University of Connecticut
Computer Science and Engineering
CSE 4402/5095: Network Security

TCP/IP Security

©Amir Herzberg

TCP/IP Security

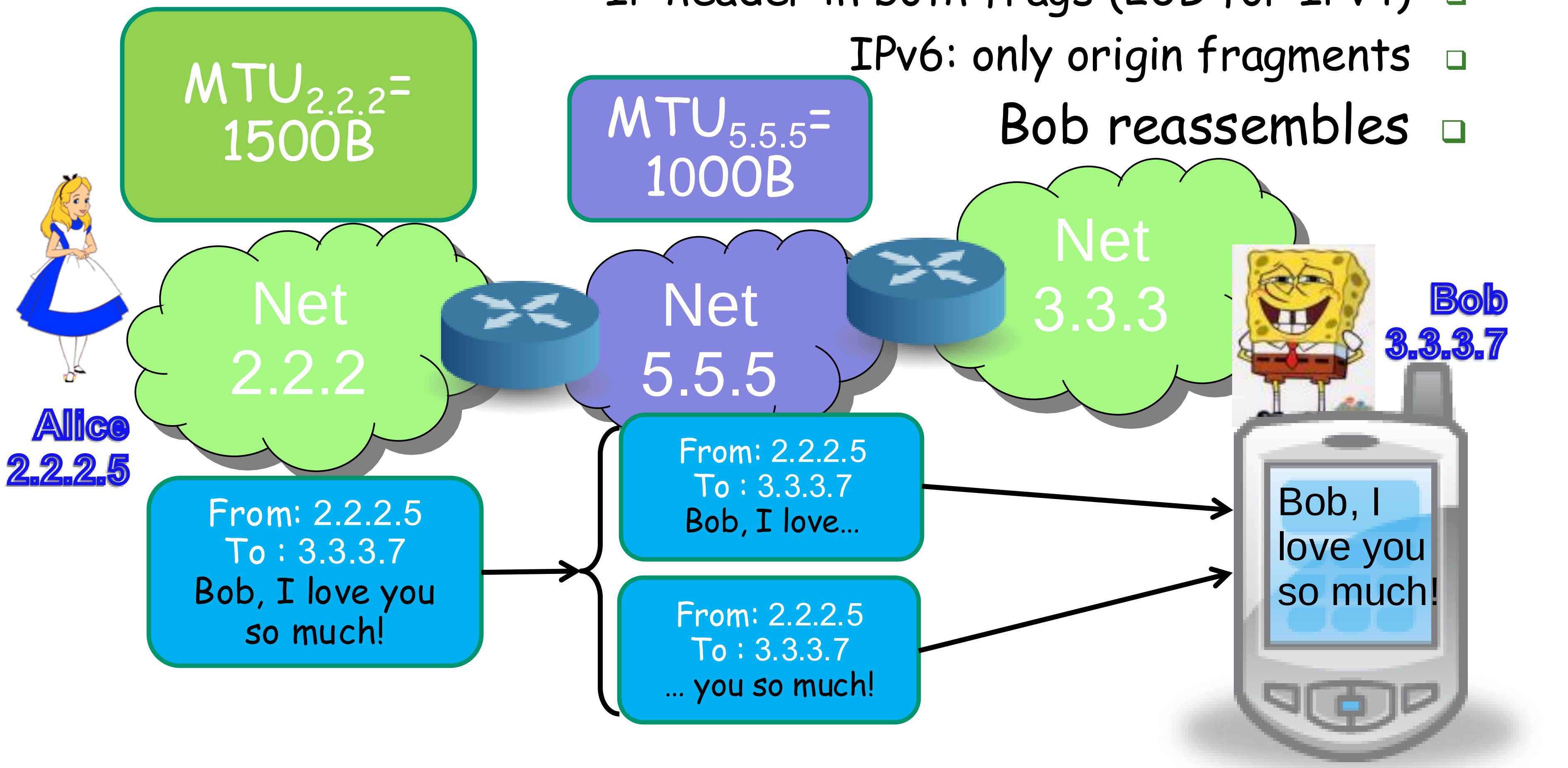
- **Internet Protocol (IP) Security**
  - **Fragmentation attacks**
  - IPsec
- Secure Transport
  - TCP injections and other attacks
  - Quic
- Conclusion

IP Security: Goals, Models, Expectations

Attacker models → **Security goals**	**MitM**	**Eavesdropper**	**Off-Path**
Confidentiality and privacy	None (without IPsec)	None (without IPsec)	Expected
Integrity and authentication	None (without Ipsec)	Trivial	Expected: spoofing, but no modification
Availability (and efficiency)	None	Trivial	Expected (except by clogging)

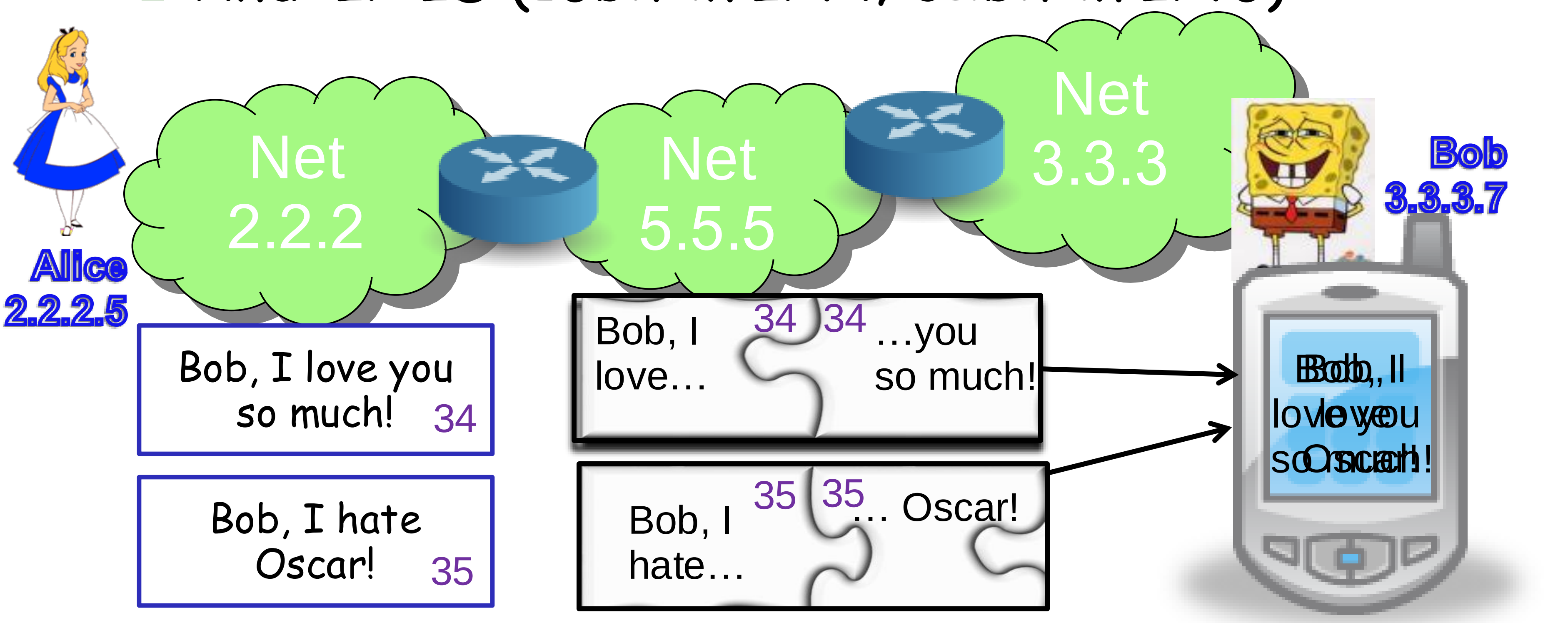
The Internet Protocol: Fragmentation

- Every network has a size-limit on packet size (MTU)
- What if we need to send more?
- Solution: Fragmentation
  - IP header in both frags (20B for IPv4)
  - IPv6: only origin fragments
  - Bob reassembles



Packet Reassembly: Careful(!)

- Bob receives fragments of multiple packets
- How to reassemble without mixing?
- Identify each packet
  - By Src, Dst addresses and protocol
  - And: IP-ID (16bit in IPv4; 32bit in IPv6)



‘Fragmentation considered Harmful’

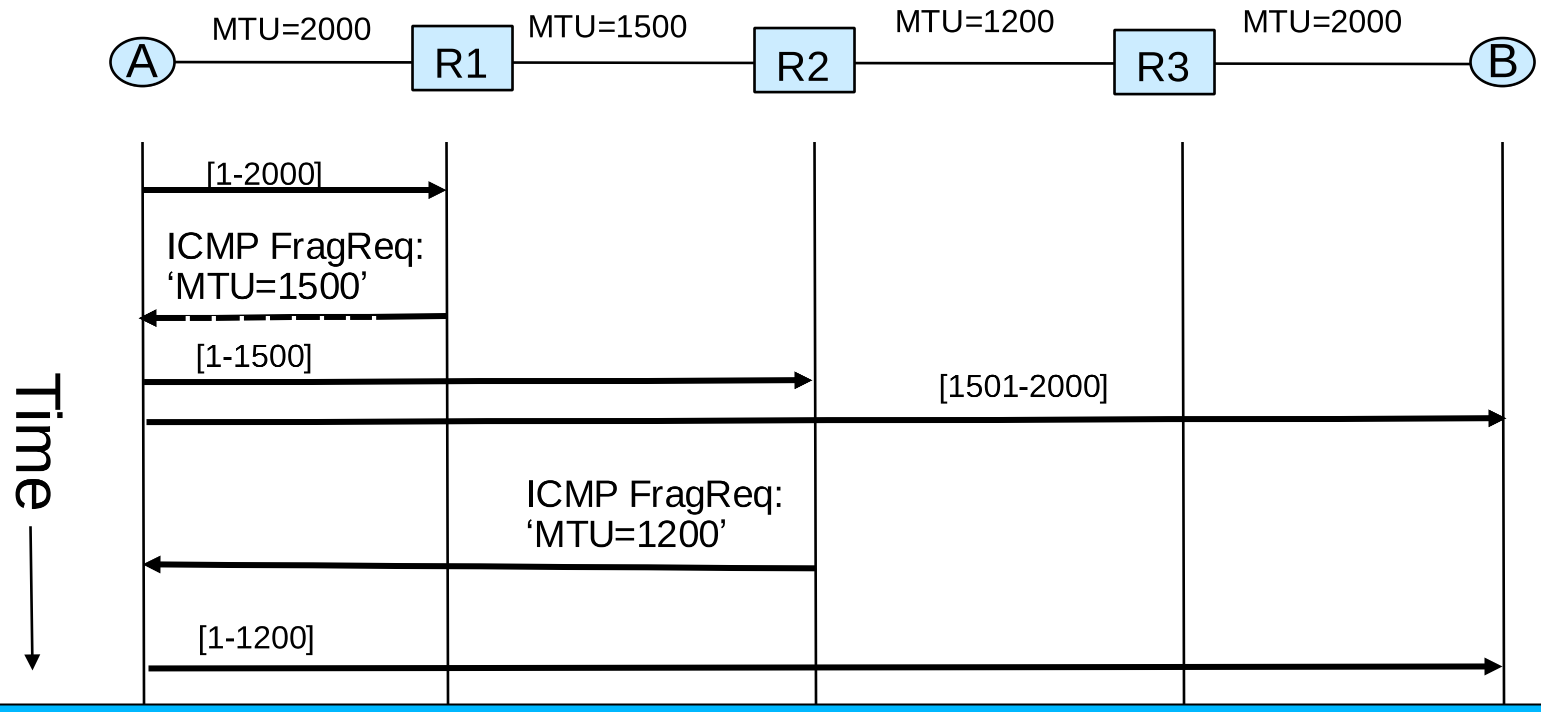
- IP fragmentation is conceptually easy, but…
  - Complexities: may arrive late or out of order
  - What if fragments overlap? How much storage?
  - Significant overhead
- ➔ Can we avoid fragmentation ? How?
- Always send only short packets
  - IPv4 allows short MTUs, but very rarely used
  - IPv6: MTU always at least 1200B
- Find minimal MTU on path
  - pMTUd: Path-MTU discovery
  - ➔ Limit packets to Path-MTU → no frag!

Avoiding Fragmentation

- Always send only short packets
  - DNS: originally limited to 512 bytes payload
  - But later, Extended-DNS allows longer packets
  - IPv6: all links have MTU>1200B
    - No fragmentation – except at source (UDP; why?)
- **Path MTU discovery (pMTUd)**
  - Send packets with Do not Fragment (DF) flag
  - Router sends back ICMP `Frag required` with MTU
  - Use path-MTU segments (TCP) or source-frag
- Works! few (<0.5%) packets fragmented
  - Yet fragmentation is still used
    - Mainly: UDP, often with <u>source-fragmentation</u>

Path MTU discovery (pMTUd)

- All packets sent with DF (don't fragment) bit set
- Periodically (rarely) try again to use larger MTU



Resending [1-1200] etc.:

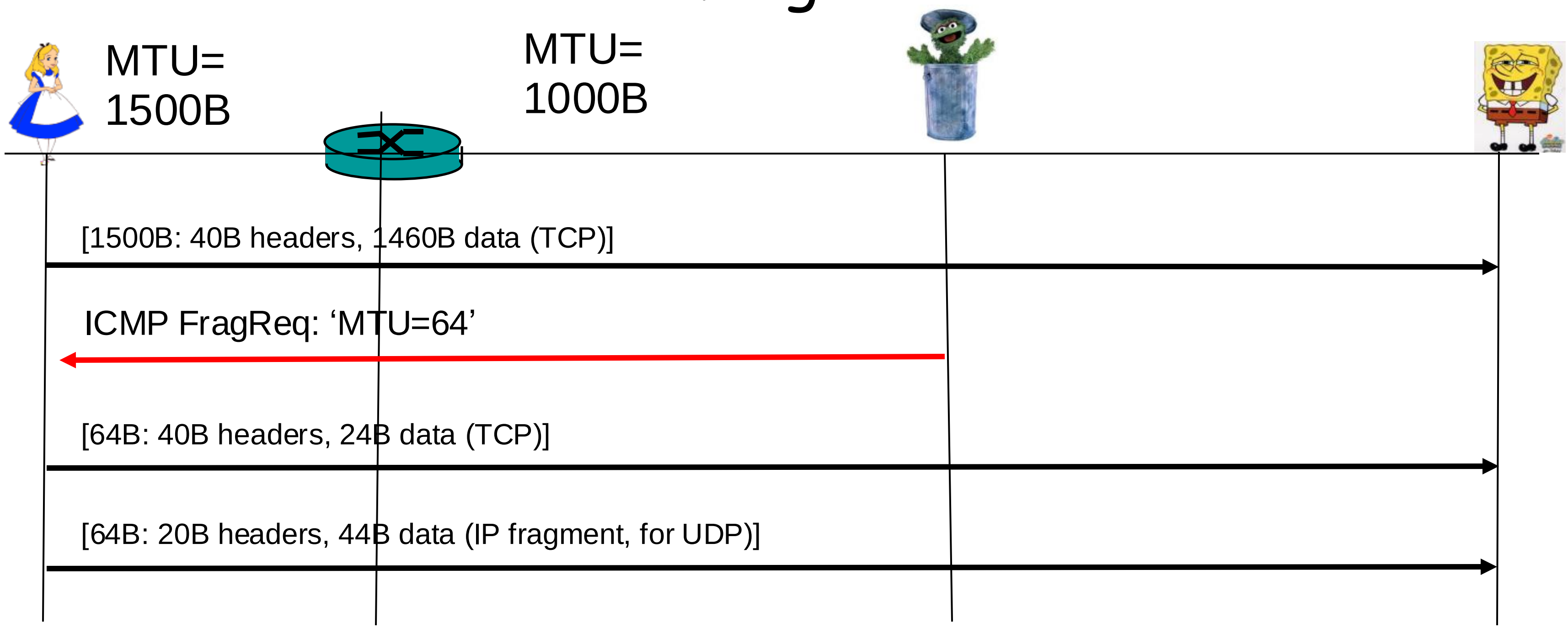
source-fragmented (UDP) or as new (shorter) packet (TCP)

Fragmentation attacks on IP

Attacker models → / Security goals	MitM	Eavesdropper	Off-Path attack Exploiting IP-ID
Confidentiality and privacy	Trivially broken without crypto	Broken (without IPsec)	2nd Frag interception attack
Integrity and authentication	Trivially broken without crypto	Trivial	2nd Frag spoofing attack
Availability (and efficiency)	Trivially broken	Trivial	Frag-based packet drop and overhead attacks
Stealthy scan	Trivially broken	Trivial	Off-path stealthy TCP scan

