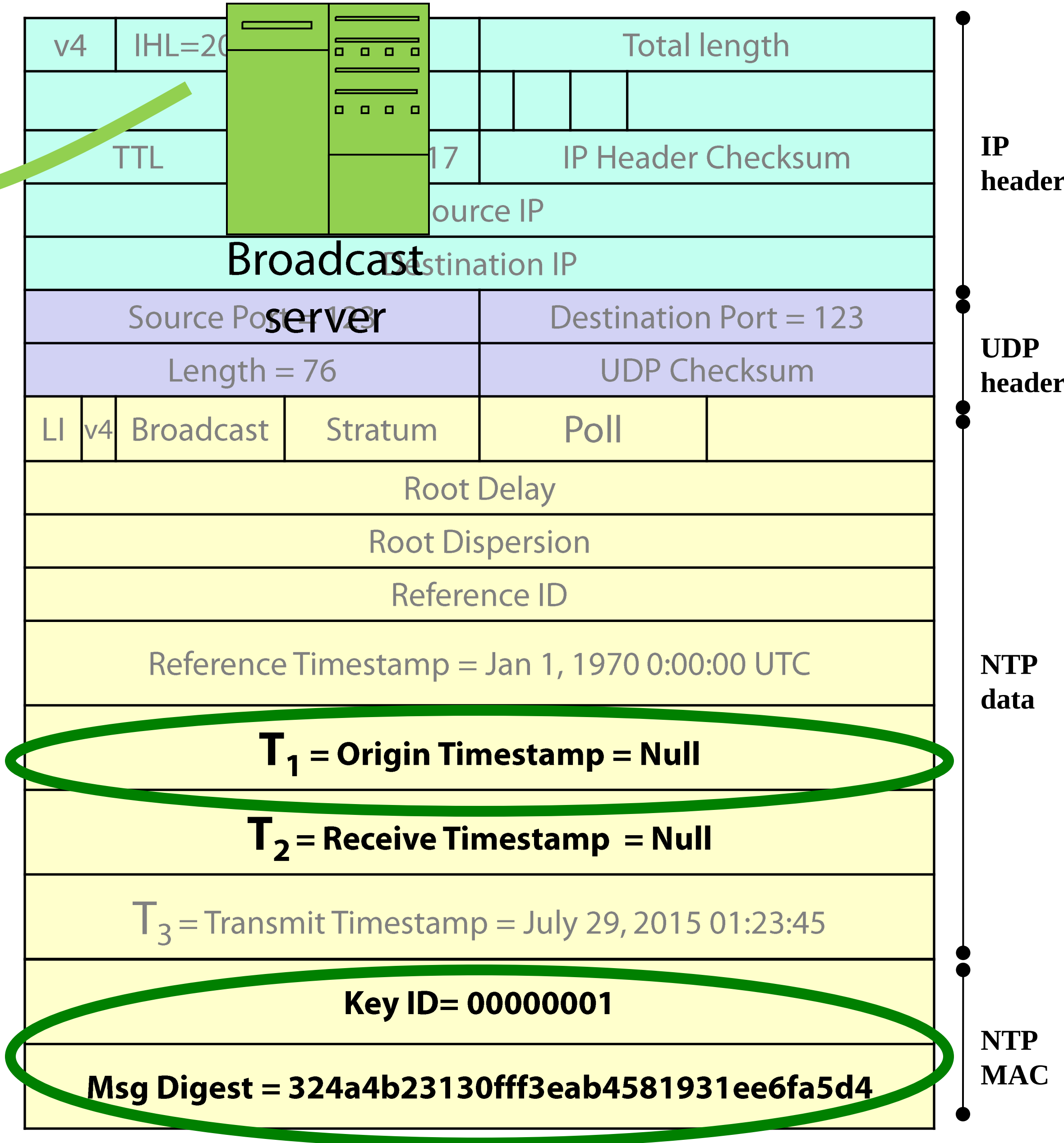
RFC5905 requires broadcast mode to be crypto authenticated

But why?

- TEST2 does not apply

RFC5905 says:

"If preconfigured to accept broadcast packets, the client accepts packets from ANY server that sends it broadcast packets."



