Off-Path TCP Exploits: Global Rate Limit Considered Dangerous

Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao,
Srikanth Krishnamurthy, Lisa M. Marvel†

† ARL

Our TCP Attack

- Discovered a subtle TCP side channel vulnerability in Linux 3.6+ (CVE-2016-5696)
- Given any two *arbitrary* hosts on the internet, blind attacker can infer:
  - Existence of communication
  - Sequence number
  - ACK number
- Can be used towards:
  - TCP connection termination attack
  - Malicious data injection attack

Outline

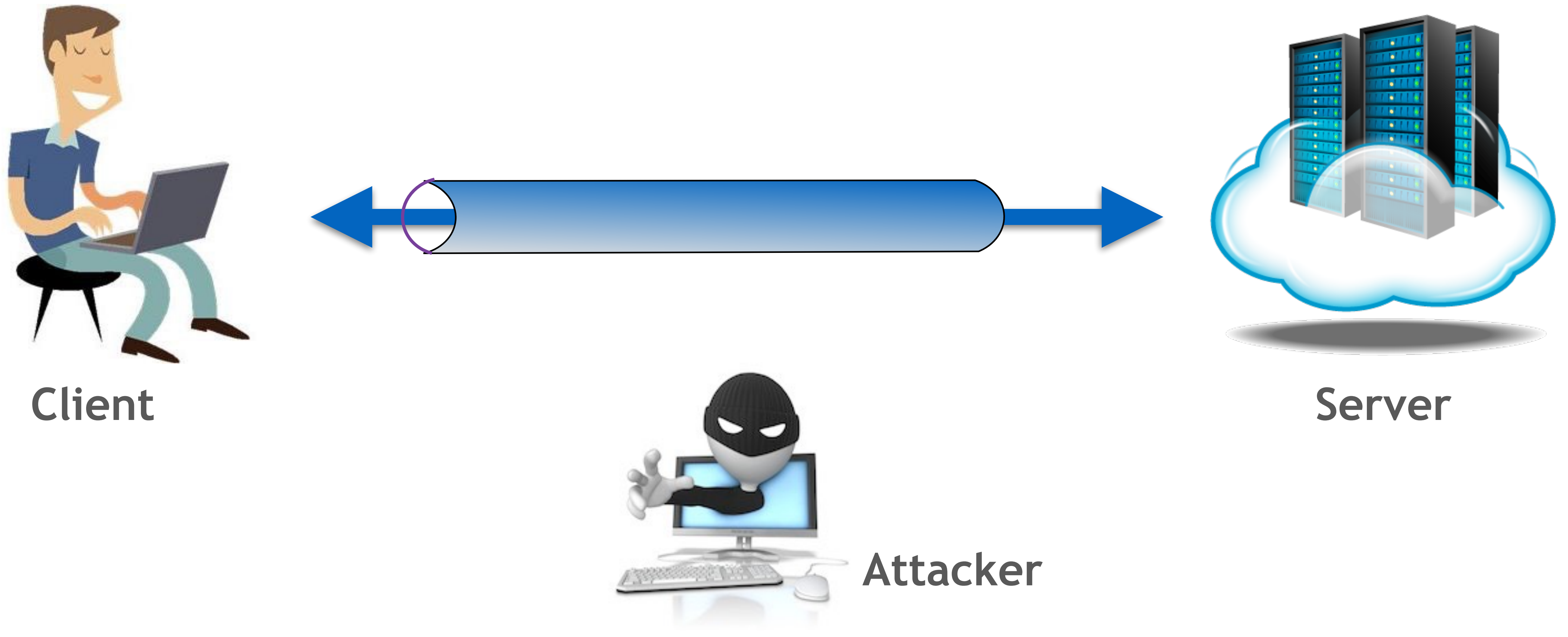
- Threat Model
- Background
- Vulnerability
- Our Attacks
- Evaluation
- Defense & Conclusion

Outline

- Threat Model
- Background
- Vulnerability
- Our Attack
- Evaluation
- Defense & Conclusion

Threat Model

- Consists of:
  - An arbitrary pair of client and server
  - A blind *off-path* attacker(no eavesdropping capability)
- Assumption: the attacker can send ***spoofed*** packets with the ***victim (client or server)'s IP address***



Threat Model

Outline

- Thread Model
- Background
  - History of RFC 5961
  - 3 modifications in RFC 5961
  - Why does this vulnerability exist?
- Vulnerability
- Our Attack
- Evaluation
- Defense & Conclusion

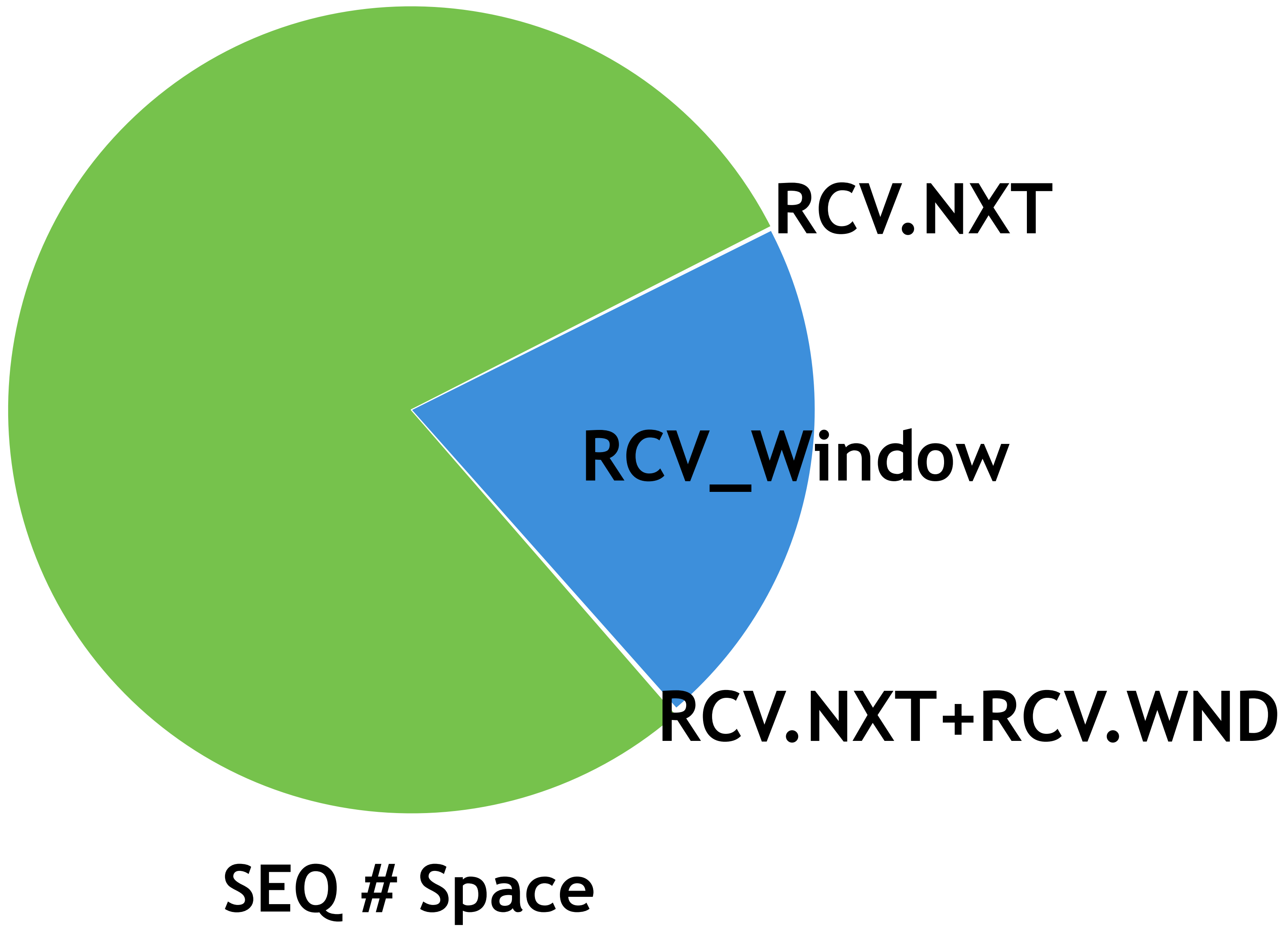
Background

- Traditional blind in-window attacks (brute force):
  - Connection termination & data injection attack
  - Success requirement (spoofed packet with):
    - Known 4-tuple <src IP, dst IP, src port, dst port>
    - Guessed SEQ # is in-window (recv window)

- RFC 5961 (Aug 2010)
  - Mitigate blind in-window attacks
  - Modification of receiving scheme
    - SYN receiving scheme
    - RST receiving scheme
    - Data receiving scheme
  - **Ironically**, Linux implementation introduced the side channel vulnerability