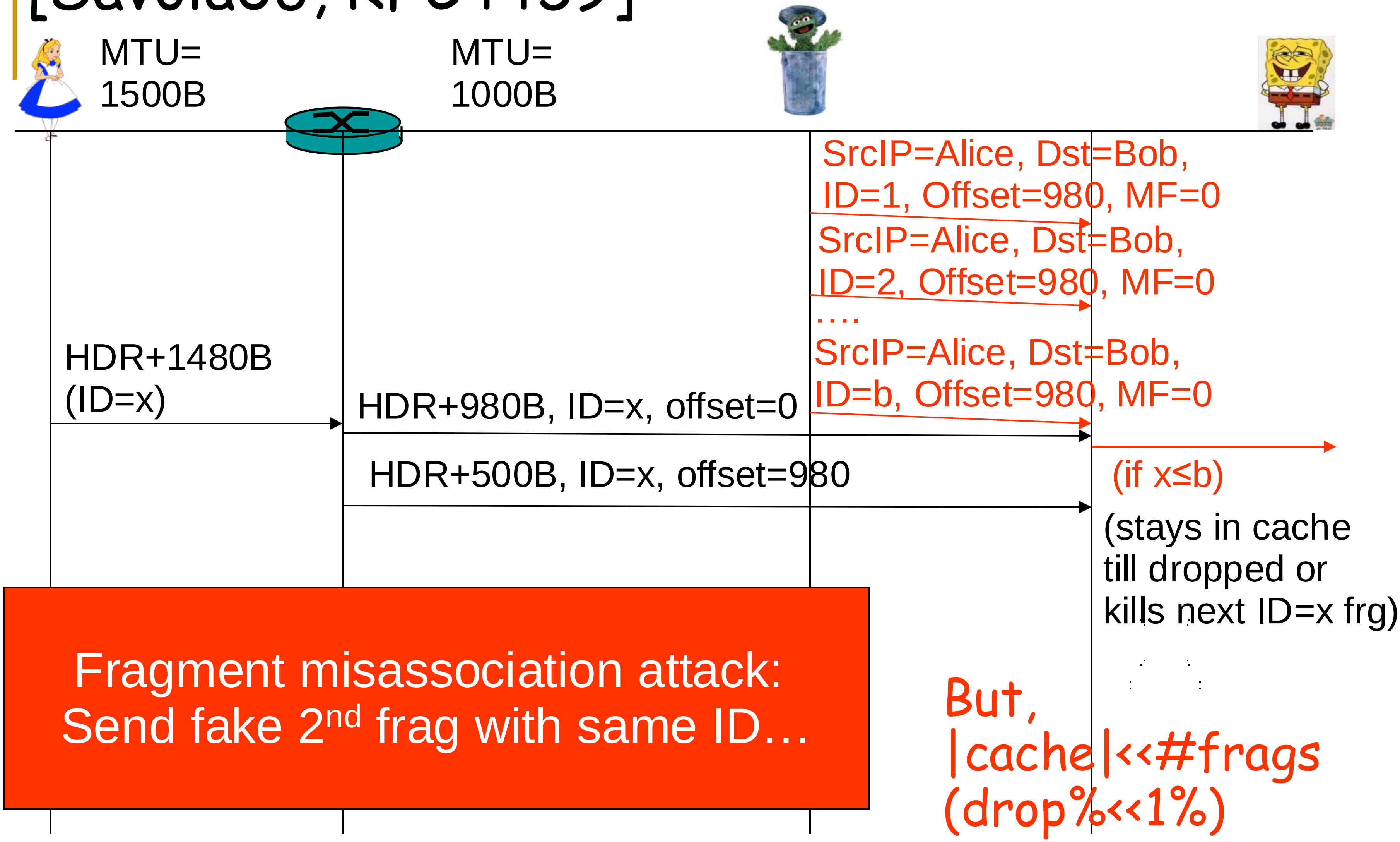
Spoofed ICMP-Frag-Req Attack

- All links have same MTU (say, 1500B)
- Oscar sends ICMP Frag-Req to Alice
- Alice uses short MTU (TCP) or fragments at source (UDP - even in IPv6)
  - So let's see these frag attacks…



Fragment Misassociation Attack

[Savola06, RFC4459]

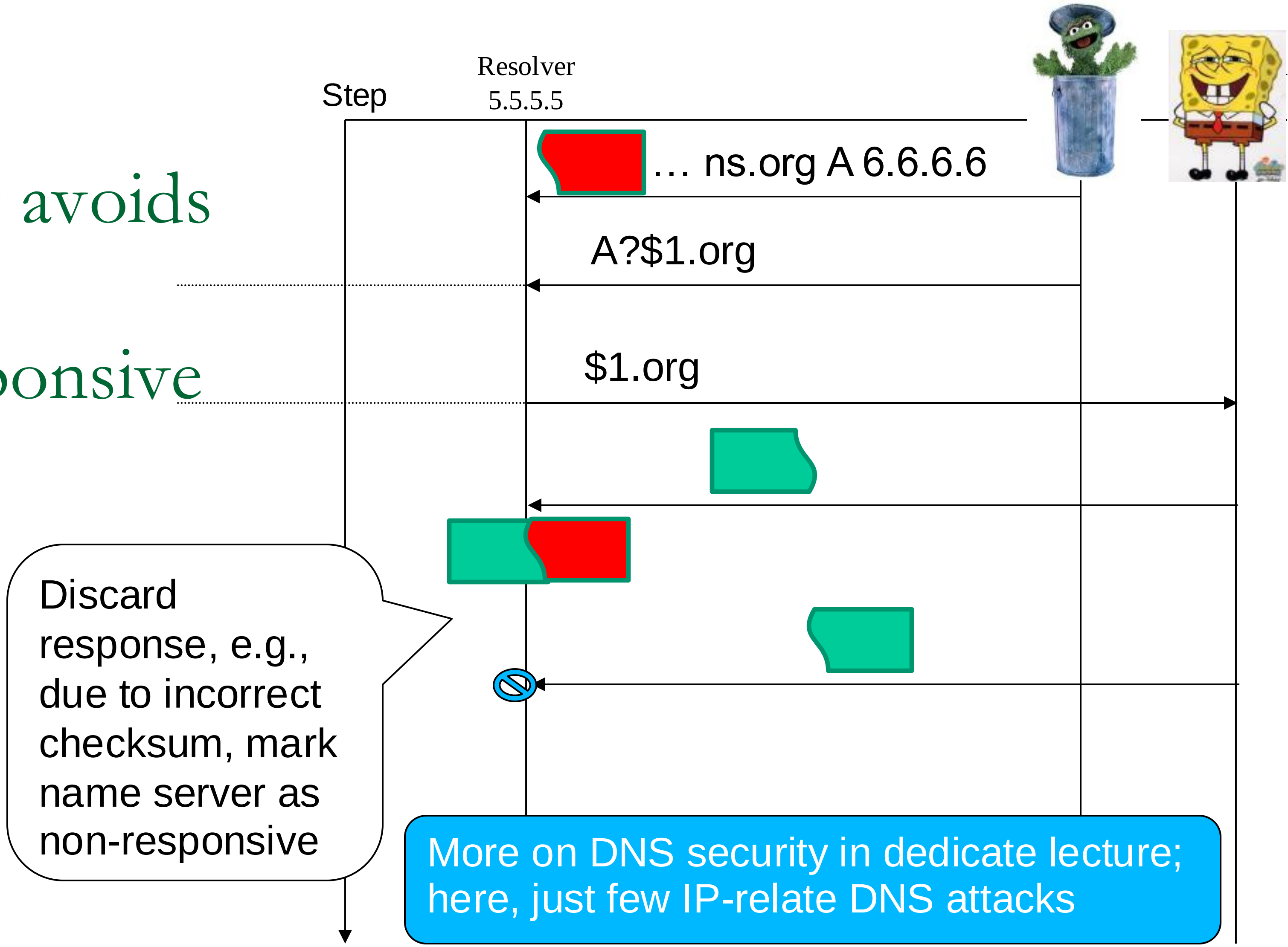


What if Oscar can *find* the next IP-ID?

- **Attacker can 'kill' packets (DoS)**
- **Assuming fragments are sent in-order:**
  - Send 2nd fragment with next IP-ID
  - Cached by host (or GW, NAT)
  - When 1st fragment arrives (same srcIP, IP-ID):
    - Packet is reconstructed
    - UDP/TCP checksum error → packet discarded
    - Same as Savola's attack, but high success rates!
- Exercise: 'kill' packet when fragments are sent in reverse order (2nd fragments, then 1st)
- Application: block ('kill') DNS responses. Why?
  - So we have more time to poison
  - So resolver will use attacker-chosen name server: **transitive trust attack on DNS**

DNS Response Blocking

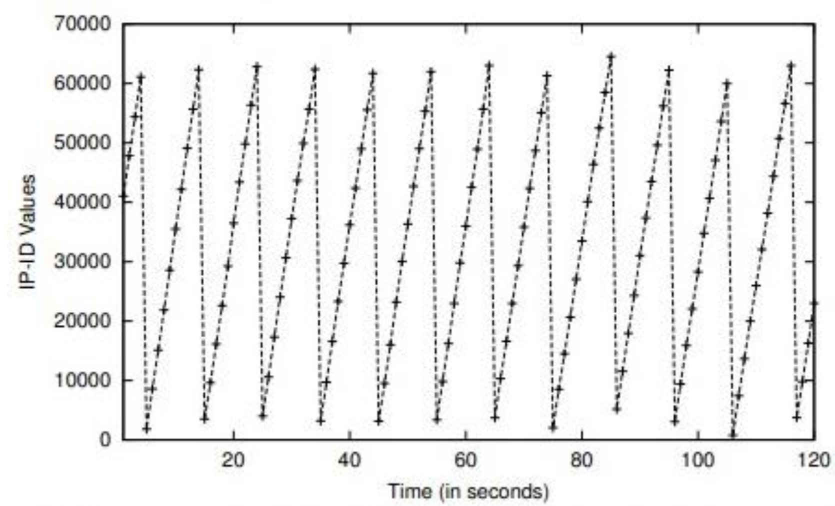
Resolver avoids querying non-responsive name servers



More on DNS security in dedicate lecture; here, just few IP-relate DNS attacks

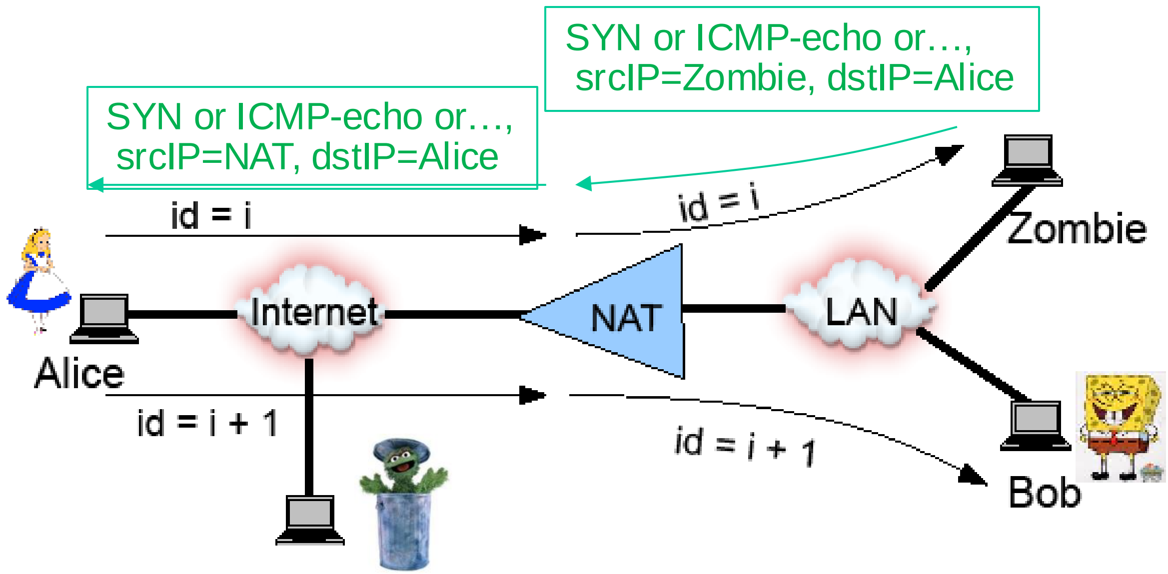
Can attacker *find* the IP-ID?

- How is the IP ID chosen (by the sender)?
  - Usually a counter – this is recommended in spec
    - To reduce probability of collisions (birthday paradox)
  - Three main approaches:
    - Per-destination incrementing (e.g., Linux)
    - Global incrementing (e.g., Windows, FreeBSD)
    - Mixed (Linux, from 4.1.8), see [1]
  - With all, it is often possible to find IP-ID
  - Can we find globally-inc
    - Send request to name server
    - Observe IP-ID in response
    - Measured at NS of ORG
    - And per-dest-inc IP-ID?



Sometimes, finding per-dest IP-ID is Easy!

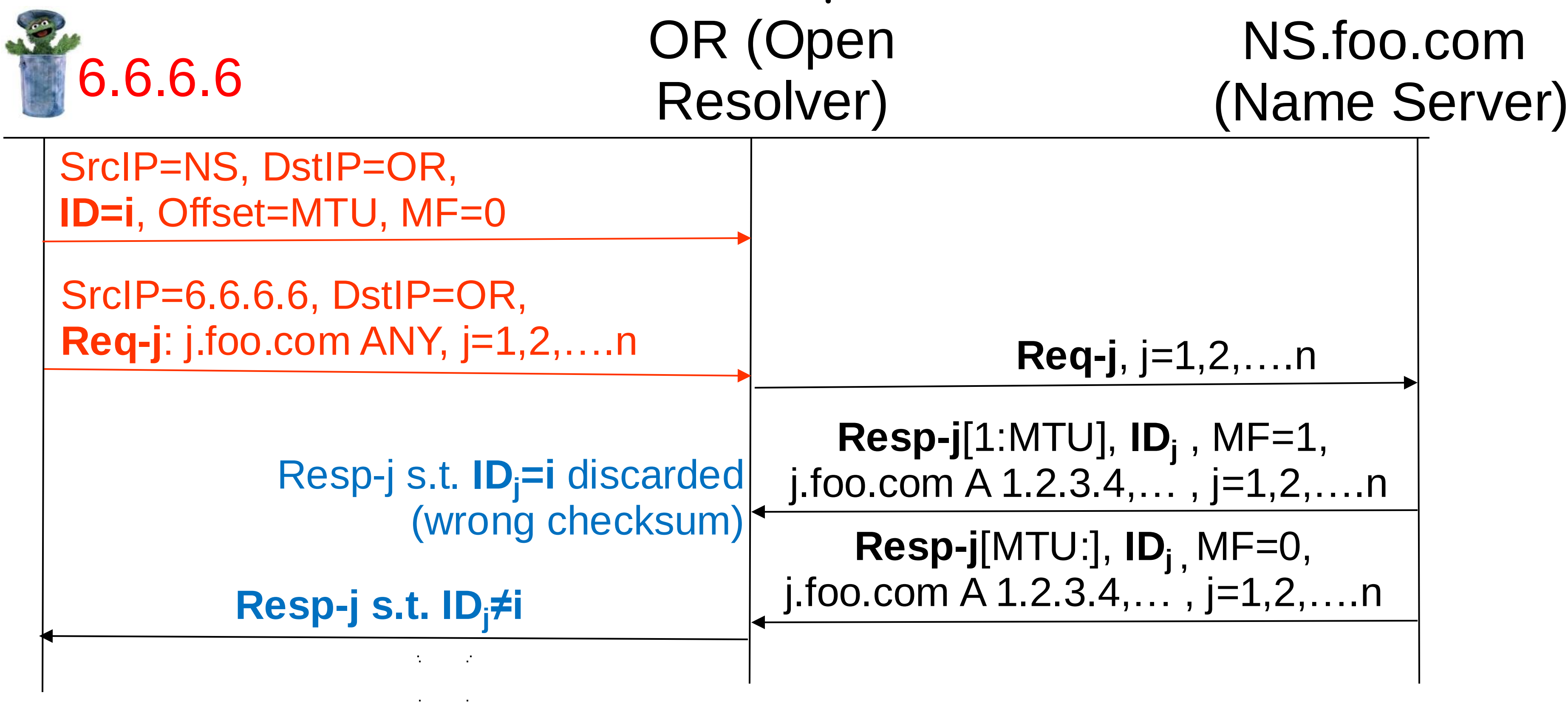
- Assume Oscar has an adversarial agent behind a NAT with the destination



- But why should Alice send a packet to the Zombie??
- Zombie may 'ping' Alice (in different ways), learn the current IP-ID Alice uses to send to NAT's IP from response
- Allows to intercept 2nd fragment by rewriting header(!)
- We'll return to this later…
- First: another way to find per-destination IP-ID

Finding ***per-destination*** *IP-ID*

- Cause and detect loss to find per-dest IP-ID
- Example: find IP-ID of Name Server
  - Destination is open resolver
- Cause loss: send spoofed 2nd frags to resolver
- Detect loss: timeout of responses from resolver



Finding ***per-destination*** *IP-ID*

- <u>Cause and detect loss</u> to find per-destination IP-ID
- Example: find IP-ID of Name Server (sending to OR)

6.6.6.6

OR (Open Resolver)

NS.foo.com (Name Server)

SrcIP=NS, DstIP=OR,
ID=i, Offset=MTU, MF=0

SrcIP=6.6.6.6, DstIP=OR,
Req-j: j.foo.com ANY, j=1,2,….n

Req-j, j=1,2,….n

Resp-j[1:MTU], ID_j , MF=1,
j.foo.com A 1.2.3.4,… , j=1,2,….n

Resp-j s.t. ID_j=i discarded
(wrong checksum)

Resp-j[MTU:], MF=0,
j.foo.com A 1.2.3.4,… , j=1,2,….n

Resp-j s.t. ID_j≠i

Detects timeout (loss) of Resp-j*
Concludes: ID_{j*}=i
Hence: next IP-ID = i+(n-j)+1

Minor error, can you spot?