v4 | IHL=20 | Total length

TTL | 17 | IP Header Checksum

Source IP

Destination IP

Source Port = 123 | Destination Port = 123

Length = 76 | UDP Checksum

LI | v4 | Broadcast | Stratum | Poll

Root Delay

Root Dispersion

Reference ID

Reference Timestamp = Jan 1, 1970 0:00:00 UTC

T_1 = Origin Timestamp = Null

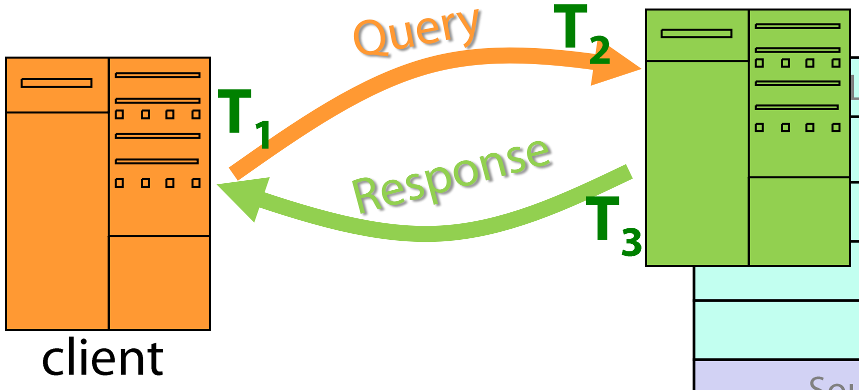
T_2 = Receive Timestamp = Null

T_3 = Transmit Timestamp = July 29, 2015 01:23:45

Key ID= 00000001

Msg Digest = 324a4b23130fff3eab4581931ee6fa5d4

how does a broadcast client detect replay attacks?



IP header:

L=20 | TOS | Total length

Protocol = 17 | IP Header Checksum

Source IP

Destination IP

UDP header:

Source Port = 123 | Destination Port = 123

Length = 76 | UDP Checksum

NTP data:

LI | v4 | Broadcast | Stratum | Poll

Root Delay

Root Dispersion

Reference ID

Reference Timestamp = Jan 1, 1970 0:00:00 UTC

T_1 = Origin Timestamp = NULL

T_2 = Receive Timestamp = NULL

T_3 = Transmit Timestamp = March 22, 2016 01:53:45

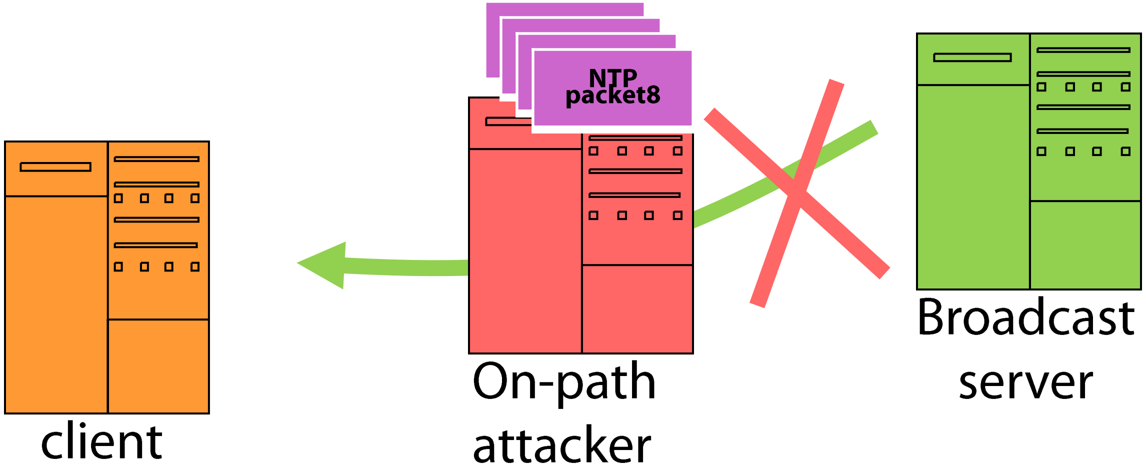
NTP MAC:

Key ID= 00000001

Message Digest = 324a4b23130fff3eab4581931ee6fa5d4

TEST1: Prevents replay of **most recent** packet!