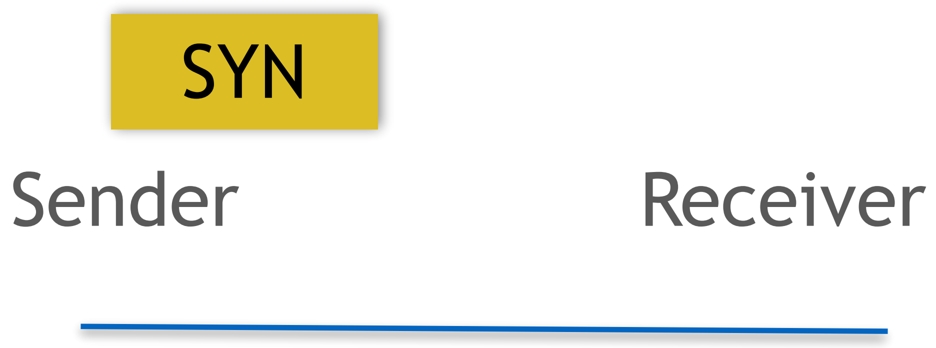
SYN Receiving Scheme

- Before RFC 5961: blind RST Attack by sending spoofed **SYN** packet



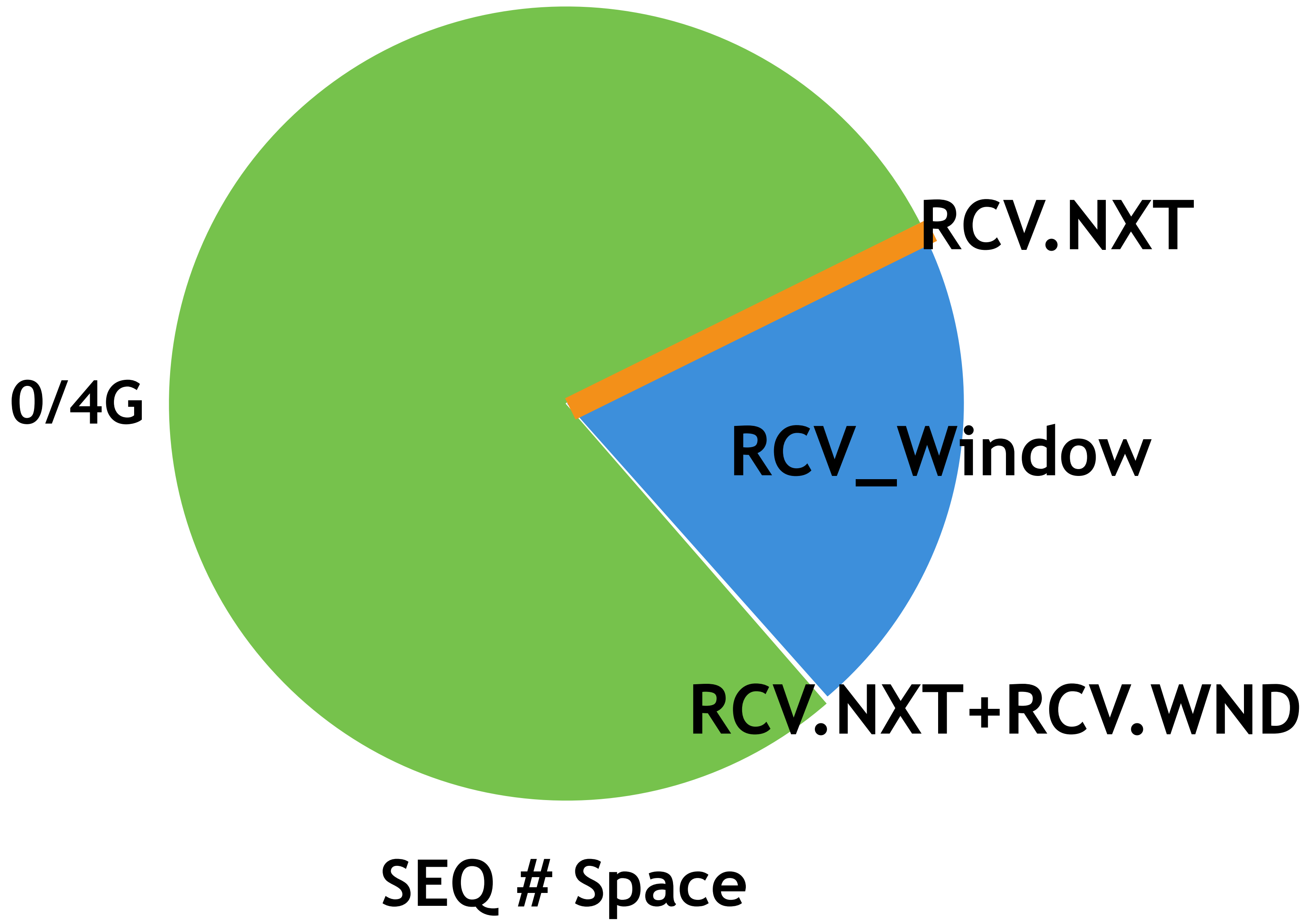
SEQ #:	Before RFC 5961	After RFC 5961
Out-of-Window	ACK back	Challenge ACK
In_Window	Reset Connection	Challenge ACK

Challenge ACK: ask sender to confirm if it indeed restarted

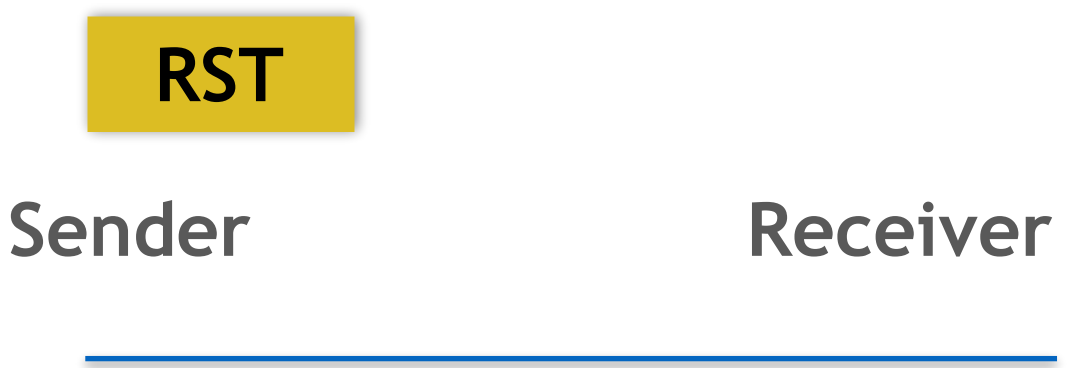


RST Receiving Scheme

- Before RFC 5961: blind RST Attack by sending spoofed **RST** packet



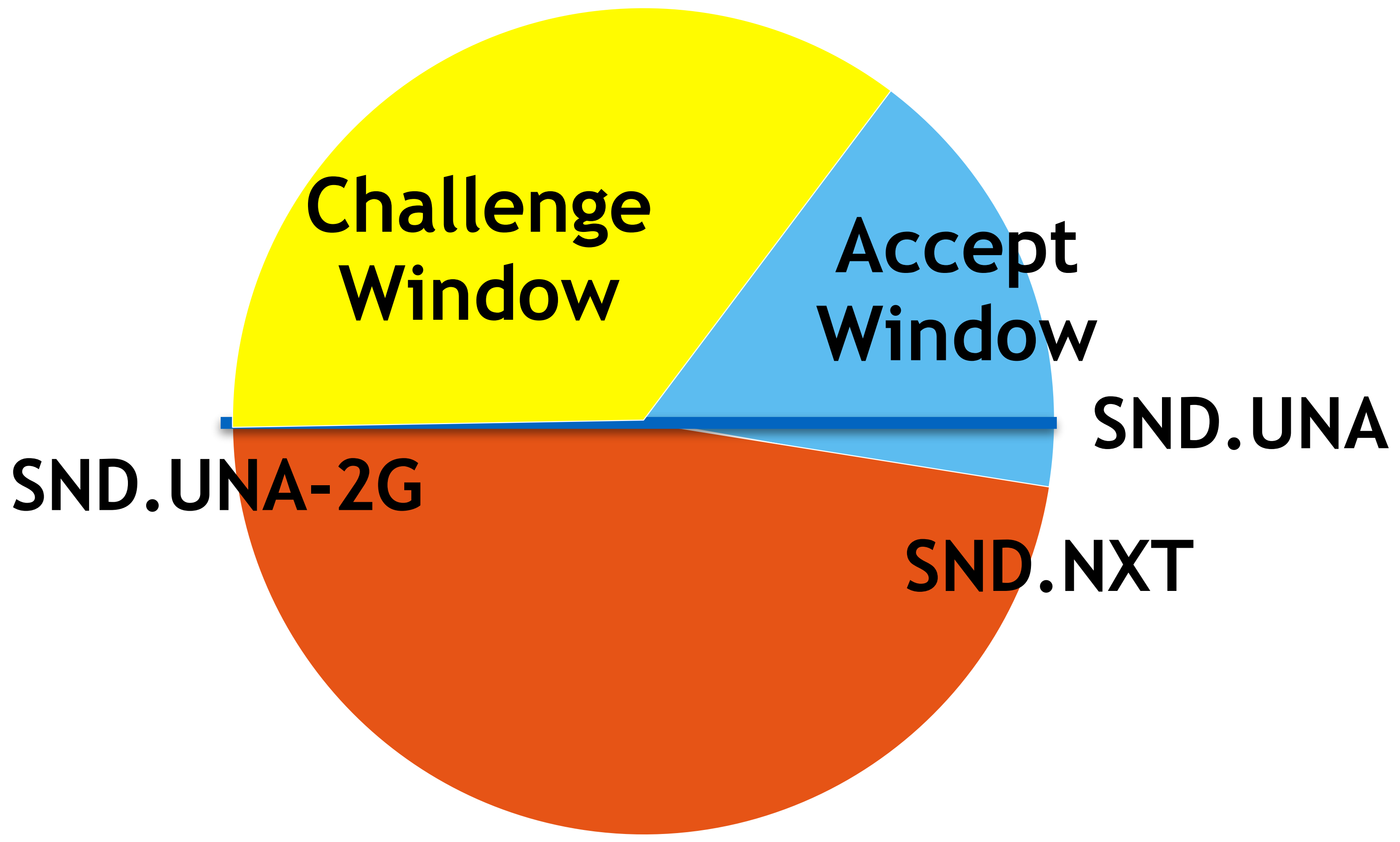
SEQ #:	Before RFC 5961	After RFC 5961
Out-of-Window	Drop the Packet	Drop the Packet
In-Window	Reset Connection	Challenge ACK
Exactly match		Reset Connection