Finding ***per-destination*** *IP-ID*

- Cause and detect loss to find per-destination IP-ID
- Example: find IP-ID of Name Server (sending to OR)

6.6.6.6

OR (Open Resolver)

NS.foo.com (Name Server)

SrcIP=NS, DstIP=OR,
ID=i, Offset=MTU-**20**, MF=0

SrcIP=6.6.6.6, DstIP=OR,
Req-j: j.foo.com ANY, j=1,2,….n

Req-j, j=1,2,….n

Resp-j[1:MTU-**20**], ID_j , MF=1,
j.foo.com A 1.2.3.4,… , j=1,2,….n

Resp-j s.t. ID_j=i discarded
(wrong checksum)

Resp-j[MTU-**21**:], MF=0,
j.foo.com A 1.2.3.4,… , j=1,2,….n

Resp-j s.t. ID_j≠i

Detects timeout (loss) of Resp-j*
Concludes: ID_{j*}=i
Hence: next IP-ID = i+(n-j)+1

Challenges:

- ORs don't allow ANY query
- Ensuring fragmented response
- What is n? can it be smaller?

Ensuring Fragmented Responses

- Method 1: Query to attacker's subdomains
  - For NS or .org, register subdomains: One-Domain-to-Rule-them-All.org
    - With many name servers with long names
    - ➔ Fragmented response
    - Or: apply attack to find IP-ID used by 'large' open resolver, responding to 'client OR'
- Method 2: Spoofed ICMP `fragmentation req' packet.

Find ***per-dest*** *IP-ID:* battleship optimization

- As presented, we need $n = 2^{16}$ queries to be sure we'll have Resp-j such that $ID_j = i$
- Goal: finding IP-ID of NS (to OR) with less queries
- Idea: send spread 'missiles' (like in Battleship game)

Find ***per-dest*** *IP-ID:* battleship optimization

- Goal: finding IP-ID of NS (to OR) with <u>less queries</u>
- Idea: send multiple 'missiles' (like Battleship game)

6.6.6.6

OR (Open Resolver)

NS.foo.com (Name Serve

SrcIP=NS, DstIP=OR,
ID∈ $\{i_1, \ldots, i_m\}$, Offset=MTU-**20**, MF=0

SrcIP=6.6.6.6, DstIP=OR,
Req-*j*: j.foo.com ANY, $j = 1,2, \ldots . n$

$m \cdot n = 2^{16}$

Req- *j*, $j = 1,2, \ldots . n$

Resp- *j*[1:MTU-**20**], ID_j , MF=1,
j.foo.com A 1.2.3.4,… , $j = 1,2, \ldots . n$

Resp- ***j*** s.t. $ID_j \in \{i_1, \ldots, i_m\}$
discarded (wrong checksum)

Resp- *j*[MTU-**19**:], MF=0,
j.foo.com A 1.2.3.4,… , $j = 1,2, \ldots . n$

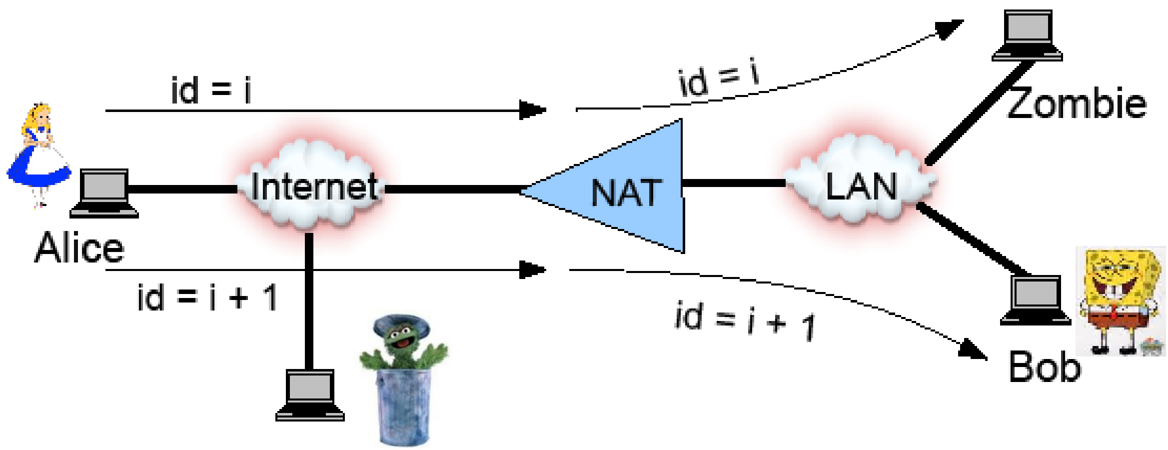
Resp-j s.t. $ID_j \notin \{i_1, \ldots, i_m\}$

Detects timeout (loss) of Resp-j*
Concludes: $ID_j \in \{i_1, \ldots, i_m\}$, hence: next IP-ID ∈ $\{i_1 + n - j + 1, \ldots\}$
How can attacker find exactly next IP-ID?

Hint: $\log(m)$ 'rounds' to divide-and-conquer the m possible values

Recall: finding per-dest IP-ID behind NAT

- Assume Oscar has an adversarial agent behind a NAT with the destination

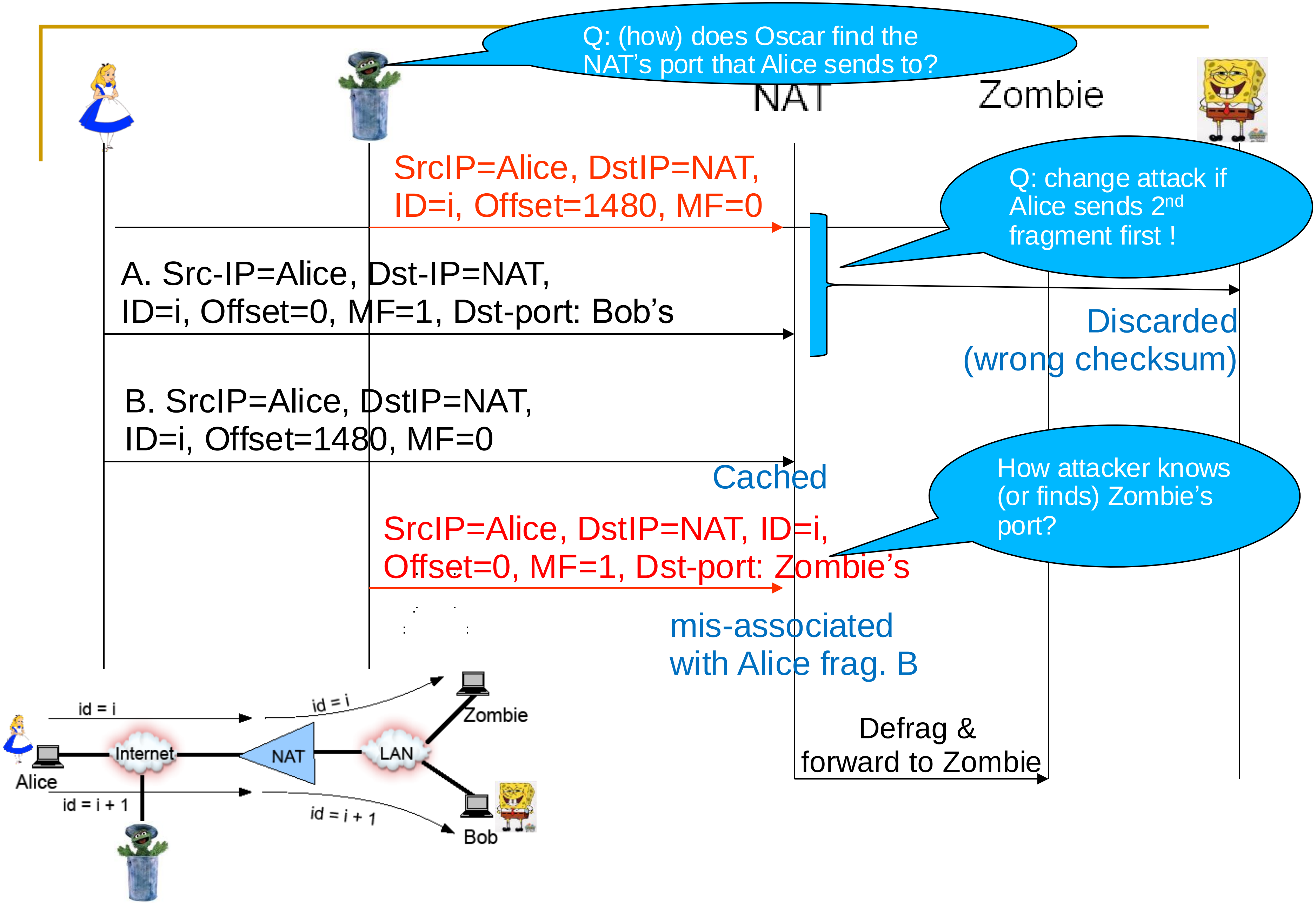


Fragmentation attacks on IP

Attacker models → Security goals	MitM	Eavesdropper	Off-Path
Confidentiality and privacy	None (without IPsec)	None (without IPsec)	1. 2nd Frag interception attack
Integrity and authentication	None (without Ipsec)	Trivial	2. 2nd Frag spoofing attack
(a[illegible]			[illegible]

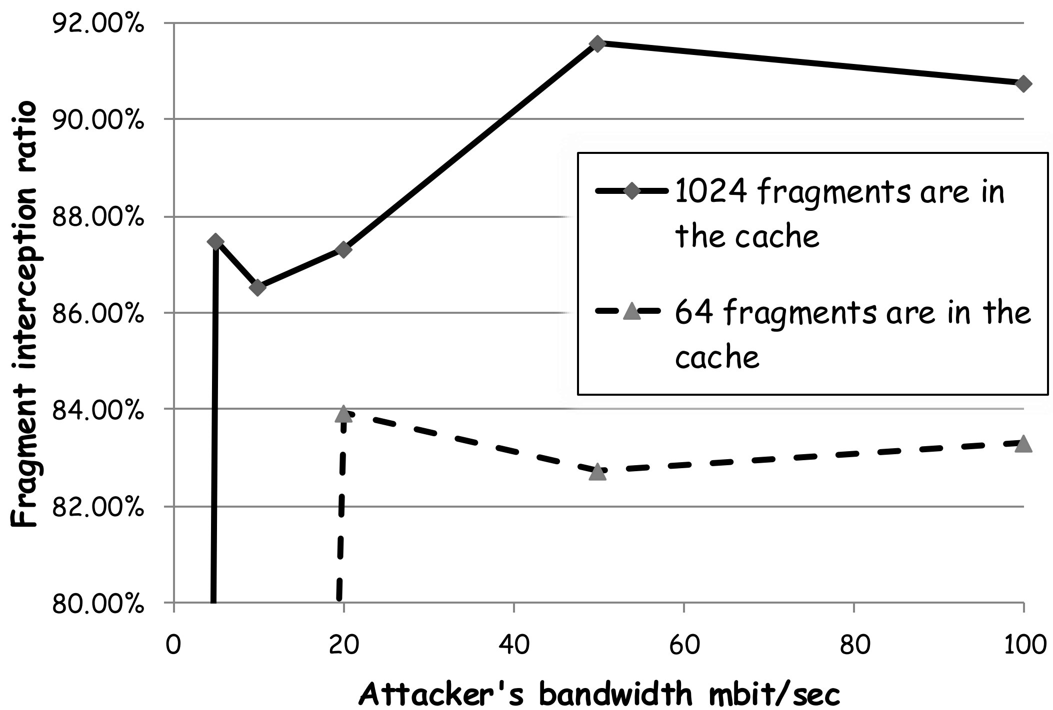
- Next: 2nd-frag **intercept / spoof attacks**
- Idea: replace 1st or 2nd fragment, respectively
- Both: assume IP-ID known (and fragmentation)