déjà vu: time sticking attack via packet replay

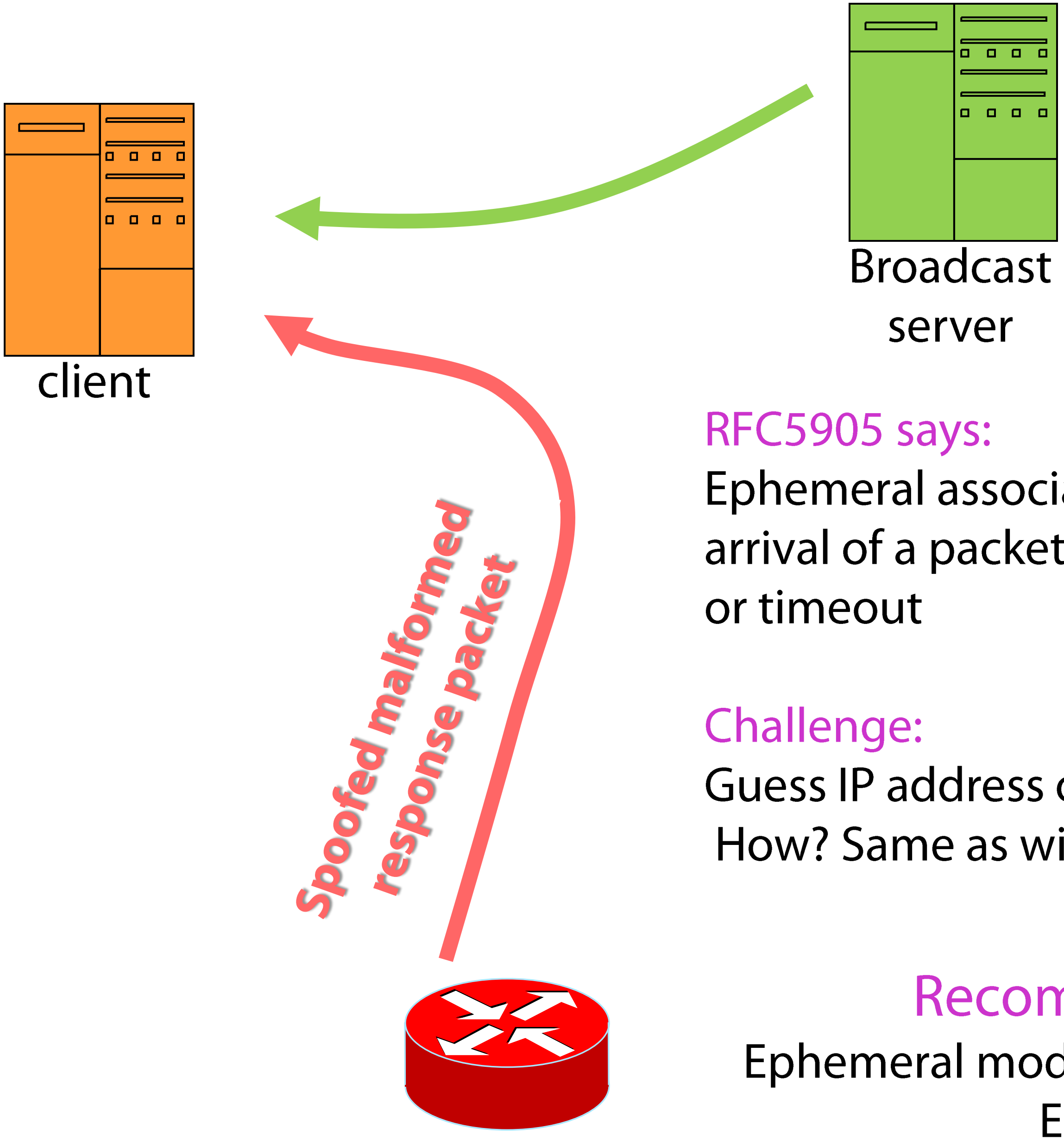


Attack: Replay, in order the eight most recent timestamps

Recommendation:

Add an incrementing counter to the (timestamp) fields that are null

off-path DoS attack using malformed crypto packets



RFC5905 says:

Ephemeral associations are mobilized upon arrival of a packet and demobilized upon error or timeout

Challenge:

Guess IP address of the server.

How? Same as with our KoD attacks

Recommendation:

Ephemeral modes considered harmful.

Eliminate.

other recent attacks

The dreaded NTP DDoS Reflection attack

- this is still a problem
- send the monlist control query to an NTP host (via UDP!)
- … get a list of last 600 IPs interacting with that host.

Cisco ASIG attacks on NTP

- zero origin timestamp attack (from RFC5905)
- NAK to the future (crypto NAK implementation flaw)
- crypto key misbinding leading to a sybil attack vulnerability
- origin timestamp leak vulnerability
  - Use NTP control queries to learn exact value of origin timestamp
  - … and bypass TEST2.

recommendations for NTP users

- **Firewall your NTP instances**.
  - End-hosts should not accept **anything** other than mode 4 packets from their preconfigured servers.
  - All hosts (incl. servers) should not accept **any** control queries (not just monlist) from arbitrary IPs. (Use the ntpd **`noquery`** option)
  - Firewalls should block KoD (mode 4) with high poll (eg poll >10)

- If ntpd is configured with the –g option, monitor for reboots
  - NTP is much more vulnerable to attacks when it reboots

- Don’t use broadcast mode except in a safe firewalled network
  - Even if your packets are authenticated, you are still vulnerable.
  - Make sure no broadcast packets come into your network from outside

- Don’t fragment NTP packets

longer recommendations for securing NTP

- Stop leaking so much information.
  - NTP packet leak the reference ID and reference time.
  - NTP control packets leak timestamps and lots of internal state
  - All this information can be collected using UDP
  - And in most cases is available by default

- Get rid of the KoD, use RRL style rate limiting instead

- Use a modern control protocol instead of leaky UDP control protocol

- Latest burst of bugs has shown that RFC5905 is underspecified

- Lots more work to be done to develop crypto for NTP

Questions?

Attacking the Network Time Protocol
Aanchal Malhotra, Isaac E. Cohen, Erik Brakke and Sharon Goldberg
NDSS'16, San Diego, CA. Feb 2016.

Attacking the NTP's Authenticated Broadcast Mode
Aanchal Malhotra and Sharon Goldberg
SIGCOMM Computer Communication Review, April 2016.

http://www.cs.bu.edu/~goldbe/NTPattack.html