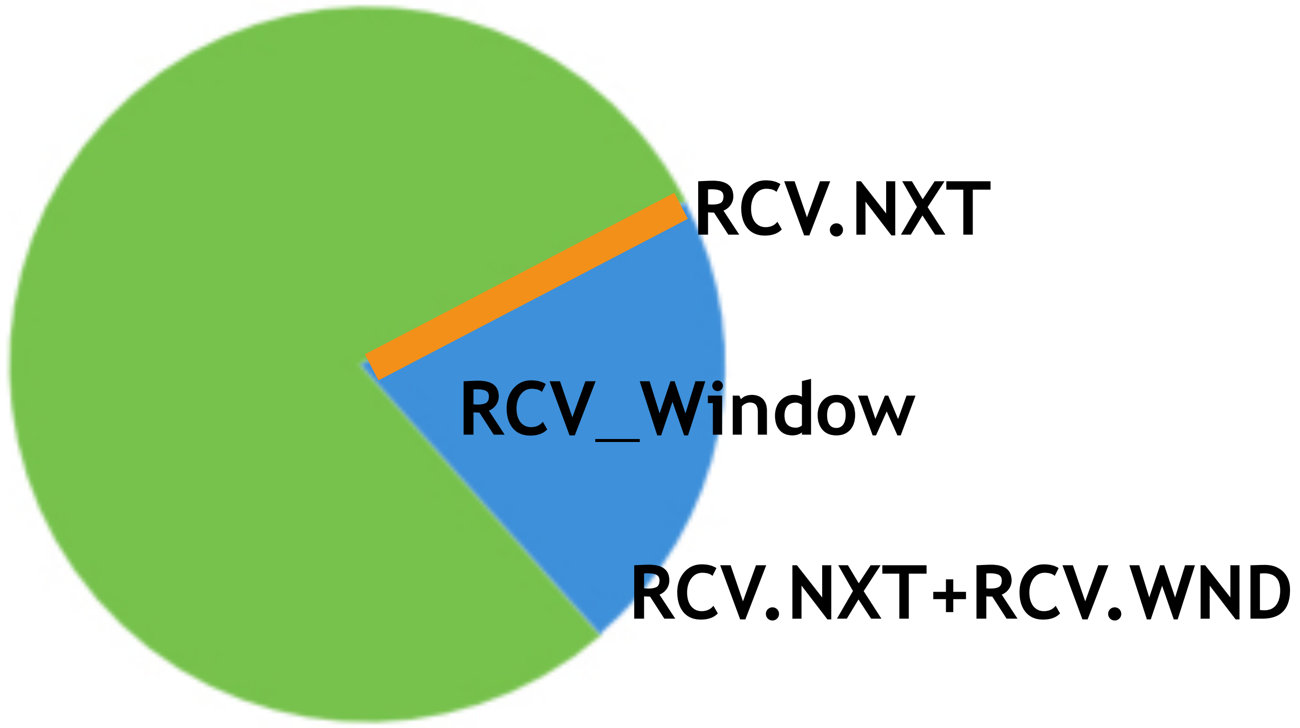
Challenge ACK: tell sender to confirm if it indeed terminated the connection

Data Receiving Scheme

- Before RFC 5961: blind Data Injection Attack by injecting spoofed **DATA** packet



ACK # Space



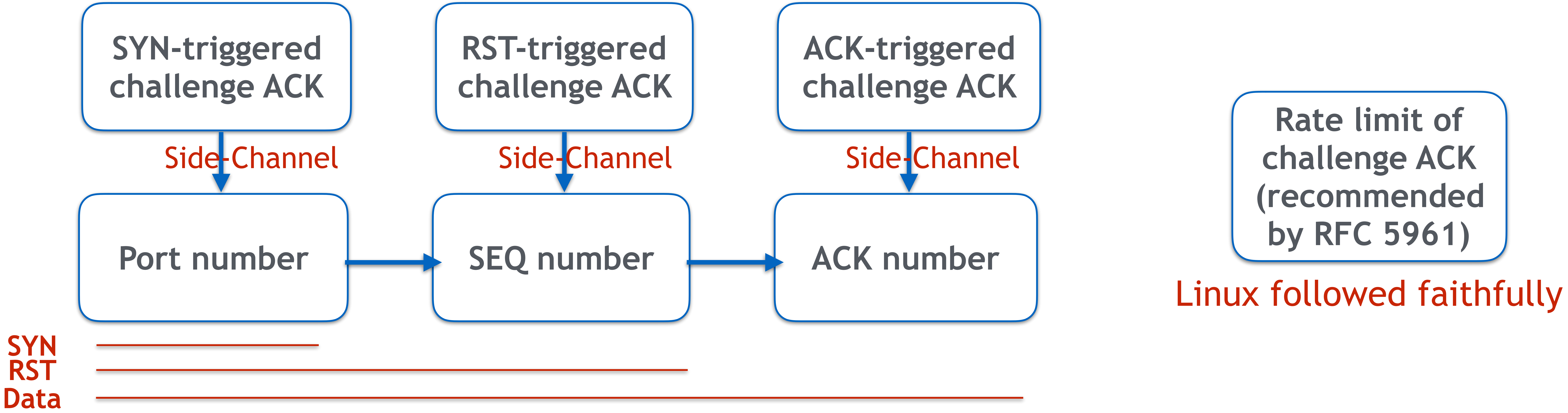
SEQ # Space

ACK #:	Before RFC 5961	After RFC 5961
Out-of-Window	Drop	Drop
In-Accpt_Window	Process Data	Process Data
Challenge Window (Old ACK)		Challenge ACK

SEQ #: In-RCV_Window —> Check ACK #

Why Does This Vulnerability Exist?

- RFC 5961: a much stricter check on incoming packets
- Challenge ACK is triggered in a established connection:
  - SYN packet with **correct 4-tuples <srcIP, dstIP, srcPort, dstPort>** (any SEQ #)
  - RST packet with **4-tuples, in-window SEQ #**
  - Data packet with **4-tuples, in-window SEQ #, old ACK #(in challenge window)**

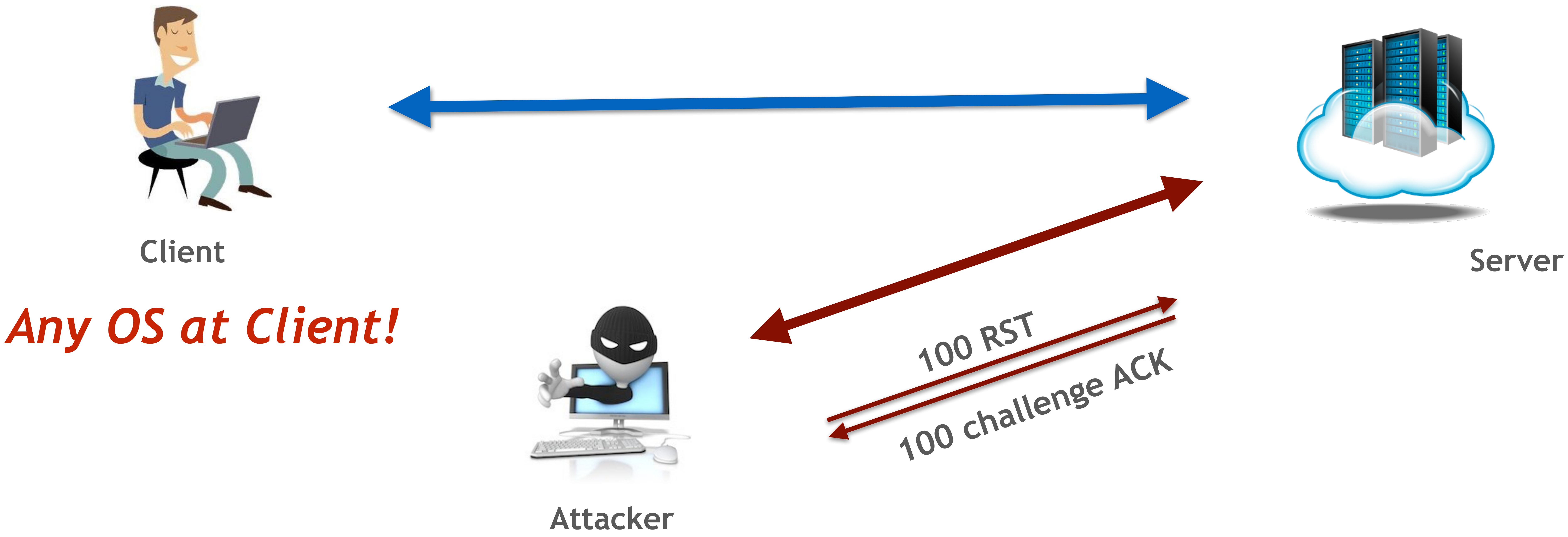


Outline

- Thread Model
- Background
- Vulnerability
  - Side channel vulnerability
  - Guess-Then-Check Method
  - Optimizations
- Our Attack
- Evaluation
- Defense & Conclusion