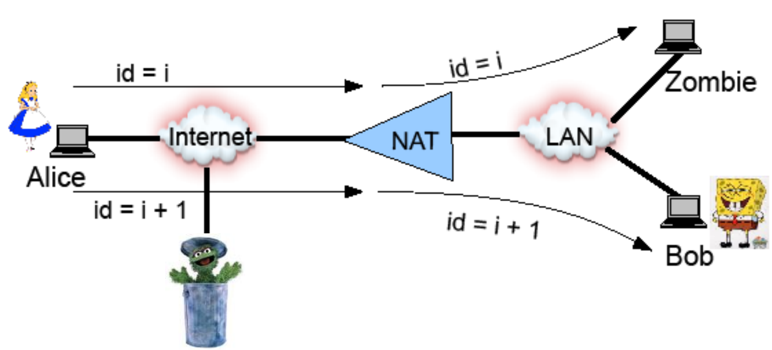
Off-path 2nd fragment intercepting



2nd Fragment Interception: Results

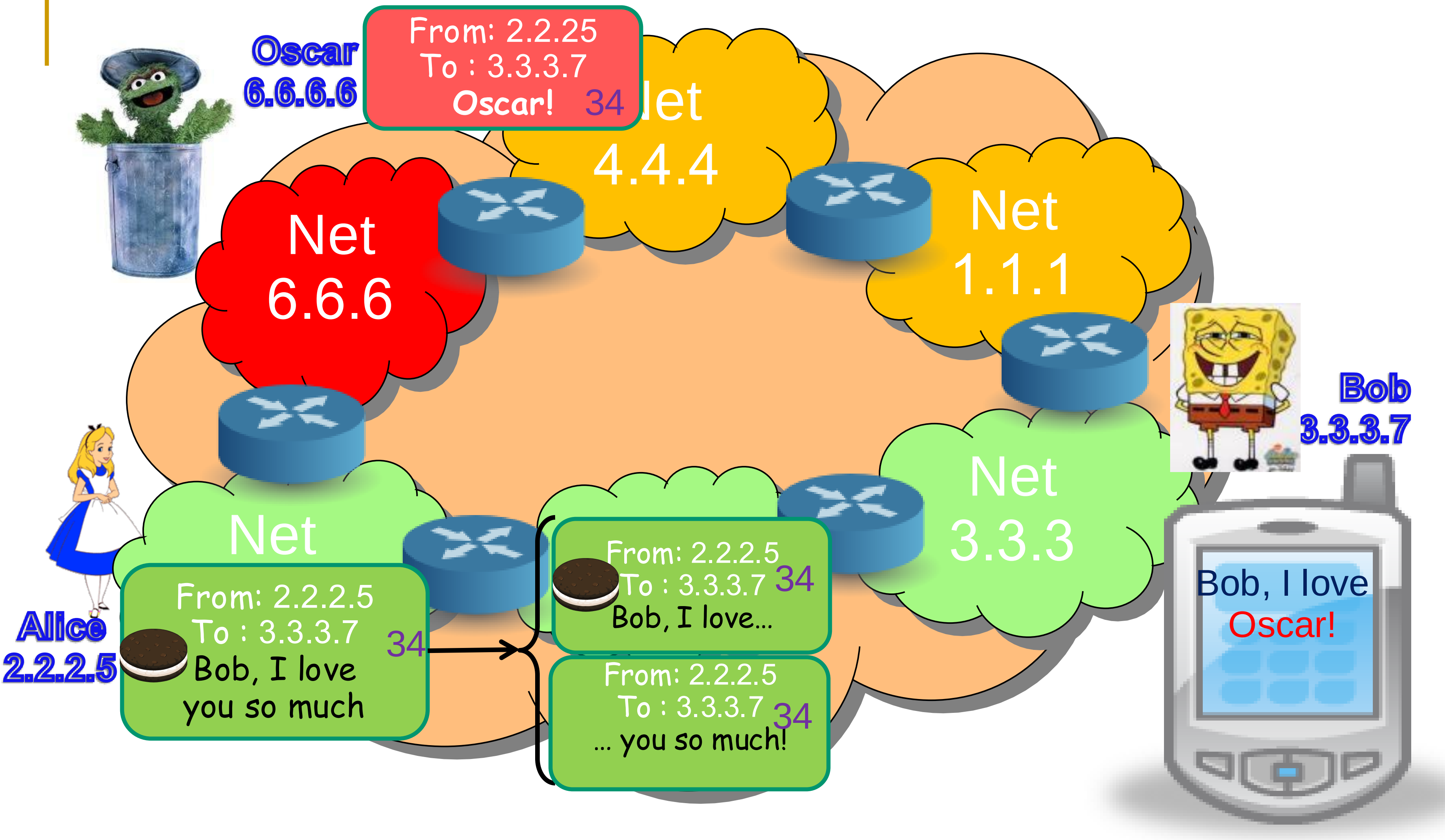
- Results for IP tables based NAT



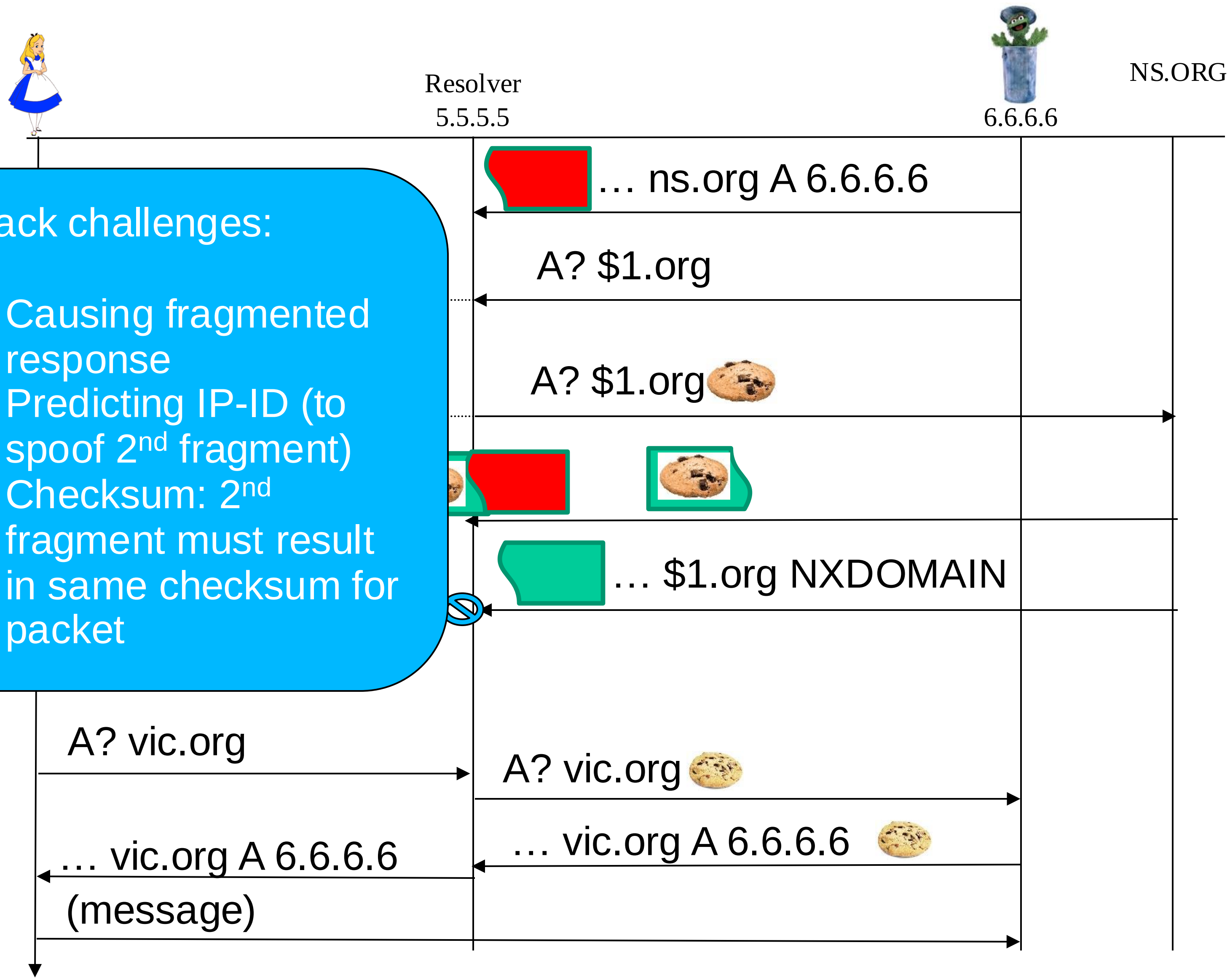


Off-path 2nd fragment spoofing attack

(for *IP/DNS/TCP header cookie grabbing*)

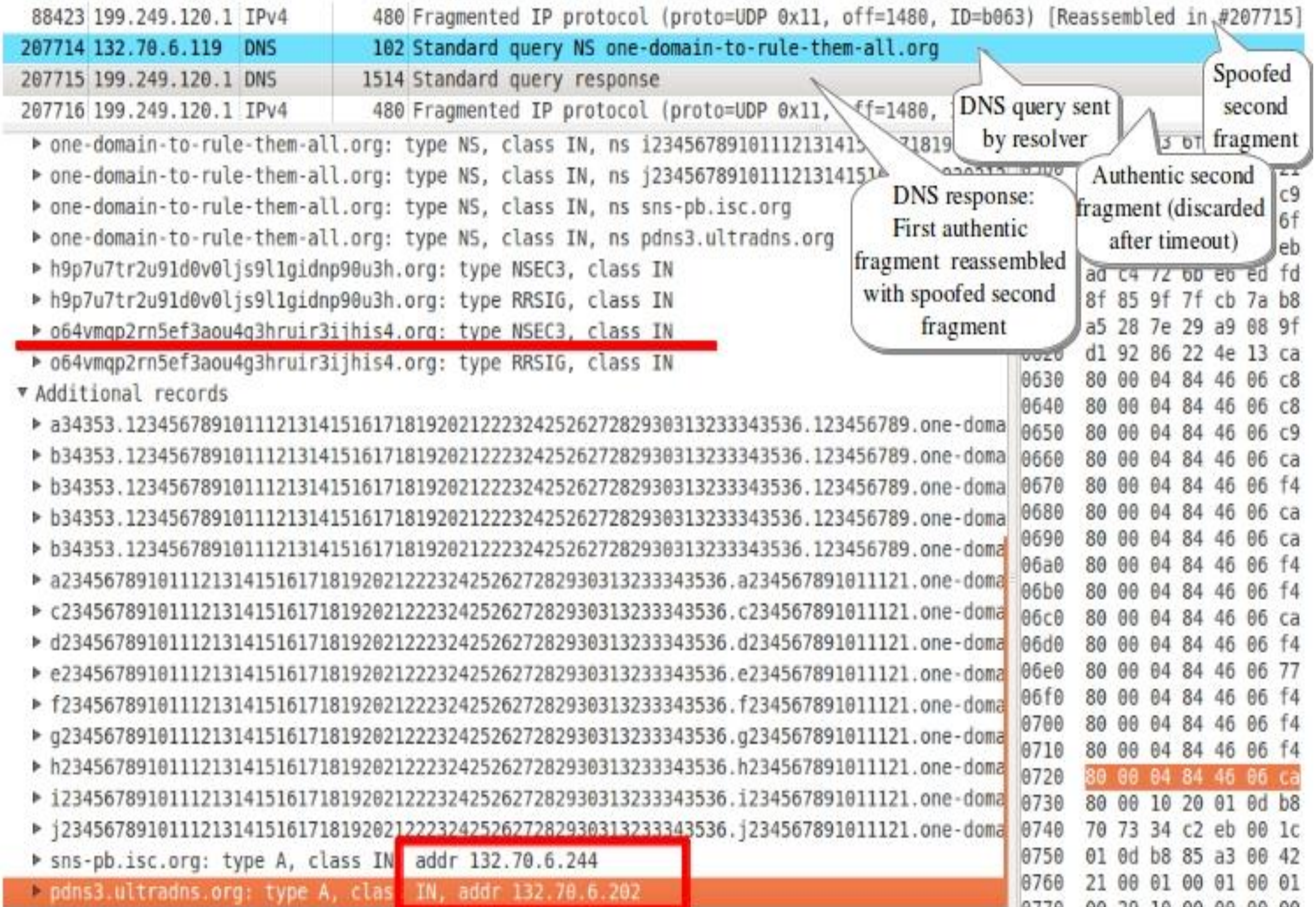


DNS Poisoning by spoofed 2nd fragment



Attack challenges:

1. Causing fragmented response
2. Predicting IP-ID (to spoof 2^{nd} fragment)
3. Checksum: 2^{nd} fragment must result in same checksum for packet

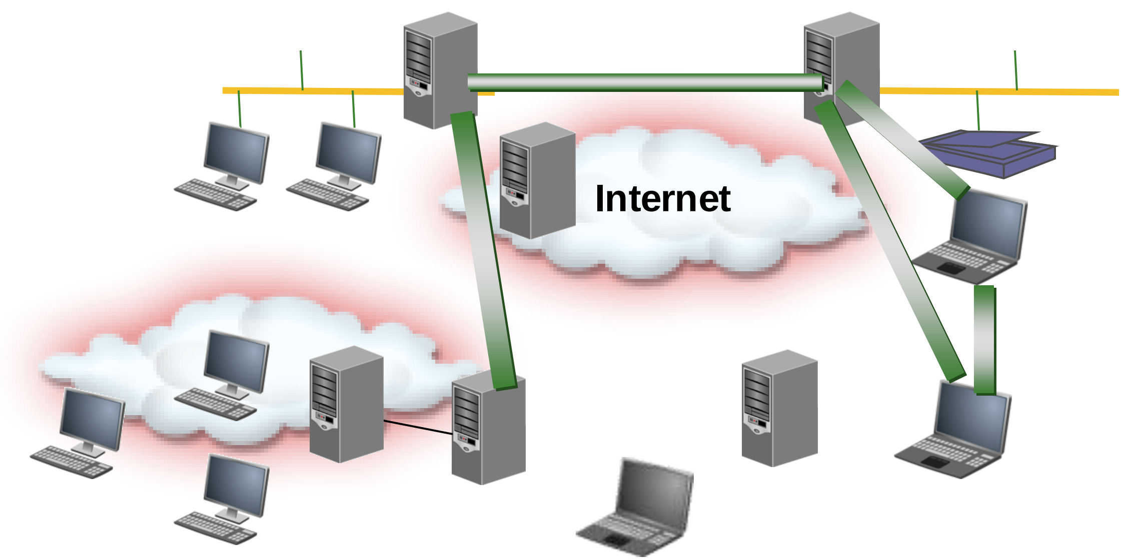

88423 199.249.120.1 IPv4 480 Fragmented IP protocol (proto=UDP 0x11, off=1480, ID=b063) [Reassembled in #207715]
207714 132.70.6.119 DNS 102 Standard query NS one-domain-to-rule-them-all.org
207715 199.249.120.1 DNS 1514 Standard query response
207716 199.249.120.1 IPv4 480 Fragmented IP protocol (proto=UDP 0x11,
one-domain-to-rule-them-all.org: type NS, class IN, ns
one-domain-to-rule-them-all.org: type NS, class IN, ns
one-domain-to-rule-them-all.org: type NS, class IN, ns sns-pb.isc.org
one-domain-to-rule-them-all.org: type NS, class IN, ns pdns3.ultradns.org
h9p7u7tr2u91d0v0ljs9l1gidnp90u3h.org: type NSEC3, class IN
h9p7u7tr2u91d0v0ljs9l1gidnp90u3h.org: type RRSIG, class IN
o64vmqp2rn5ef3aou4g3hruir3ijhis4.org: type NSEC3, class IN
o64vmqp2rn5ef3aou4g3hruir3ijhis4.org: type RRSIG, class IN
Additional records
sns-pb.isc.org: type A, class IN addr 132.70.6.244
pdns3.ultradns.org: type A, class IN, addr 132.70.6.202
DNS query sent by resolver
Spoofed second fragment
Authentic second fragment (discarded after timeout)
DNS response: First authentic fragment reassembled with spoofed second fragment

TCP/IP Security

- **Internet Protocol (IP) Security**
  - Fragmentation attacks
  - **IPsec**
- Secure Transport
  - TCP injections and other attacks
  - Quic
- Conclusion

Secure Virtual Private Networks

- Private Network: owned by single organization
- Secure VPN: secure networking over insecure Net
  - Prevent eavesdropping, spoofing, injecting, modifying
  - MitM attacker
  - Main (standard) tool: **IP-Sec**



Fragmentation attacks on IP

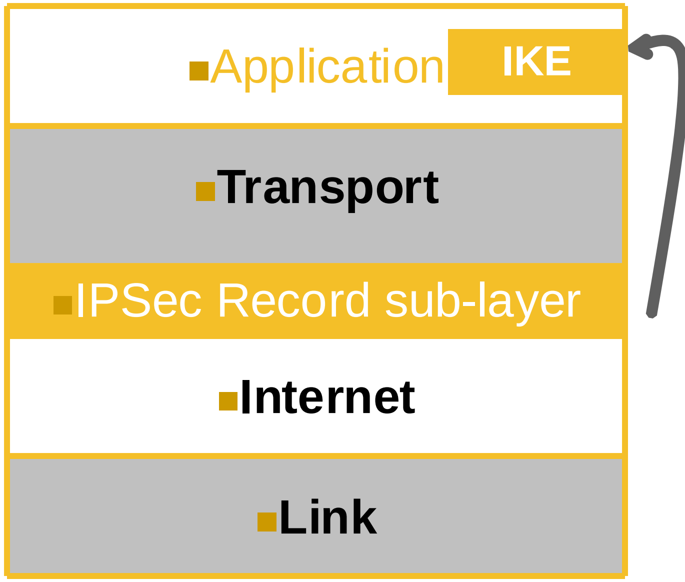
Attacker models → Security goals	MitM	Eavesdropper	Off-Path
Confidentiality and privacy	None (without IPsec)	None (without IPsec)	**2nd Frag interception attack**
Integrity and authentication	None (without Ipsec)	Trivial	**2nd Frag spoofing attack**
Availability (and efficiency)	None	Trivial	**Frag-based packet drop and overhead attacks**

IP Security: Impact of IPsec

Attacker models → Security goals	MitM	Eavesdropper	Off-Path
Confidentiality and privacy	**Fixed by IPsec** **ICMP-redirect and feedback attacks on encryption-only ESP**	None (without IPsec)	Expected
Integrity and authentication	**Fixed by IPsec** **Modification attack on encryption-only ESP**	Trivial	Expected: spoofing, but no modification
Availability (and efficiency)	None Stealthy, efficient attacks in spite of IPsec	Trivial	**Frag-based packet drop and overhead attack (even for TCP due to tunnel?)**

IP-Sec Layers

- Two separate layers
- IKE – Internet Key Exchange
  - Establish shared key (application layer)
- IP-Sec Record Sub-Layer: traffic encapsulation & protection
  - **ESP protocol – Encapsulating Security Payload**
  - **AH protocol – Authentication Header** (only authentication, no encapsulation)
  - Signal to IKE when detecting traffic that requires IP-Sec but without established IP-Sec connection