Side Channel Vulnerability

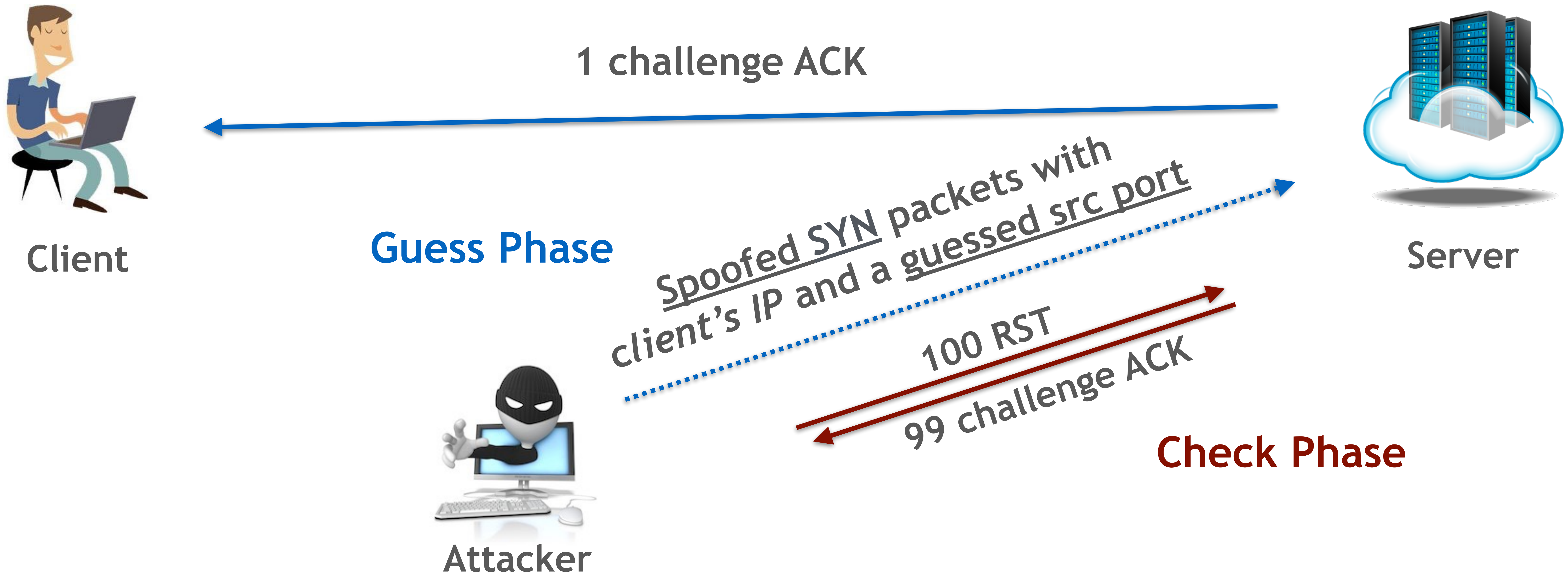
- sysctl_tcp_challenge_ack_limit: implemented in Linux 3.6+
  - *Global* limit of all challenge ACK per sec, ***shared across all connections***
  - Default value: 100 (**<u>reset</u>** per second)



Side-Channel Vulnerability Example

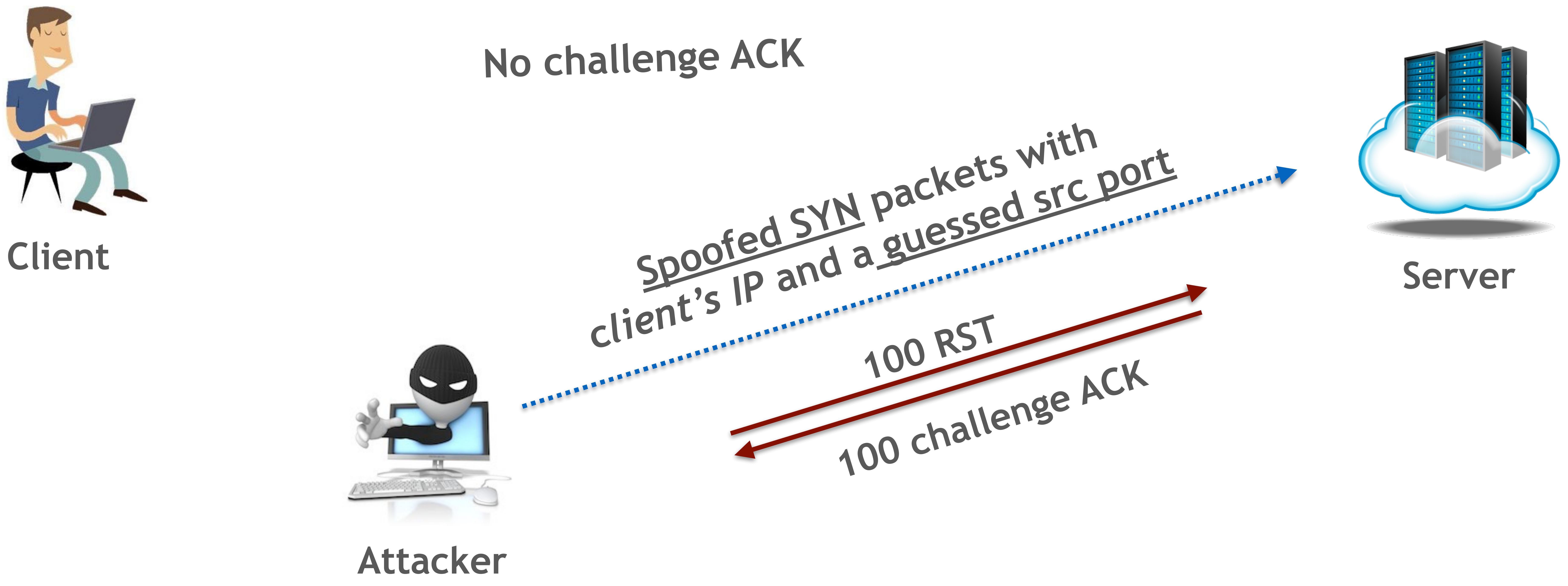
Exploit The Vulnerability

- Guess-then-Check method:
  - Send spoofed packets with guessed values
  - Example: to guess correct client-port number
    - If it's a correct guess:



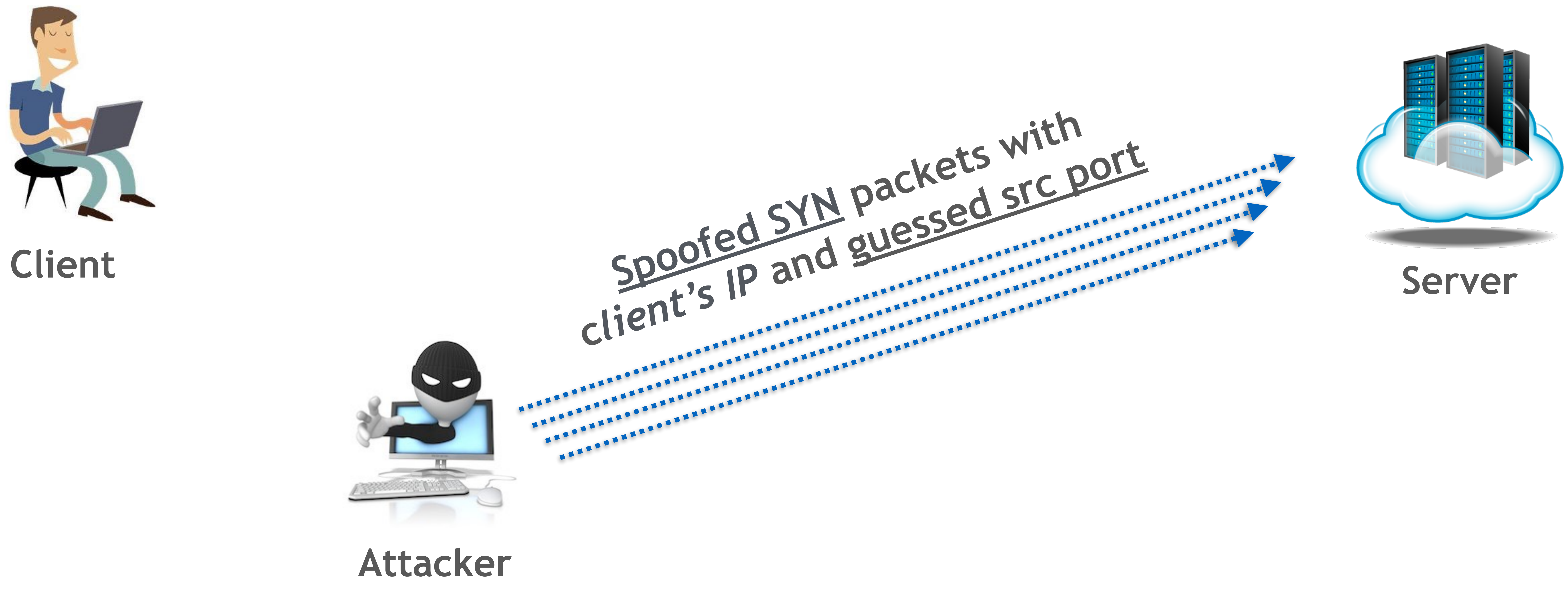
Guess-Then-Check Method

- Send spoofed packets with guessed values
  - Example: to guess correct client-port number
    - If it's a wrong guess:



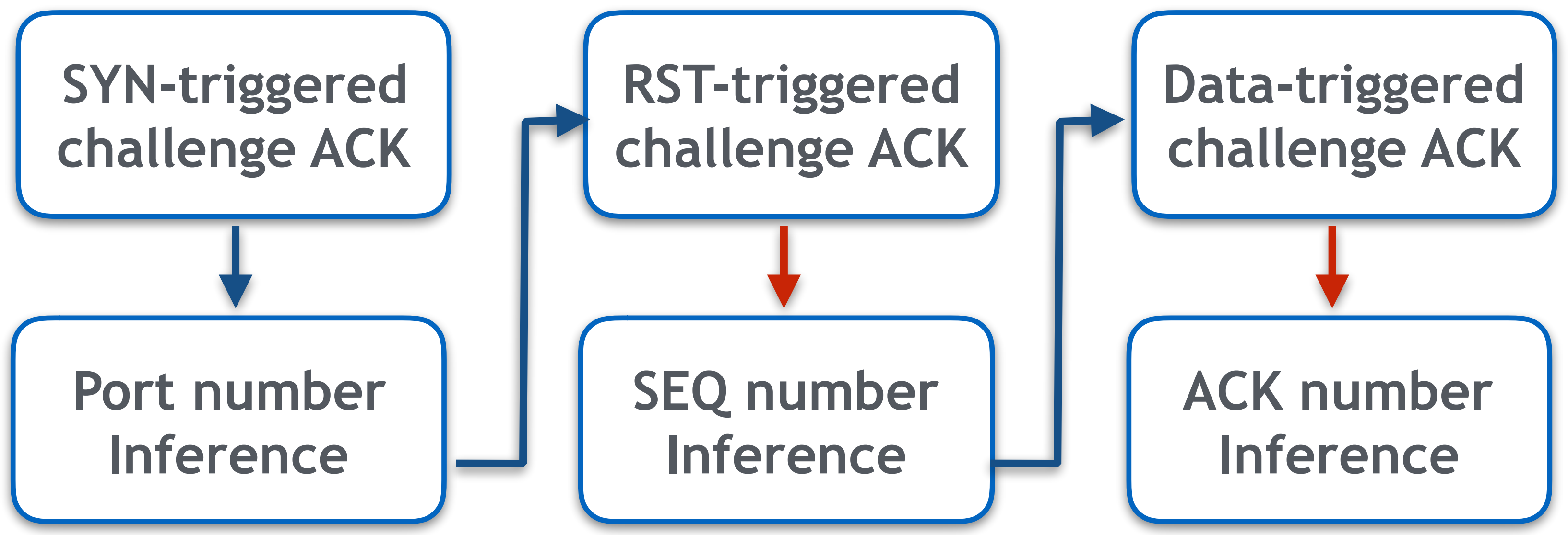
Guess-Then-Check Method

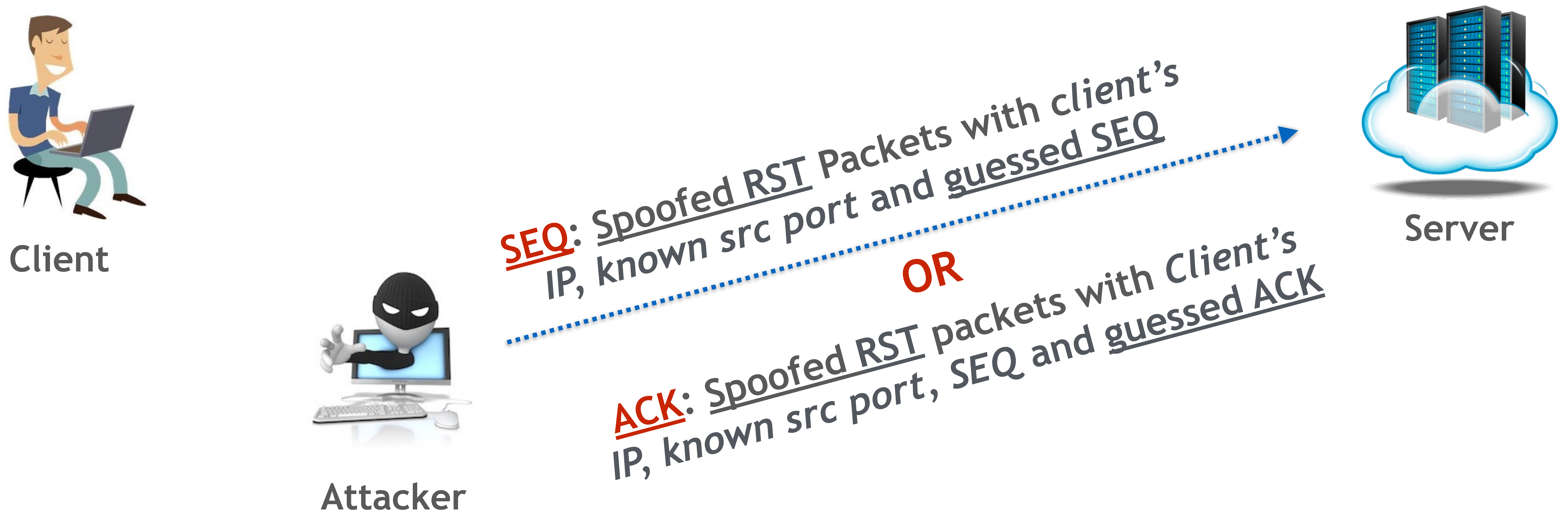
- Challenge: expensive time cost
- N: maximum spoofed probing packets in one second
  - Bandwidth dependent



Guess-Then-Check Method

- Same process works for guessing SEQ number and ACK number
- Correct guess:
  - SEQ number RST packet with correct 4-tuples, SEQ # in-window
  - ACK number Data packet with 4-tuples, SEQ # in-window, old ACK #



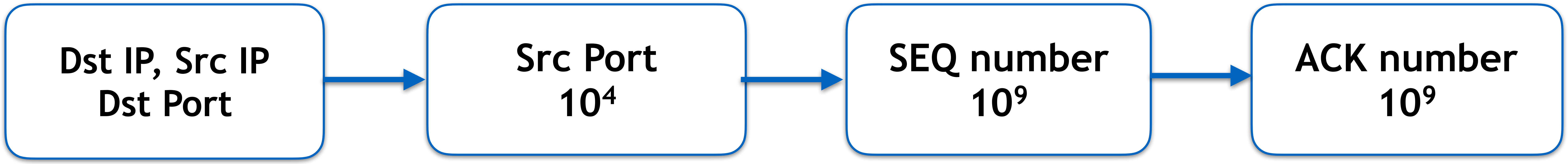


Guess-Then-Check Method

- Guess is correct when:
  - Src Port — **SYN** packet with correct **4-tuples(src Port)**
  - SEQ number — **RST** packet with correct **4-tuples, SEQ # in-window**
  - ACK number — **Data** packet with correct **4-tuples, SEQ # in-window, old ACK**

- Traditional brute-force attack: $10^4 \bullet 10^9 \bullet 10^9 = 10^{22}$ different combinations
- Our attack: Time cost is additive instead of multiplicative

Possible to finish within 1 minute!