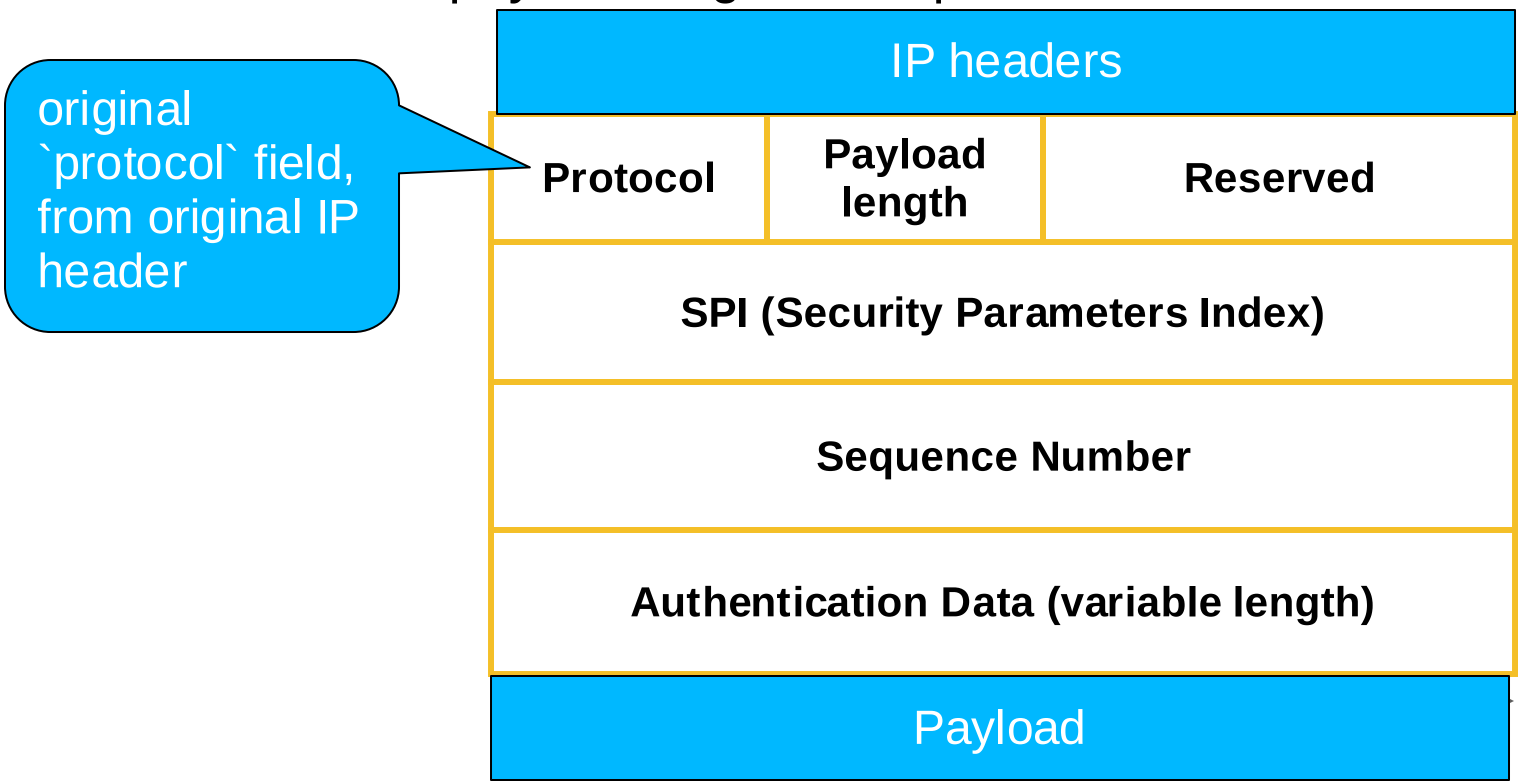


IP-Sec Record Sub-Layer

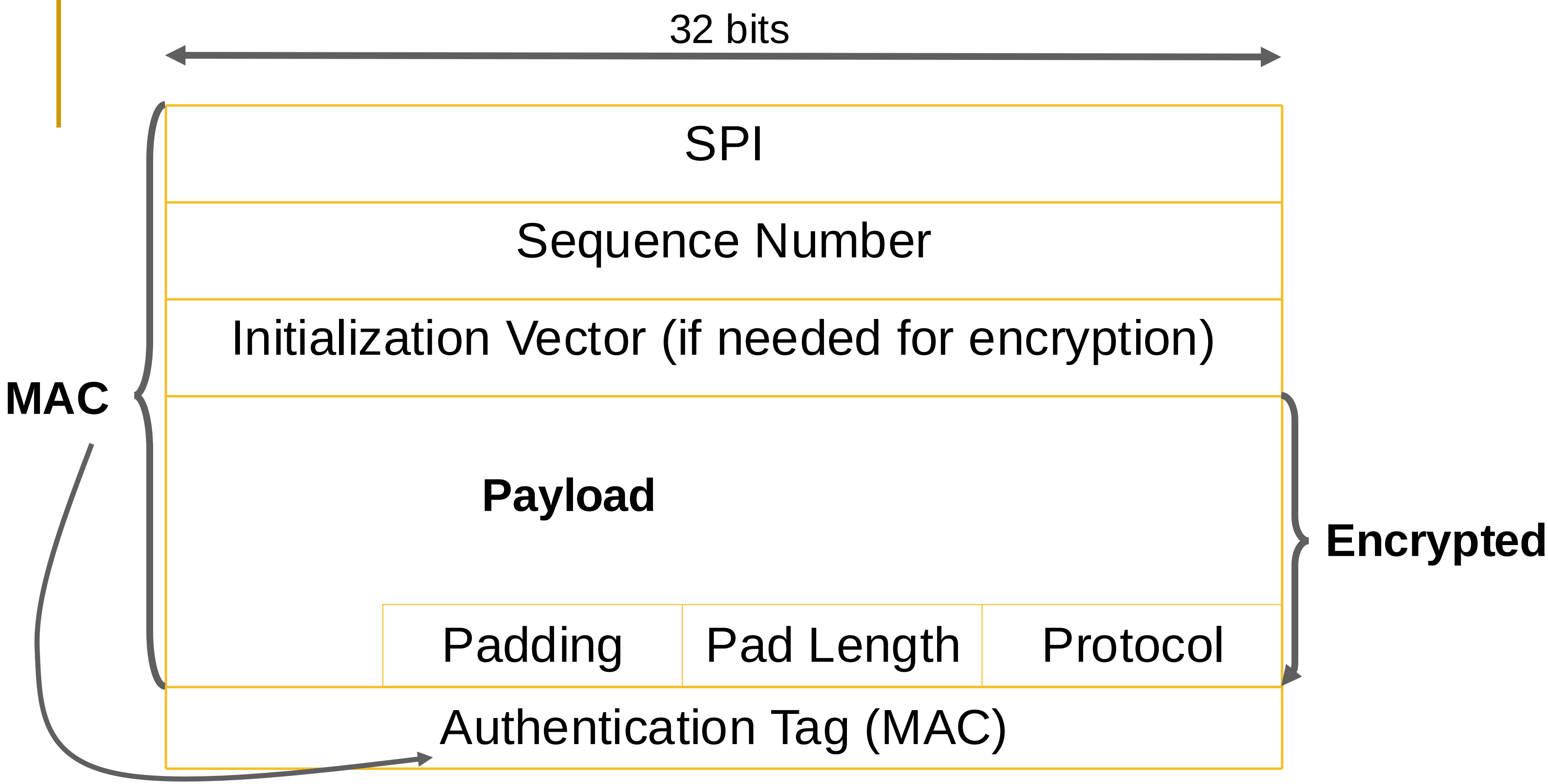
- Consists of two header protocols
  - Sent as protocols over IP
  - **ESP: Encapsulating Security Payload** (IP protocol 50): supports encryption and/or authentication (MAC)
  - **AH: Authentication Header** (IP protocol ID 51): only MAC
- Both AH and ESP:
  - Add sequence numbers to packets (to prevent duplication)
  - `Synchronized` keys, algorithms and counters
    - **Security Association (SA)**
    - Identify SA for packet via its **Security Parameter Index (SPI)**

AH - Authentication Header [RFC4302]

- Authenticates entire IP packet: IP header, AH header, and payload
  - Except router-modified fields in IP-header, e.g. TTL (hop count)
  - UDP/TCP payload begins with ports, contains checksum



ESP – Encapsulating Security Payload



Encryption-Only ESP can be Vulnerable!

- Authentication is optional in ESP
  - Advised, but not always done
- Q: does the (encrypted) checksum ensure integrity?
- No! ESP without authentication has no integrity
  - How can attacker change without the key?
  - Easier to see assuming stream-cipher encryption (e.g. OTP)…

Encryption-Only ESP can be Vulnerable!

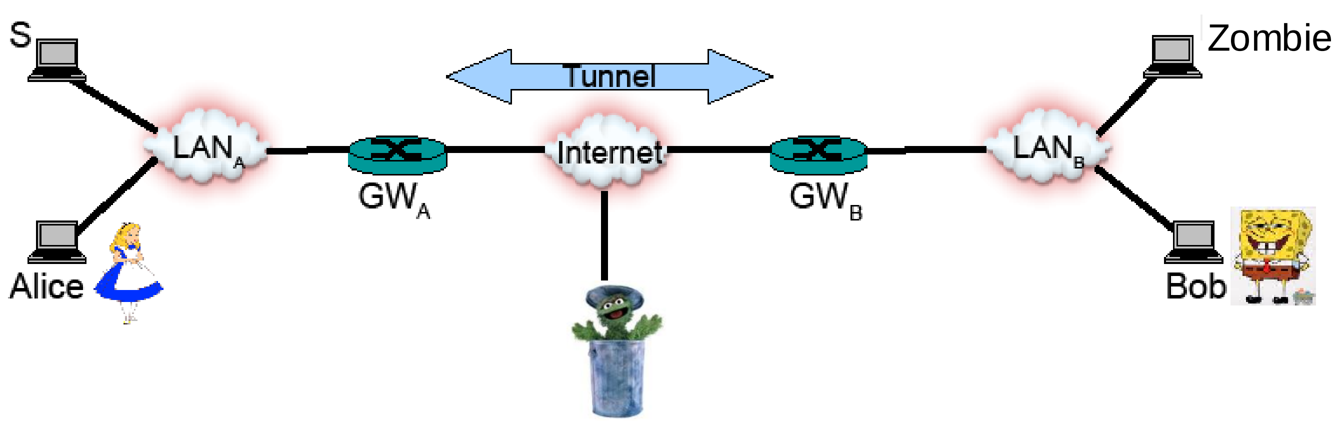
- Authentication is optional in ESP
  - Advised, but not always done (often not even default)
- ESP without authentication may not provide integrity
  - In spite of TCP/UDP checksum, etc.
  - Easier to see for stream-cipher encryption (OTP, OFB)
    - <u>Some</u> changes are easy also for CBC, CTR, CFB
- Confidentiality is vulnerable too!
  - Changed ciphertext attacks, exploiting feedback (side channel)
    - [Bellovin 96] TCP silently discards wrong checksum (no ack)
    - [Vaudenay 02, Paterson 06] response to incorrect padding
  - Modify header to ICMP echo – redirect to attacker – expose contents!
  - ➔ IPsec should <u>always also authenticate</u>
  - Attacks may depend on <u>mode of operation</u> of cipher, <u>IPSec</u>

IP-Sec: Transport vs. Tunnel Modes

- Transport Mode (end to end only)
  - Change protocol field to AH (51) or ESP (50)
    - Protocol field kept in (& restored from) IPsec header
  - Except protocol field: use existing IP header
  - End-to-end – encapsulation by source host, decapsulation by destination host (receiver)
- Tunnel Mode (gateway to gateway or end to end)
  - Entire original IP packet (including header) is payload
  - IP-Sec puts a new IP header (protocol AH (51) / ESP (50))
  - Allows **Secure Virtual Private Network (VPN)**

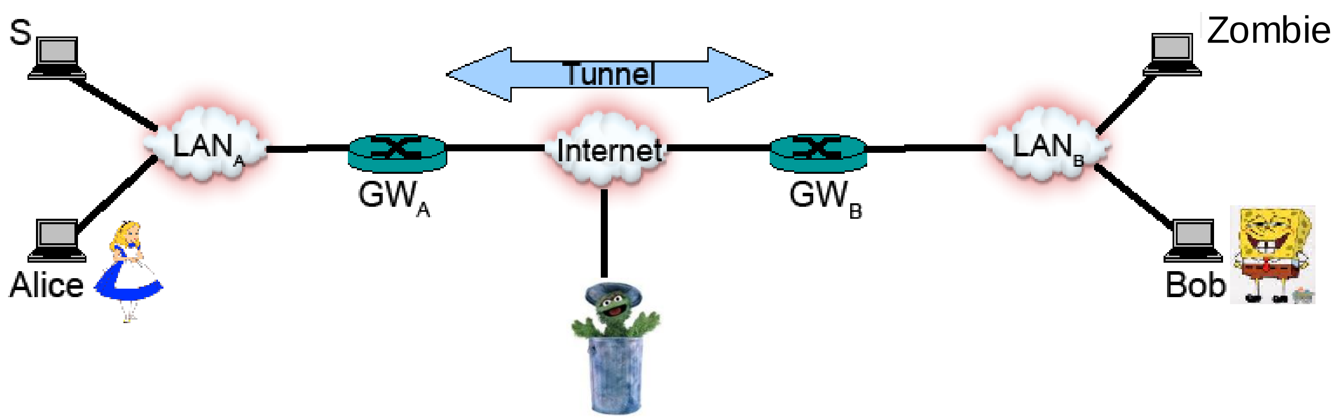
IP-ID Exposing Attacks on Tunnel

- Main difference from NAT scenario:
  - Packets `on the Internet' have a different IP header
    - Zombie, cannot see the `Internet IP-ID'
- Improved motivation: fragmentation is common in tunnels, due to extra header
  - Or caused by spoofed ICMP frag-required – possibly even for TCP, since embedded in ESP in tunnel



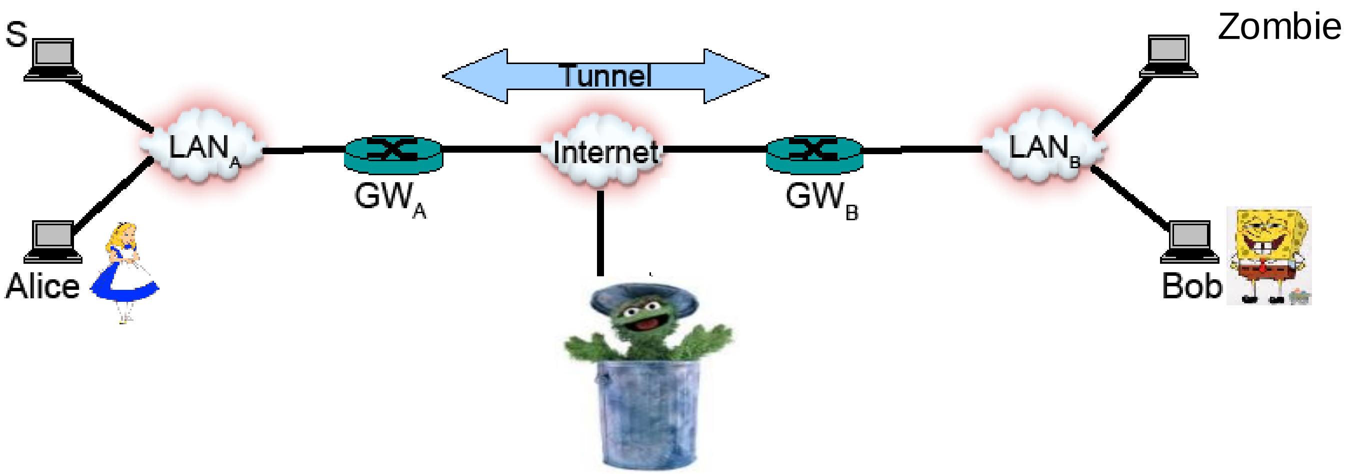
IP-ID Exposing Attack

- Use **packet loss as a side channel** to identify the current IP-ID between two hosts
  - Example: btw two gateways (end of `tunnel')
  - Similarly: btw (DNS) proxy and server
- Simplifying assumption: no benign traffic or packet loss