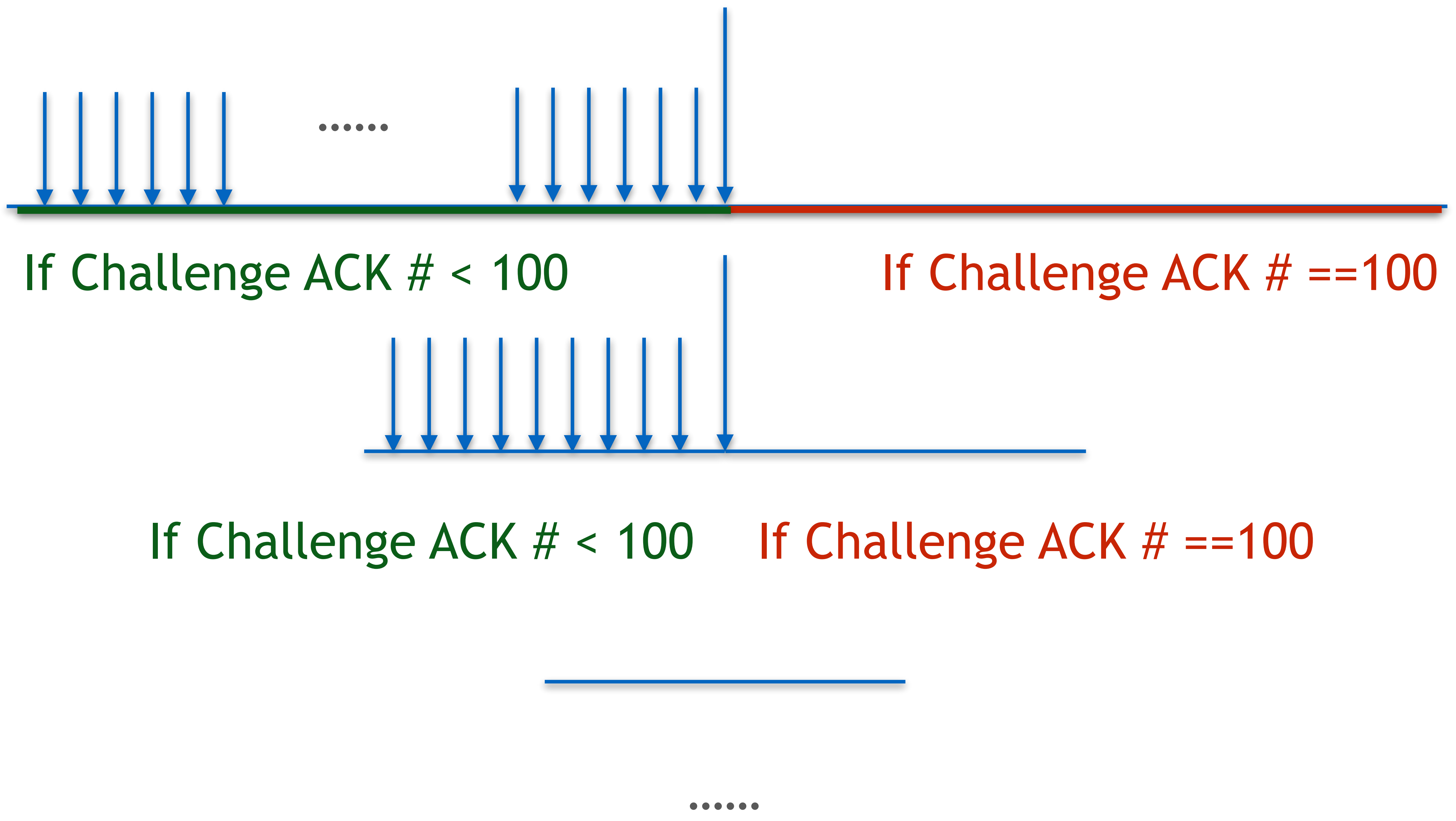


Optimizations

- Binary-style search
  - Reduce the number of probing rounds
- Multi-bin search
  - Further improvement
- Redundancy-encoded search
  - Account for packet loss

Binary-style Search

- Send spoofed packet for *all* the ports in the *1st half* range.
- Narrow down the search space *by half* and proceed to the next round



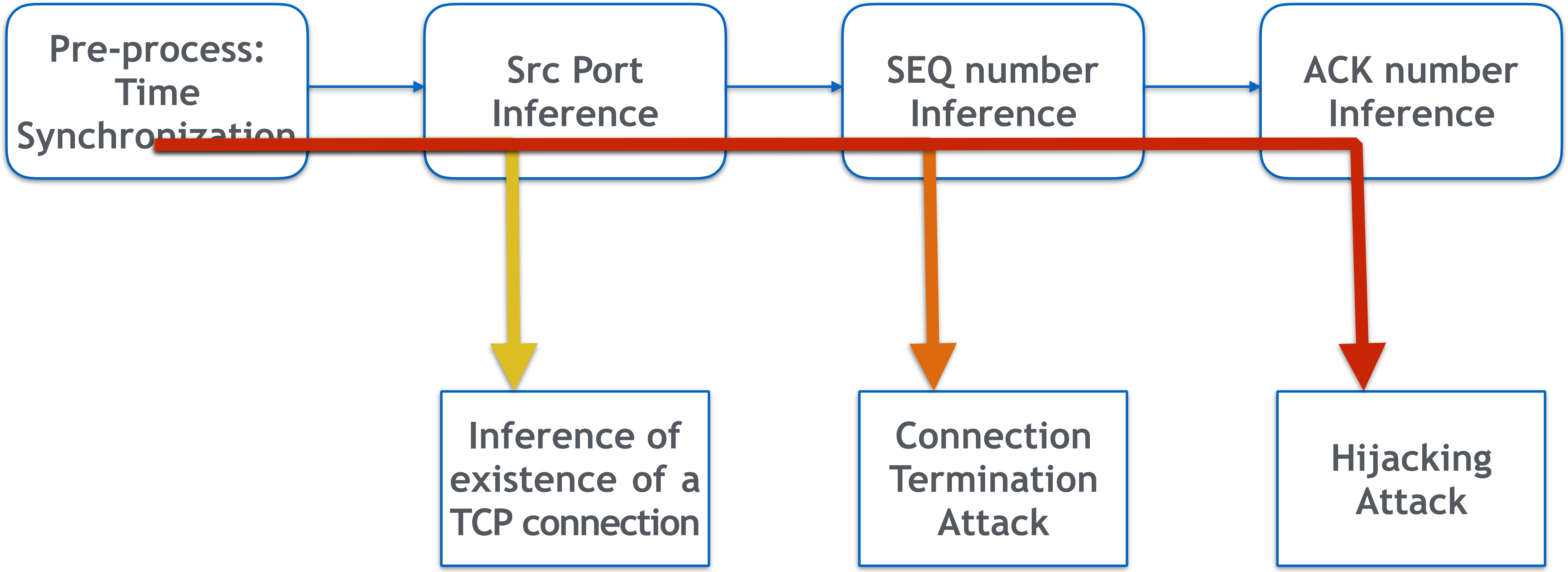
Binary Search Algorithm

Outline

- Thread Model
- Background
- Vulnerability
- Our Attack
  - Attack overview
  - Time synchronization
  - Inference of possible TCP connection
  - TCP connection termination attack
  - TCP hijacking attack
- Evaluation
- Defense & Conclusion

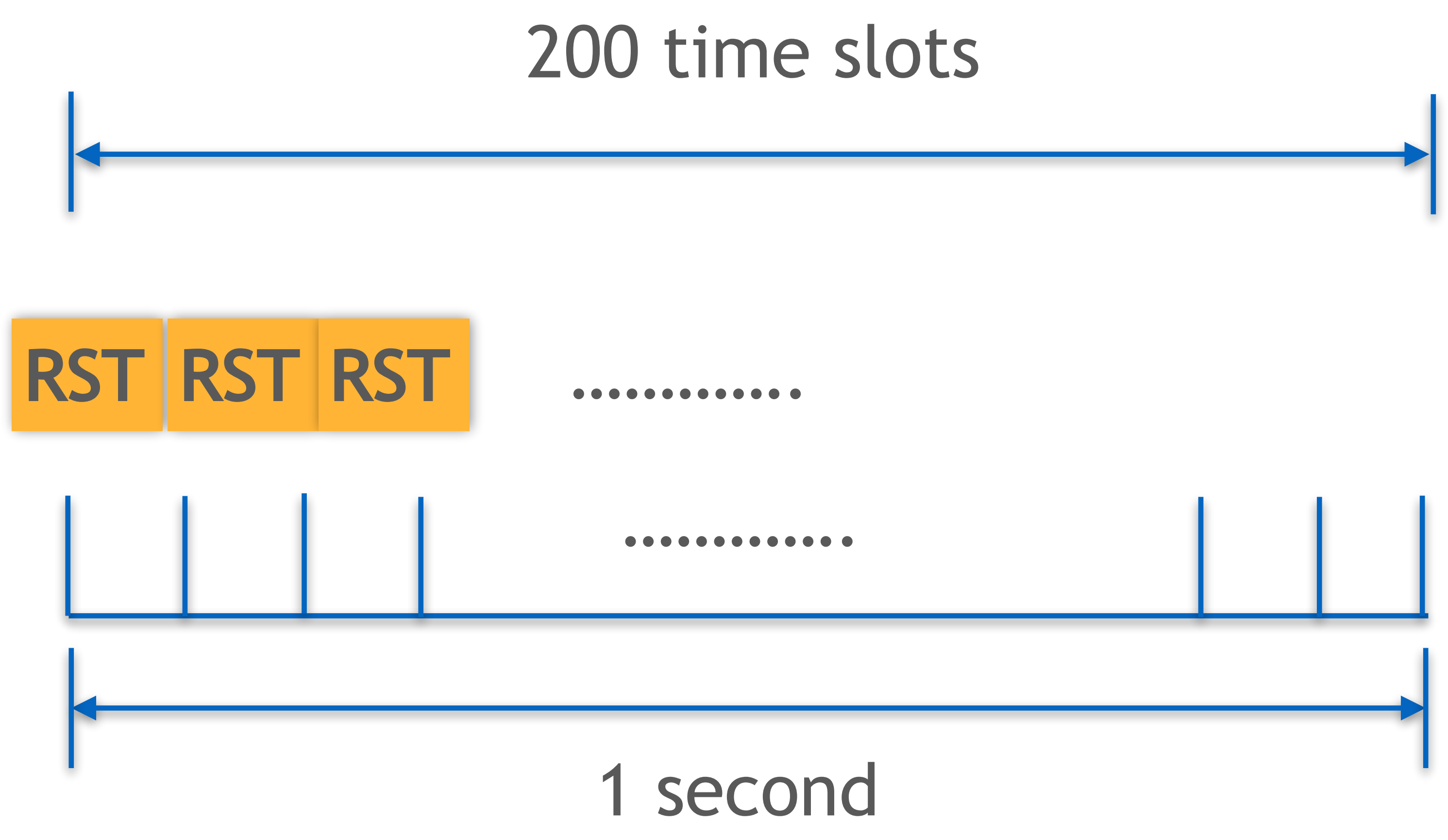
Attack Overview

- Given client and server, we already know:
  - Src IP address: client IP
  - Dst IP address: server IP
  - Dst Port number: service at server(e.g. 80)