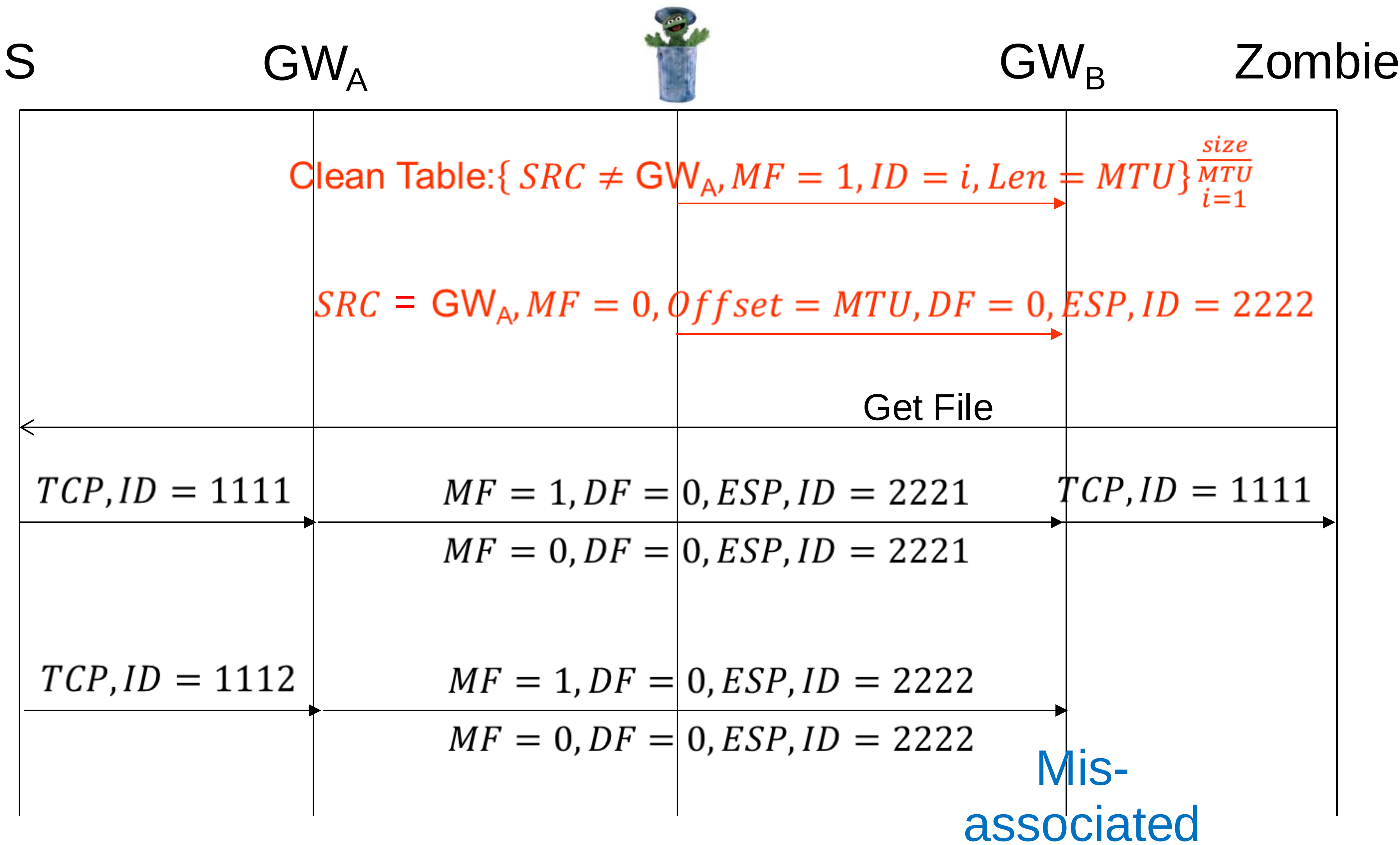


IP-ID Exposing: review

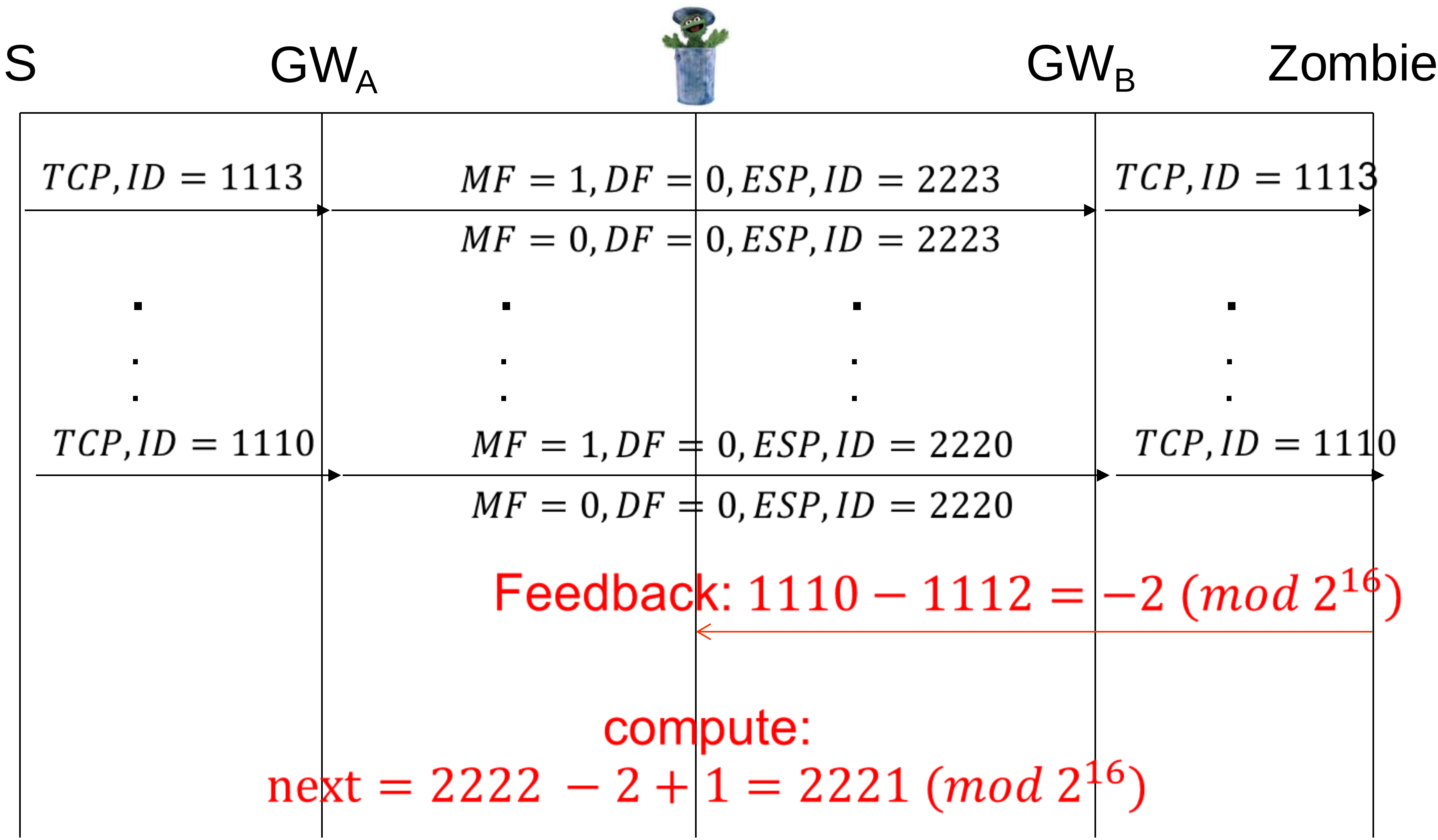
- Goal: find IP-ID of next pkt from GW_A to GW_B
  - Most efficiently (minimal # of packets)
- Assume:
  - Zombie can 'get' packets from S (e.g., files)
  - Packets encapsulated btw gateways, and fragmented
  - Incremental IP-ID
  - No other (benign) traffic, no packet loss



ID Exposing Attack: tunnel scenario

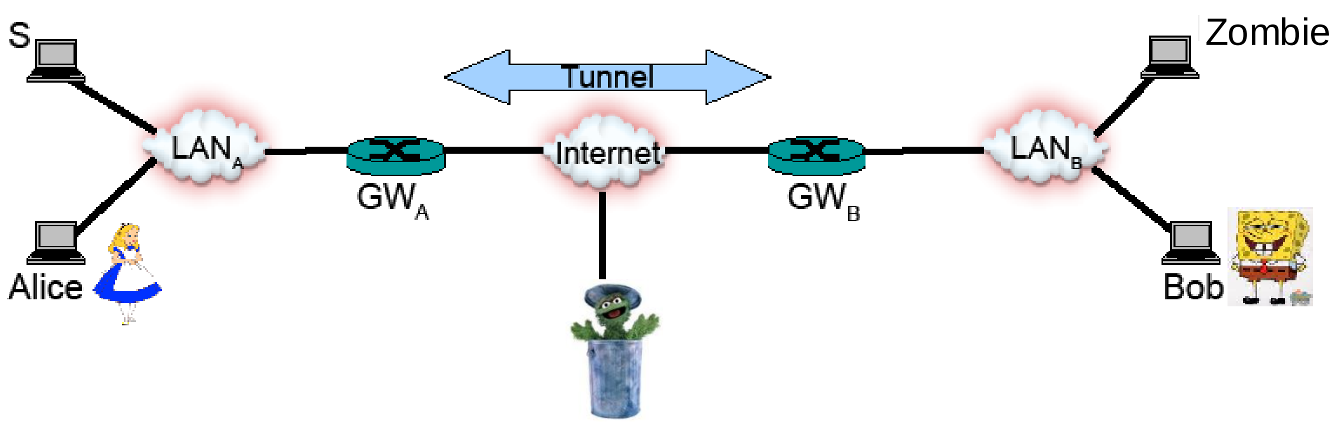


IP-ID Exposing Attack



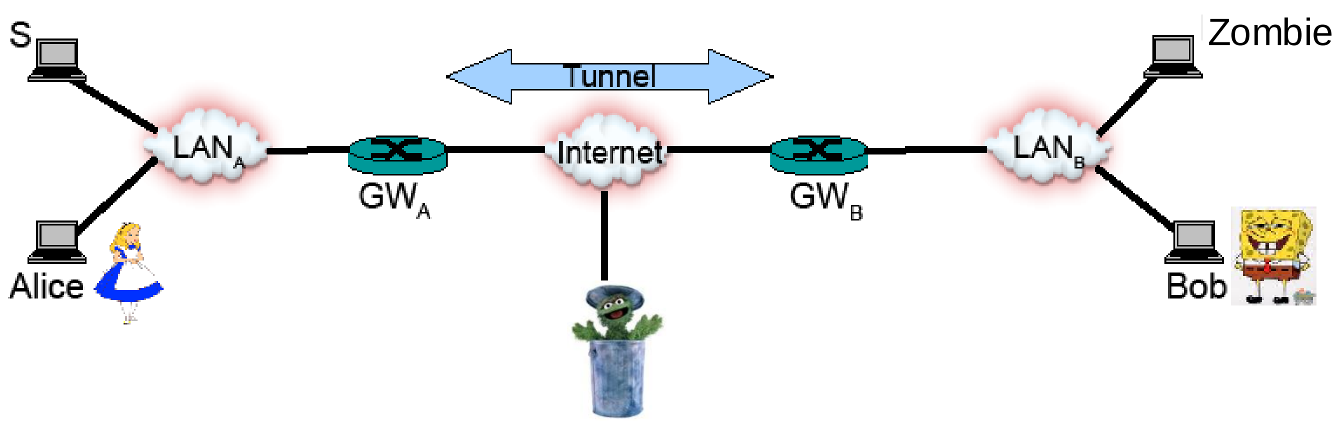
ID Exposing Attack - Meet in the Middle

- But… if n is the number of possible identifiers, this attack requires to send O(n) packets.
  - 2^{16} for IPv4, for 2^{32} IPv6
- More efficient: use Battleship optimization



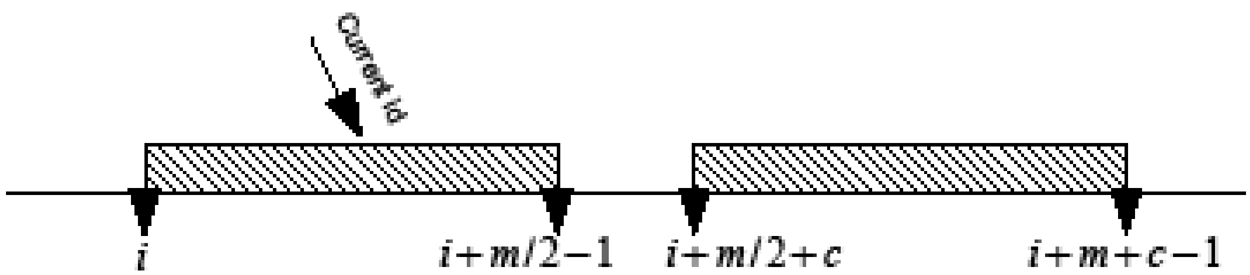
Exploit: Continual Deny & Expose

- Off-path attacker has the current IP-ID
  - Goal: deny fragmented traffic
- Difficulty: maintain synchronization with current IP-ID
  - Incremented for every packet (regardless of arrival/loss)



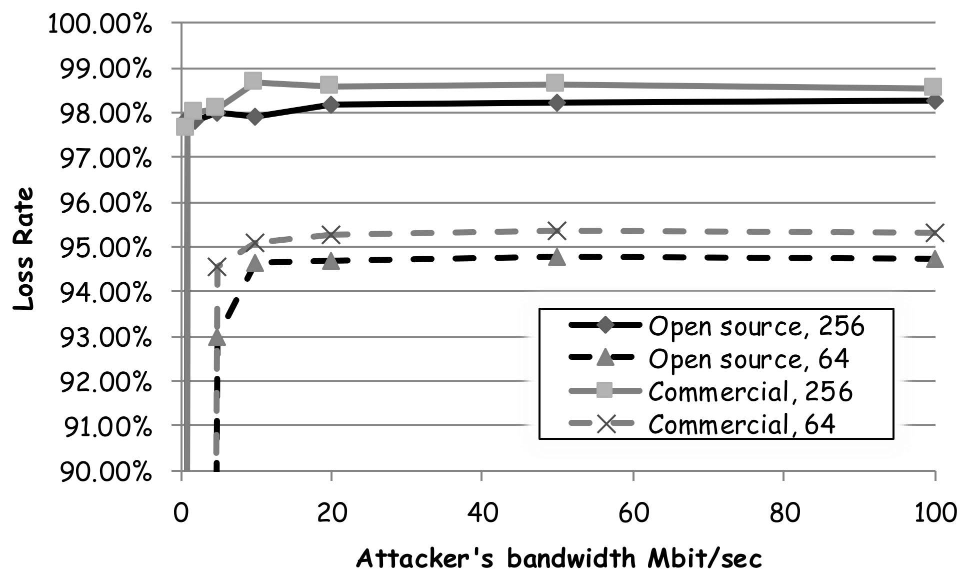
Continual Deny & Expose

- Idea: use zombie to `monitor' IP-ID progress
  - Send two sequences of spoofed fragments with consecutive IDs
  - Small `gap' of unsent IDs between them
  - Zombie makes a periodic request for data
  - Response arrives → ID within the gap
  - Send the next sequence
- Causing over 95% loss rate



Continual Deny & Expose - Results

- Success depends on the number of forged fragment attacker can `cache in'
  - Usually 64 or no limitation (except cache size, 6500+)



TCP/IP Security

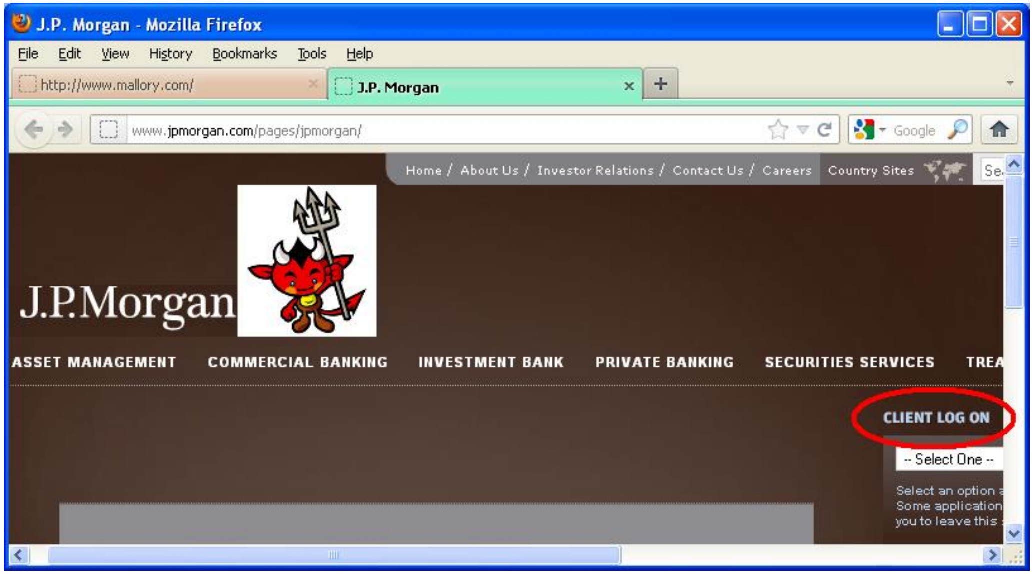
- Internet Protocol (IP) Security
  - Fragmentation attacks
  - IPsec
- **Secure Transport**
  - TCP injections and other attacks
  - Quic
- Conclusion

Off-Path Attacks on TCP

- Denial/degradation of Service (DoS)
  - SYN-flood: DoS server (crash / no new connections)
  - BW/performance degradation attacks
  - TCP Amplification attacks
  - Foil connection from client (by IP or also port)
  - Break ongoing TCP connection

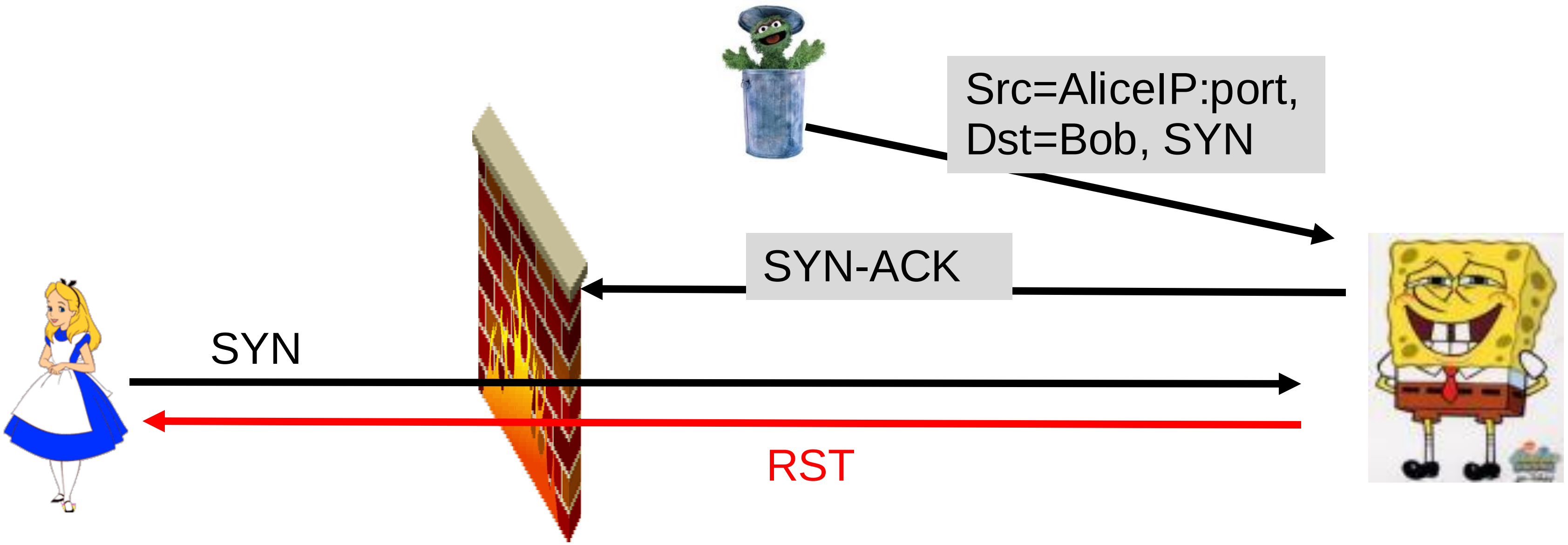
DoS lectures

- TCP Injection, e.g., of mal-script (puppet)