Time Synchronization

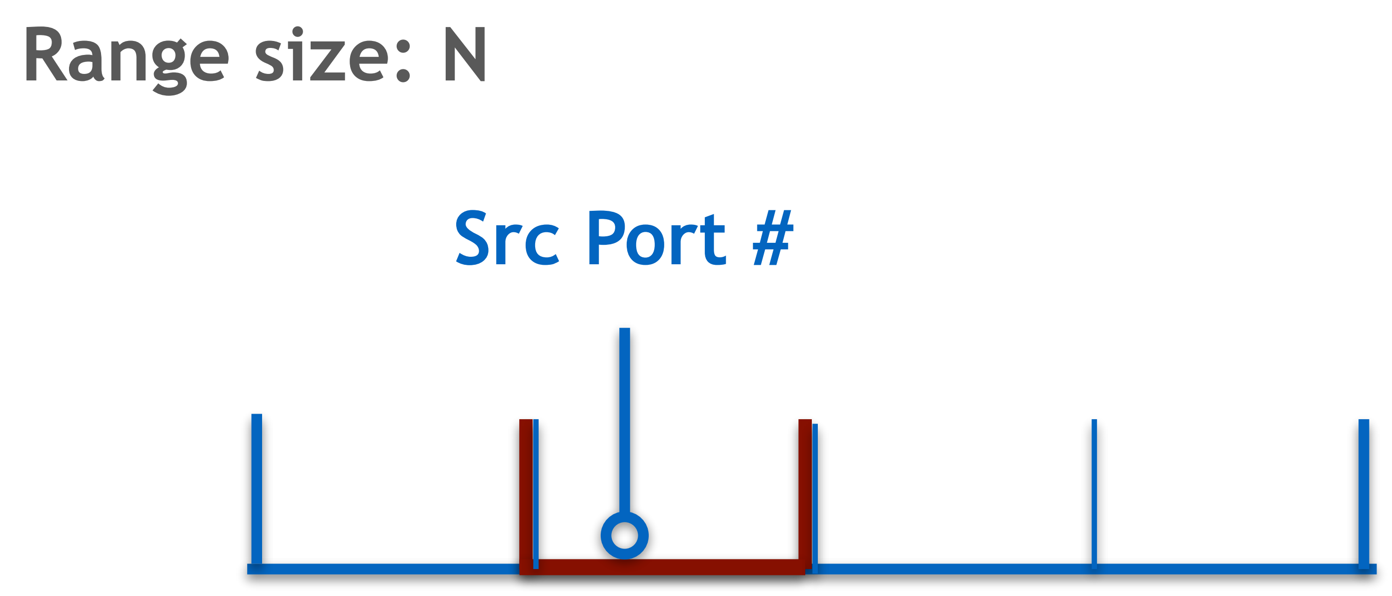
- Challenge:
  - Challenge ACK count resets each second
  - All the spoofed and non-spoofed packets MUST be within the same 1-second interval at server
- Our own method:
  - A time synchronization strategy based on this side channel



Time synchronization example

Inference Of Possible TCP Connection

- Given src IP, dst IP and expected dst port:
  - To see if client opened a port
- To infer src port:
  - 1. Throughout all port number[probe N ports in 1 sec]
    - To infer *connection exists or not*
  - 2. Find exact correct port number[Binary/Multi-bin search]
    - To be used for termination attacker or hijacking attack

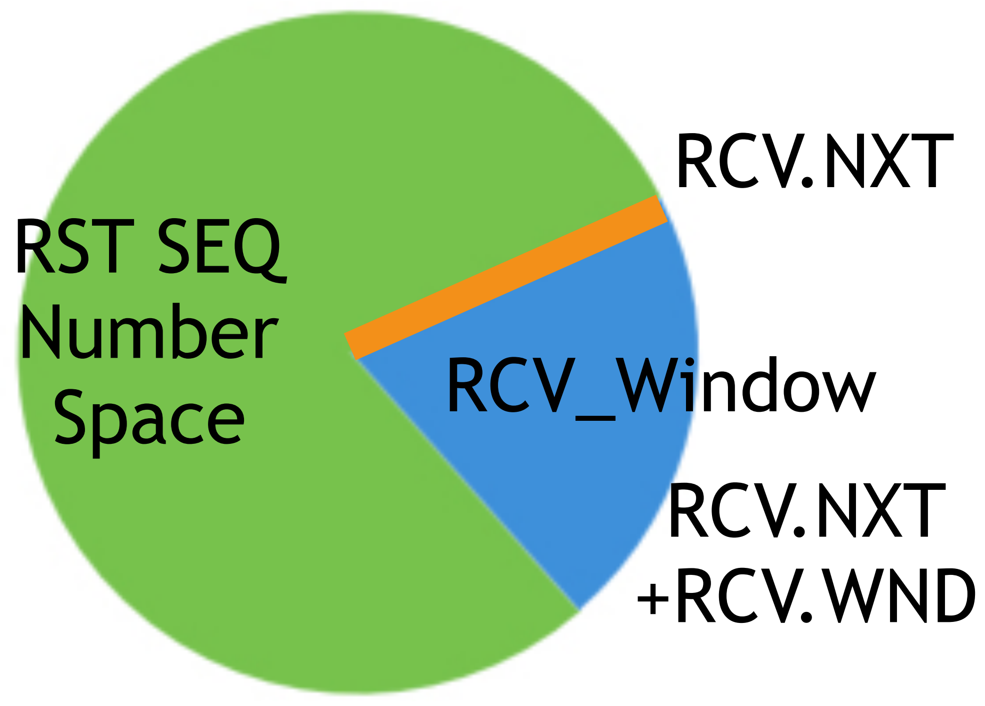


Step1: Identify Port Range

Step2: Identify Exact Port

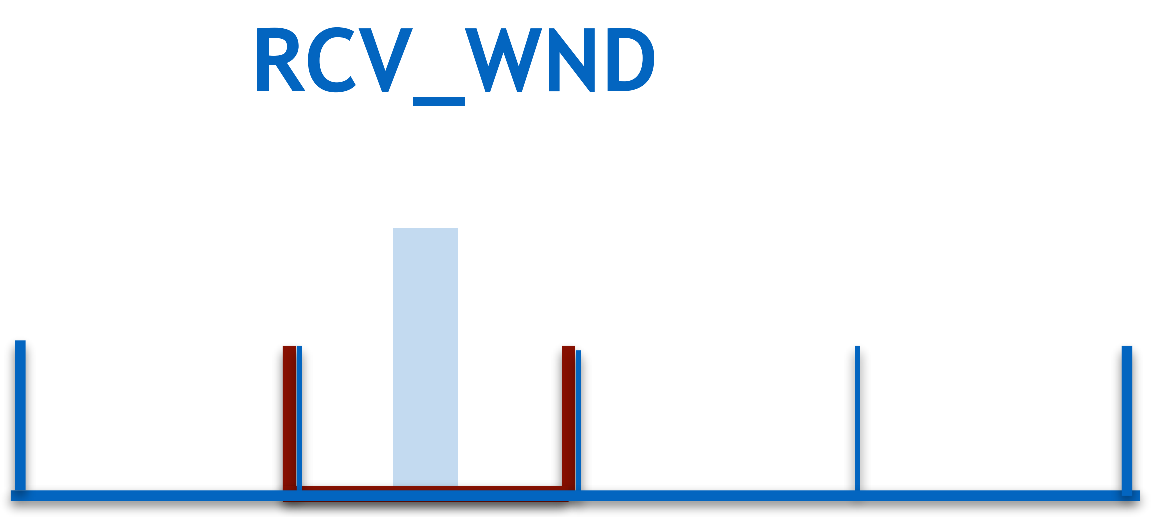
TCP Connection Termination Attack

- Given 4-tuples: src IP, dst IP, **src Port**, dst Port,
  - To send a RST packet with exactly **matched SEQ #**
- Optimization: locate receive window first, then specific SEQ number