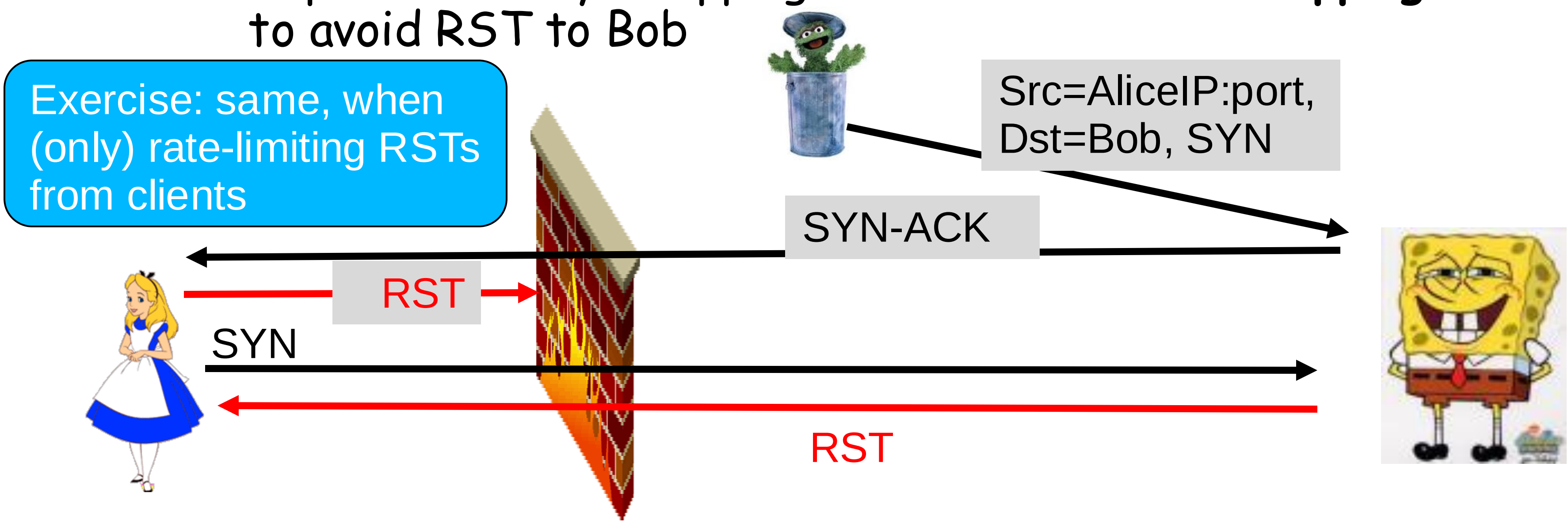
More Off-Path TCP DoS Attacks

- Foil <u>new</u> connection from given client
  - Method 1: send many SYNs ➔ server blocks client IP
    - Due to (some) rate limiting defenses (against SYN-Flood)
  - Method 2: Allocate client's 4-tuple (aka *4-tuple blocking*)
    - 4-tuple: (clientIP:**port**, serverIP:port)
    - Send SYN using 4-tuple; server allocates socket, rejects legit-client connection using same 4-tuple
    - Exploits silently-dropping FW/NAT to avoid RST to Bob



More Off-Path TCP DoS Attacks

- Foil <u>new</u> connection from given client
  - Method 1: send many SYNs ➔ server blocks client IP
    - Due to (some) rate limiting defenses (against SYN-Flood)
  - Method 2: Allocate client's 4-tuple (aka *4-tuple blocking*)
    - 4-tuple: (clientIP:**port**, serverIP:port)
    - Send SYN using 4-tuple; server allocates socket, rejects legit-client connection using same 4-tuple
    - Exploits silently-dropping FW/NAT **or RST-dropping FW** to avoid RST to Bob

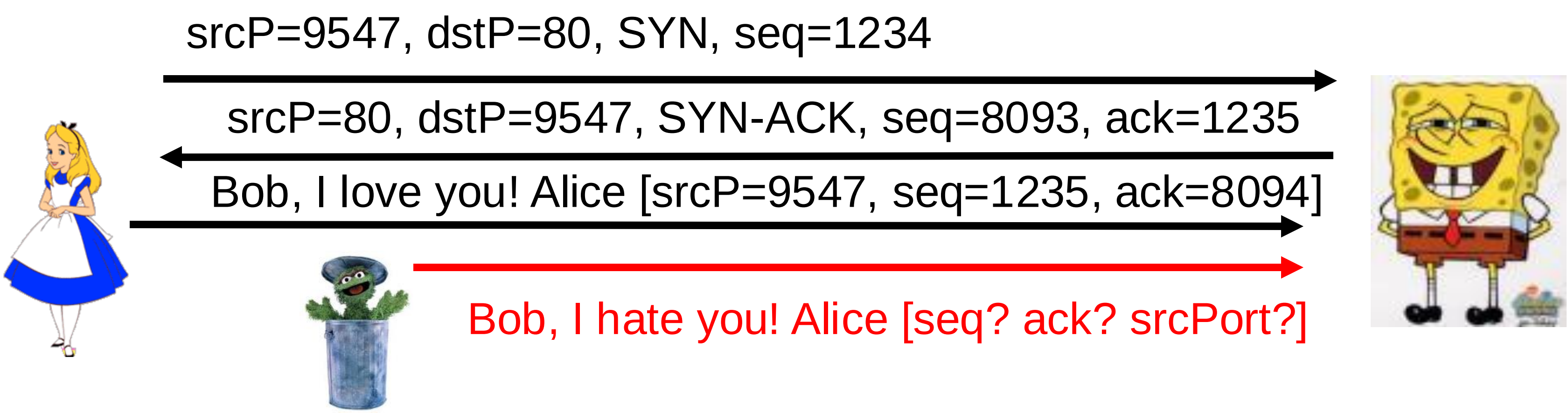


More Off-Path TCP DoS Attacks

- Foil <u>new</u> connection from given client
  - Method 1: send many SYNs ➔ server blocks client IP
    - Due to (some) rate limiting defenses (against SYN-Flood)
  - Method 2: Allocate client's 4-tuple (aka *4-tuple blocking*)
    - 4-tuple: (clientIP:**port**, serverIP:port)
    - Send SYN using 4-tuple; server allocates socket, rejects legit-client connection using same 4-tuple
    - IPs and server port are often known; Client port = ??
    - Note: may use 4-tuple-blocking also to <u>find</u> client's next port
      - For incrementing-source-port clients (most OSs)
- Break <u>ongoing</u> TCP connection
  - Method 1: spoofed RST
    - Must have correct IPs+ports <u>and</u> seq-num 'in window'
  - Method 2: Spoofed ICMP error message (same 4-tuple)

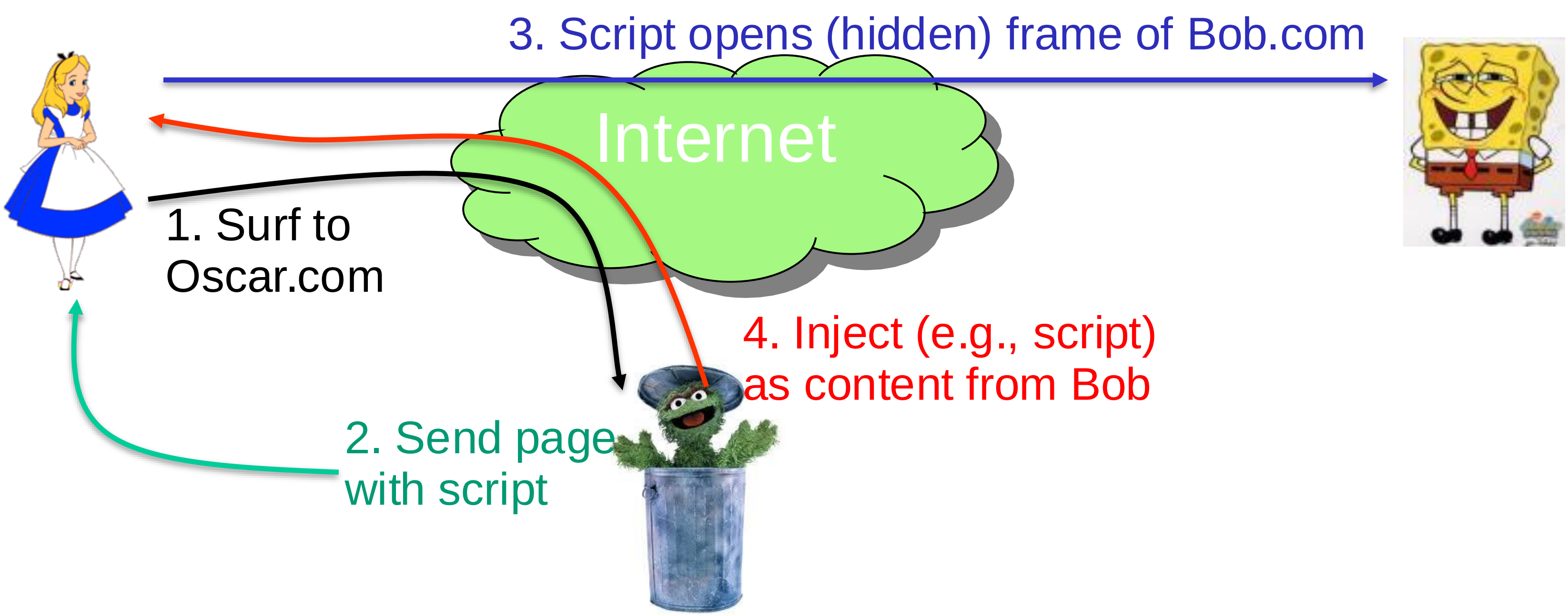
Off-path TCP inject challenges

- No explicit off-path defenses in TCP
- But… injection requires:
  - 4-tuple: (clientIP:**port**, serverIP:port)
    - IPs and server port are often known
  - And sequence/ack numbers
  - Initialized randomly (since the 1990s)



Off-Path TCP Injection: Scenario

1. Alice surfs to Oscar's site
2. Alice's browser runs Oscar's script (puppet)
3. Puppet sends HTTP requests to Bob
4. Oscar <u>injects response </u>(e.g. mal-script) into connection
   - Alice's browser assigns mal-script with origin of `Bob'
   - Cached ➔ long term attack
   - Use for phishing, credentials/info-theft, malware

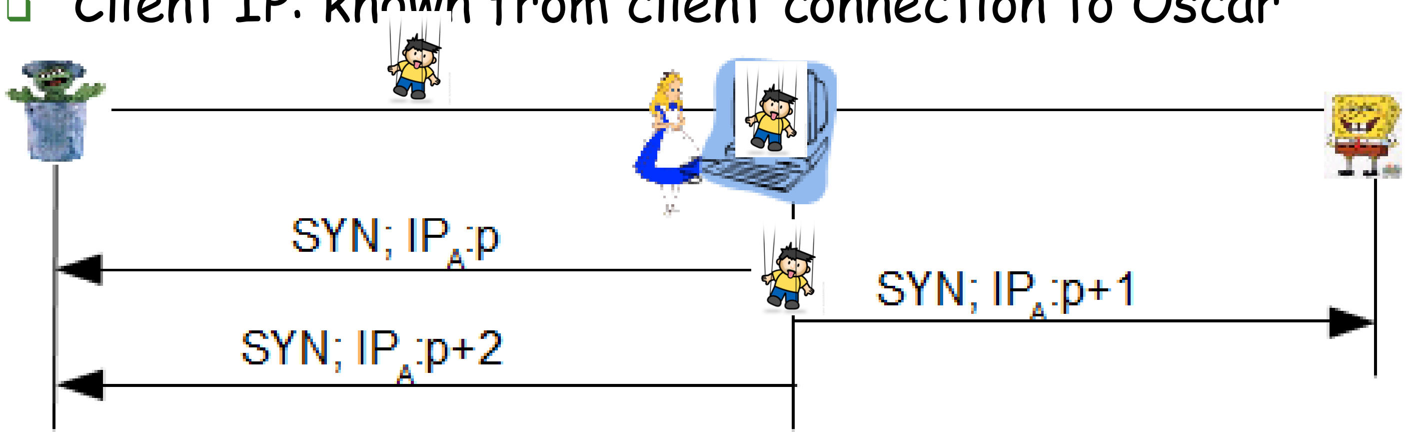


Off-path TCP Injection: Challenges

- First: identify \`victim-connection'
  - I.e., find 4-tuple: srcIP+port, dstIP+port
  - Ongoing or (easier) **initiated by attacker (puppet)**
  - **Only part we show** in this presentation
- Then: learn sequence numbers (seq+ack)
  - TCP discards packets with invalid seq #
  - We will not show in this presentation
- Finally: inject and exploit
  - Send (spoofed) data in correct (HTTP) context
  - Carefully manage TCP counters to avoid Ack storm from packets of legit destination (victim)
  - Browser assigns data the credentials of server (Bob)
    - Defeating \`Same Origin Policy'
  - Typically: send \`cross site script' (XSS)

Finding 4-tuple: <ServerIP:port, ClientIP:port>

- Trivial – for MitM or eavesdropper
- Easy for globally-seq client ports:
  - Puppet (script in browser) opens connection to Bob (server)
  - ServerIP:port known – or selected by puppet (attacker)
  - Client IP: known from client connection to Oscar



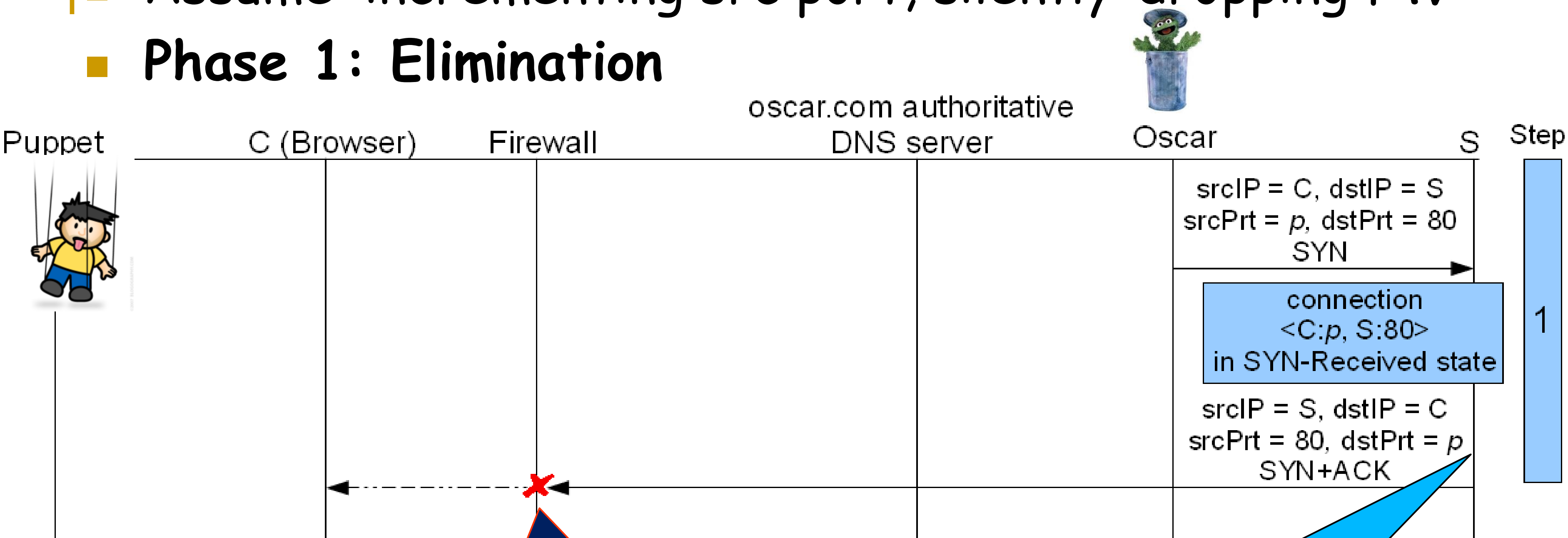
- Challenge: most clients use per destination incrementing (hash-based) ports

Per-dest incrementing (Hash-Based) Ports

- Algorithm:
  - On first connection to S: port = hash(S, key)
  - Increment port for new connections with S
- A per-destination `randomized counter'
- Common
  - Linux (since 2006), Android, iOS
  - DNS resolvers (since 2008)
  - …
- Find next port by *Eliminate-then-Test attack*:
  - Find next port to be used by per-destination incrementing client
  - By using *4-tuple blocking* to block specific ports, then detect if current port is blocked
  - Assume: silently-dropping of unsolicited SYN-ACK packets

Eliminate then Test Attack

- Goal: predict next C→S src port
- Assume: incrementing src port, silently-dropping FW
- **Phase 1: Elimination**

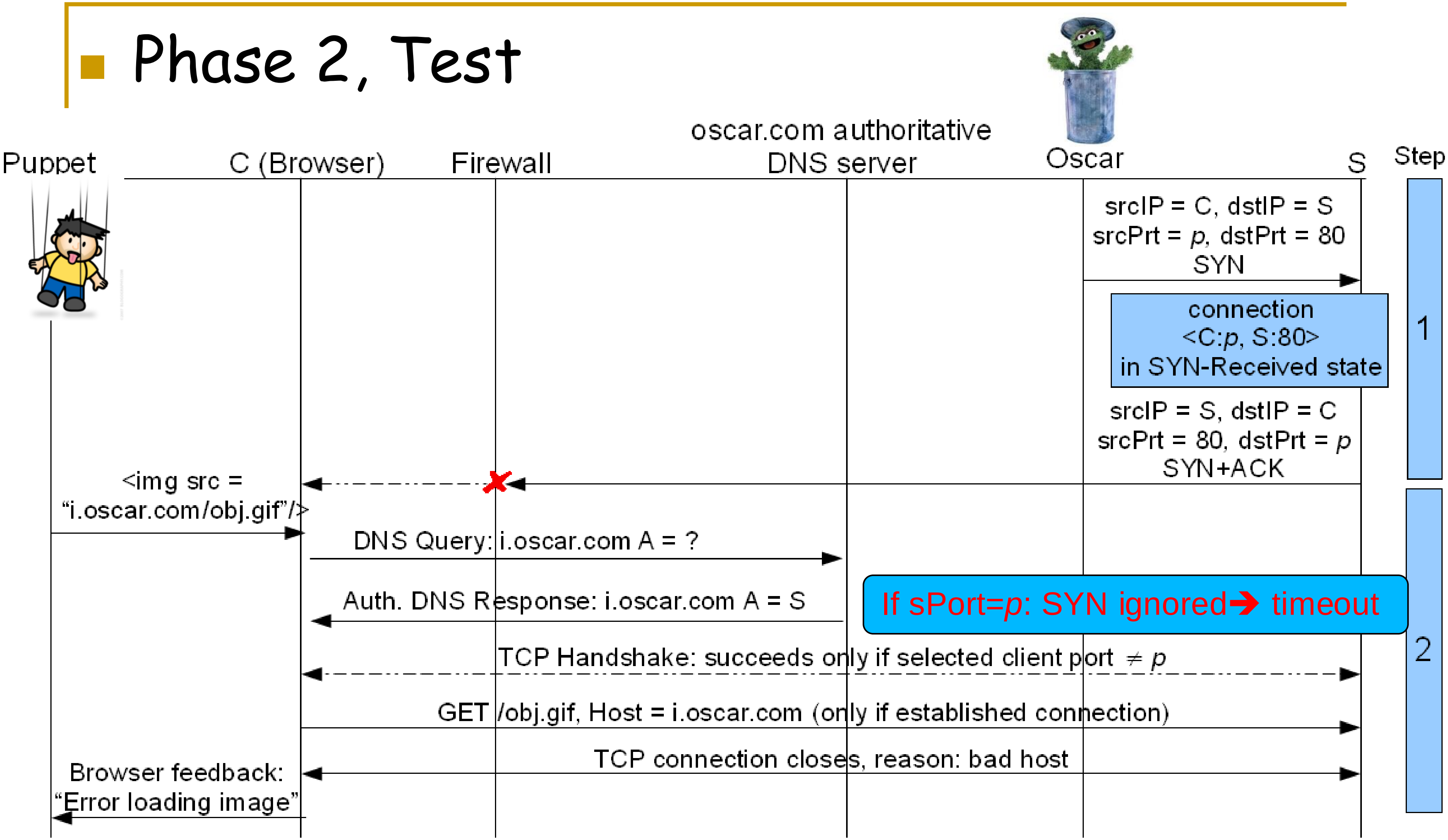


silently-dropping of unsolicited SYN-ACK packets

Move to SYN-RCVD state; ignore any pkt with incorrect seq# (incl. SYN)…

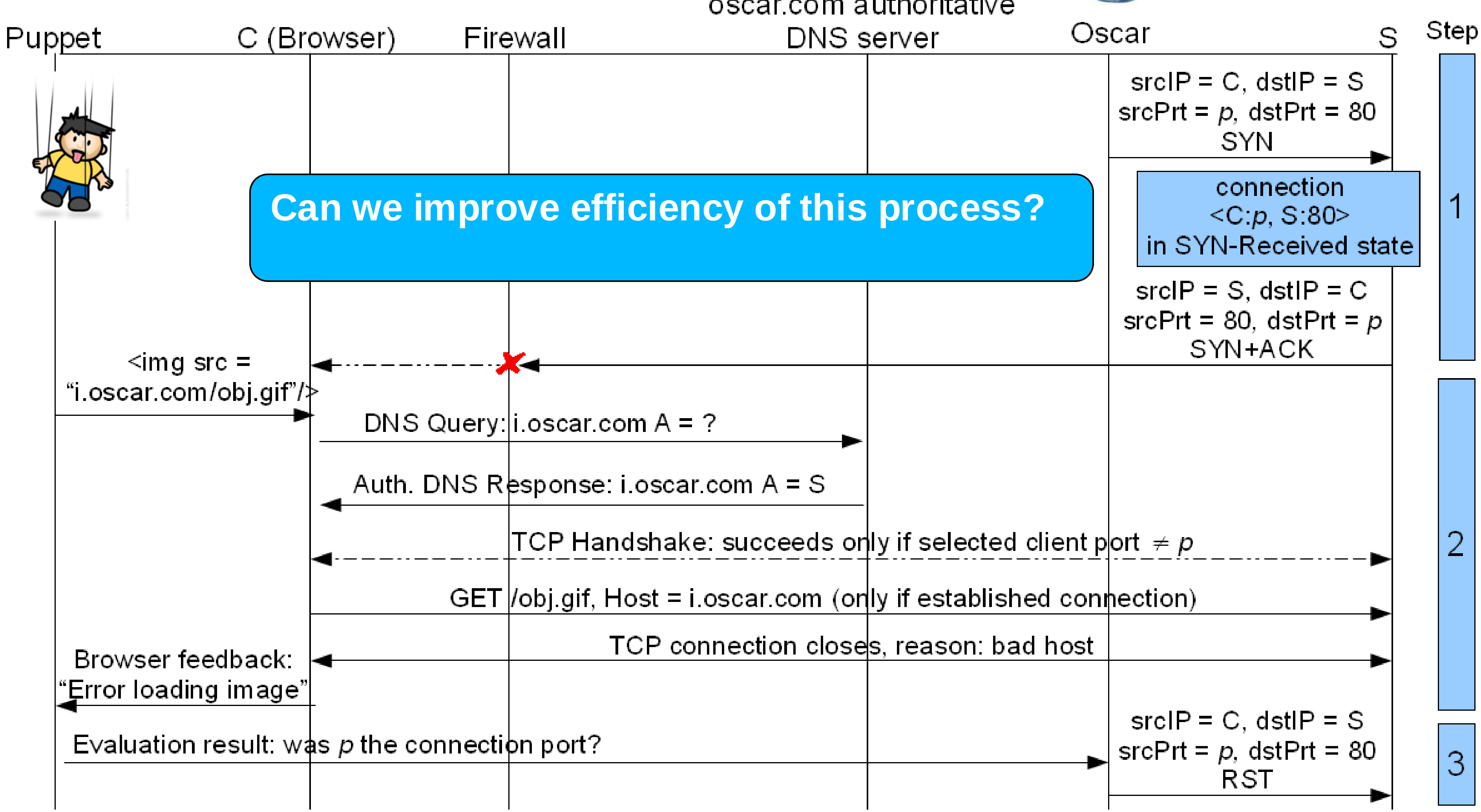
Eliminate then Test

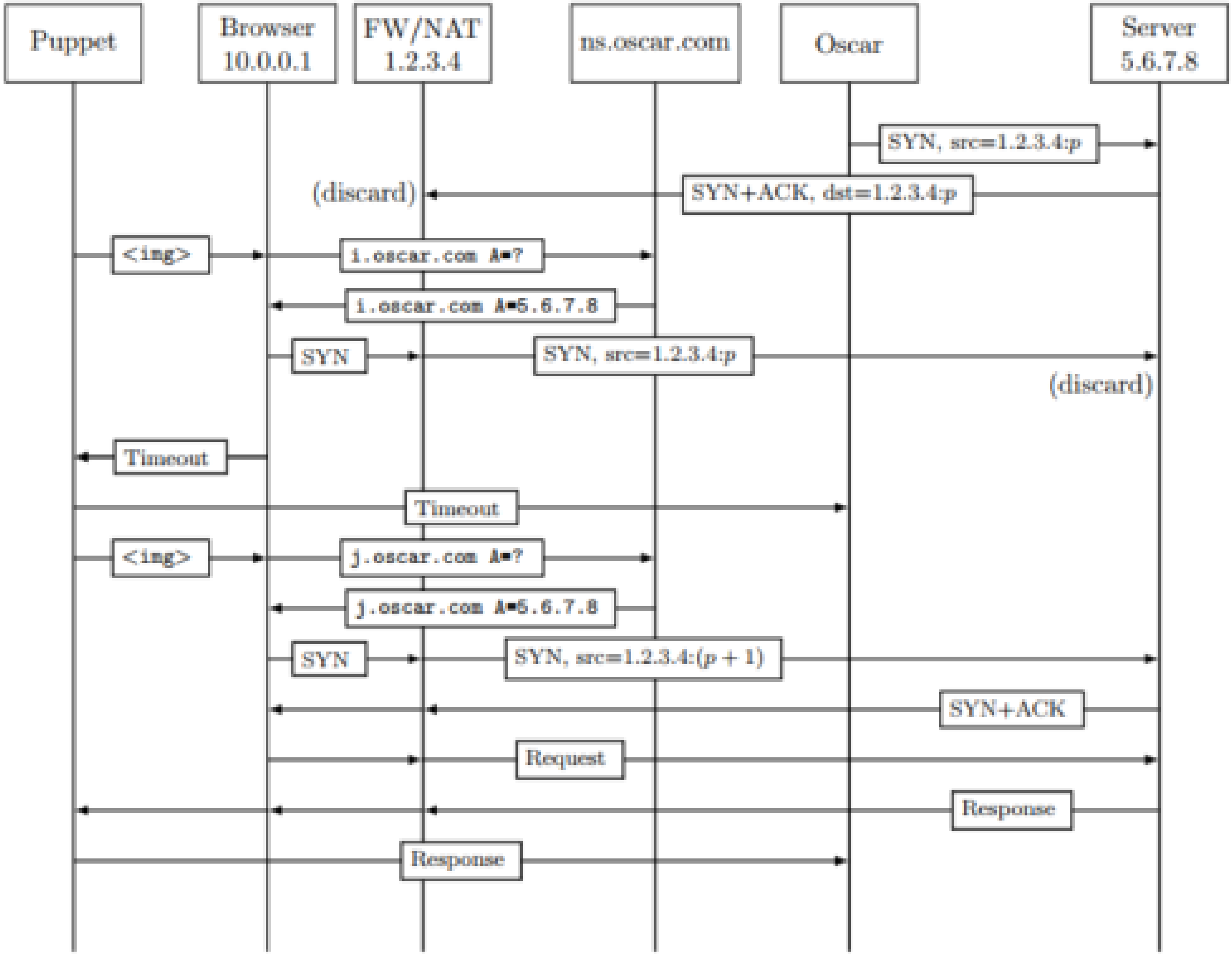
- Phase 2, Test



Eliminate then Test

- Also Phase 3, Cleanup…




Puppet
Browser
10.0.0.1
FW/NAT
1.2.3.4
ns.oscar.com
Oscar
Server
5.6.7.8
SYN, src=1.2.3.4:p
(discard)
SYN+ACK, dst=1.2.3.4:p
<img>
i.oscar.com A=?
i.oscar.com A=5.6.7.8
SYN
SYN, src=1.2.3.4:p
(discard)
Timeout
Timeout
<img>
j.oscar.com A=?
j.oscar.com A=5.6.7.8
SYN
SYN, src=1.2.3.4:(p + 1)
SYN+ACK
Request
Response
Response

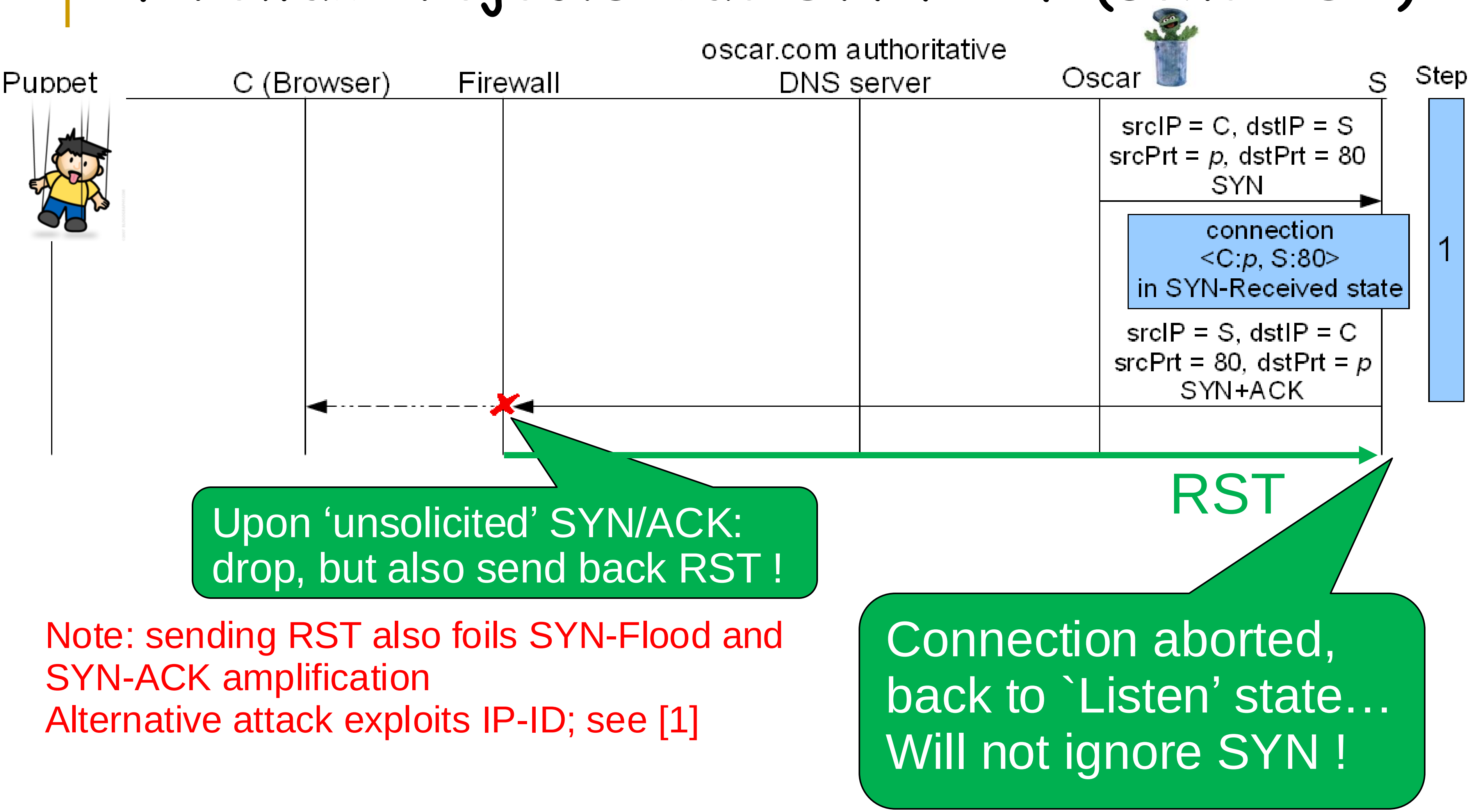
Finding client port: battleship optimization

- `Eliminate and test' can test all ports
  - Works, but slow: 65536 ports!
- Optimize with battleship optimization:
  - Eliminate ports 256*i, for i=1…256
  - Open 256 connections to server (as before)
  - … complete details – exercise!
- Over 90% success rate!

- Question: how could FW foil the attack?

Firewall Could Foils Elimination Phase

- Firewall `rejects' bad SYN/ACK (send RST)



Note: sending RST also foils SYN-Flood and SYN-ACK amplification
Alternative attack exploits IP-ID; see [1]

Finding Sequence Numbers

- Injection requires server, client seq#
- To inject data to client:
  - Server seq# should be `in window'
    - If not exact: cached
  - Client seq#: depends on OS
    - Exact, in window, or arbitrary
- Different methods work (or fail) for different systems…
  - Cat and mouse game
  - See papers… not in course

TCP/IP Security

- Internet Protocol (IP) Security
  - Fragmentation attacks
  - IPsec
- **Secure Transport**
  - TCP injections and other attacks
  - **Quick introduction to Quic and its security**
- Conclusion

Quick UDP Internet Connections

- QUIC: (Google's) new transport protocol
  - TCP-features: connections, reliability, controls…
  - Over UDP: clean-slate, user-space
    - Some UDP concerns (fragmentation, identify client)
  - Significantly (QUICkly) deployed: esp. Google ☺
  - Improve latency, perceived performance:
    - Multiplexed requests
    - Prioritized requests
    - Compression
  - Many of the functions/mechanisms of HTTP2
- Security?

QUIC Security

- Built-in secure connections
  - Combines TCP and TLS handshake and features
  - Always encrypted, authenticated
- Reuse state across connections:
  - Source-address token: detect spoofing
  - Cookie: server's public DH values
    - Allows stateless server to resume session
- Main goal: 0-RTT (reduce latency) exchange
  - Typical case: key already established, has cookie
    - Notice: 0-RTT request may be replayed
  - Related to 0-RTT exchange of TLS 1.3

Quic vs. Firewalls & other tools

- Easy to allow Quic to flow (open UDP:443)
- NAT: how to identify end of Quic session?
  - General UDP problem
  - May become harder as Quic gets popular
- Hard to validate, intercept Quic traffic
  - Adding the 'fake CA' to the OS certificate store is not sufficient: which fake cert to send??!
    - Good or bad? Depends whom you ask
    - E.g., prevents inspection by firewall, IDS/IPS,…
    - Possible response: block Quic !

Conclusions: TCP/IP security

- TCP/IP stack is insecure against MitM
  - Unless using crypto: IPsec, TLS, DNSSEC
- But also: many off-path attacks
  - Some we learned, many we didn't
  - Some trivial, many not so trivial
  - Most work only in some scenarios
    - Typically: with insecure versions, configurations
- Few of the many topics not covered:
  - IPv6 Security
  - Failure localization and accountability