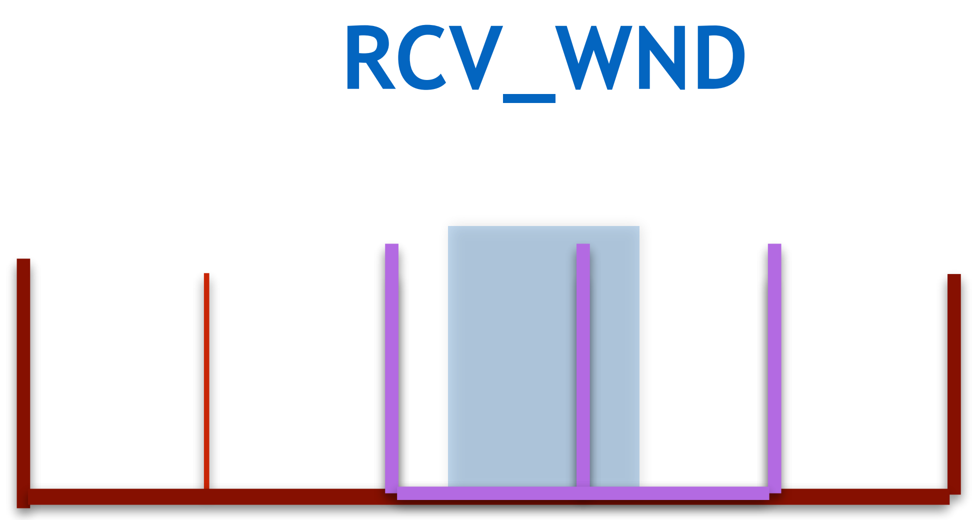


SEQ # Space

Range Size: N*Win_size

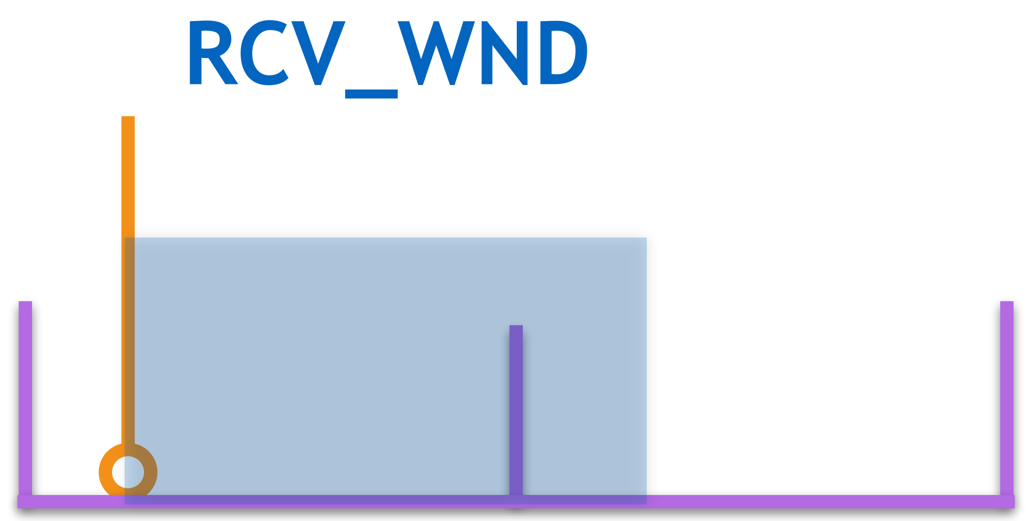


Step1: identify the window range



Step2: narrow down to a single window

Find Receive Window

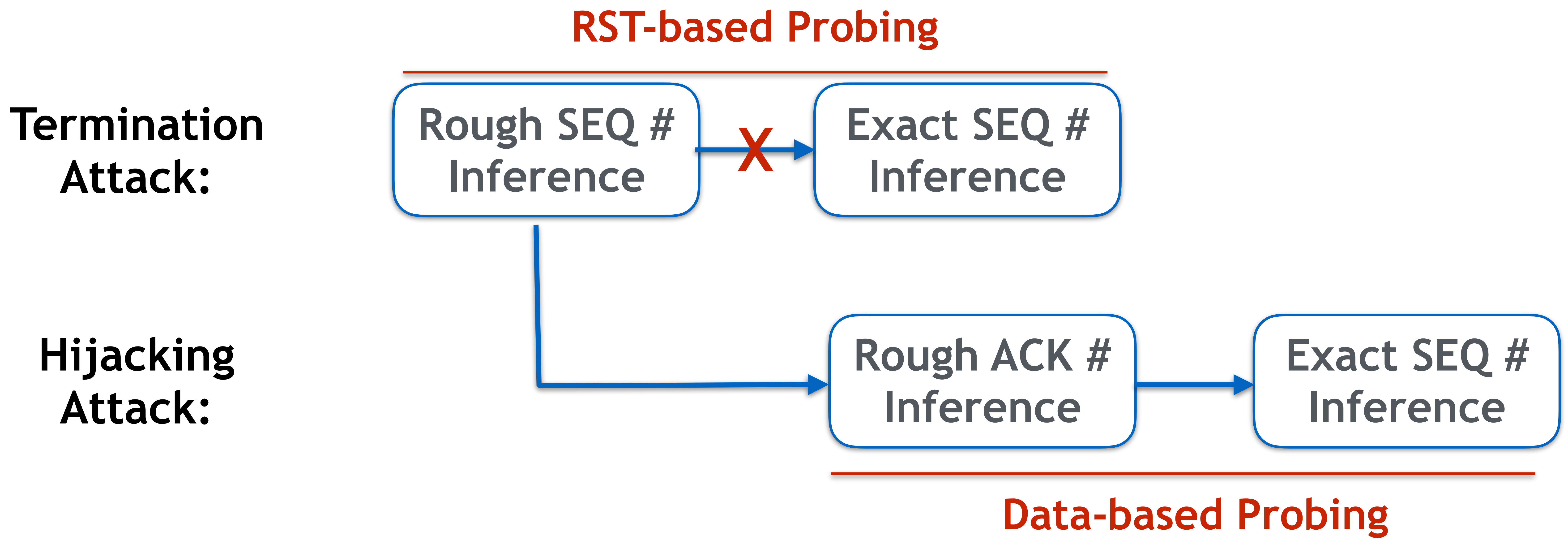


Step3: probe RCV.NXT

Find Exact SEQ #

TCP Hijacking Attack

- Challenge: a RST packet with correct SYN packet will terminate the connection
- Main idea (take a detour):
  - 1. Locate rough SEQ # in-window (same as before)
  - 2. Use Data-based probing to infer a rough ACK # in window
  - 3. Use Data-based probing to infer exact SEQ #



Outline

- Thread Model
- Background
- Vulnerability
- Our Attack
- Evaluation
  - Time micro-analysis
  - Case study: termination attack
  - Case study: hijacking attack
- Defense & Conclusion

Evaluation: Time Cost

- Time Micro-analysis:
  - Time cost differences in each step between Binary search and Multi-bin search
  - Time cost vs bandwidth

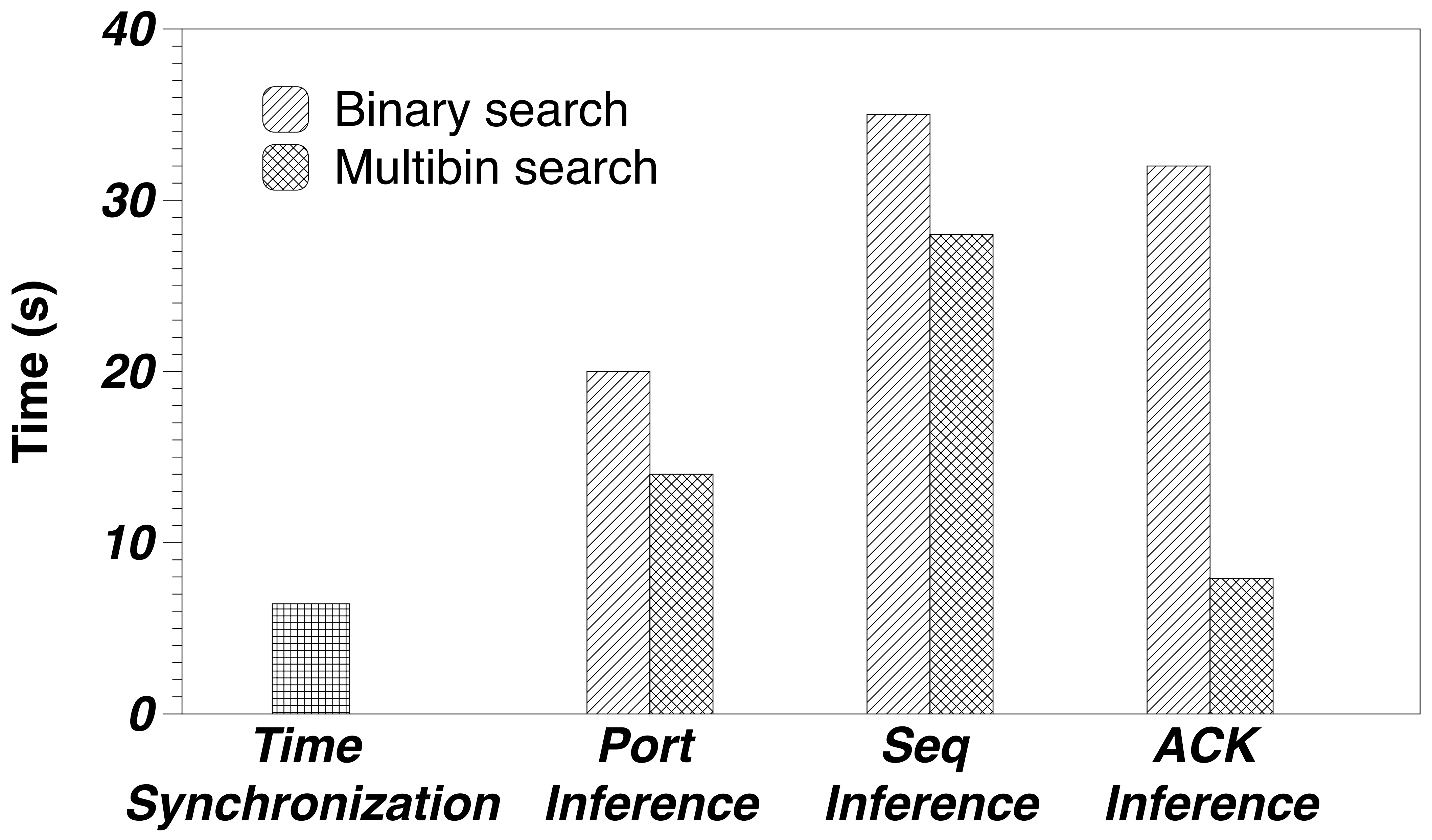


Fig1. Time Breakdown

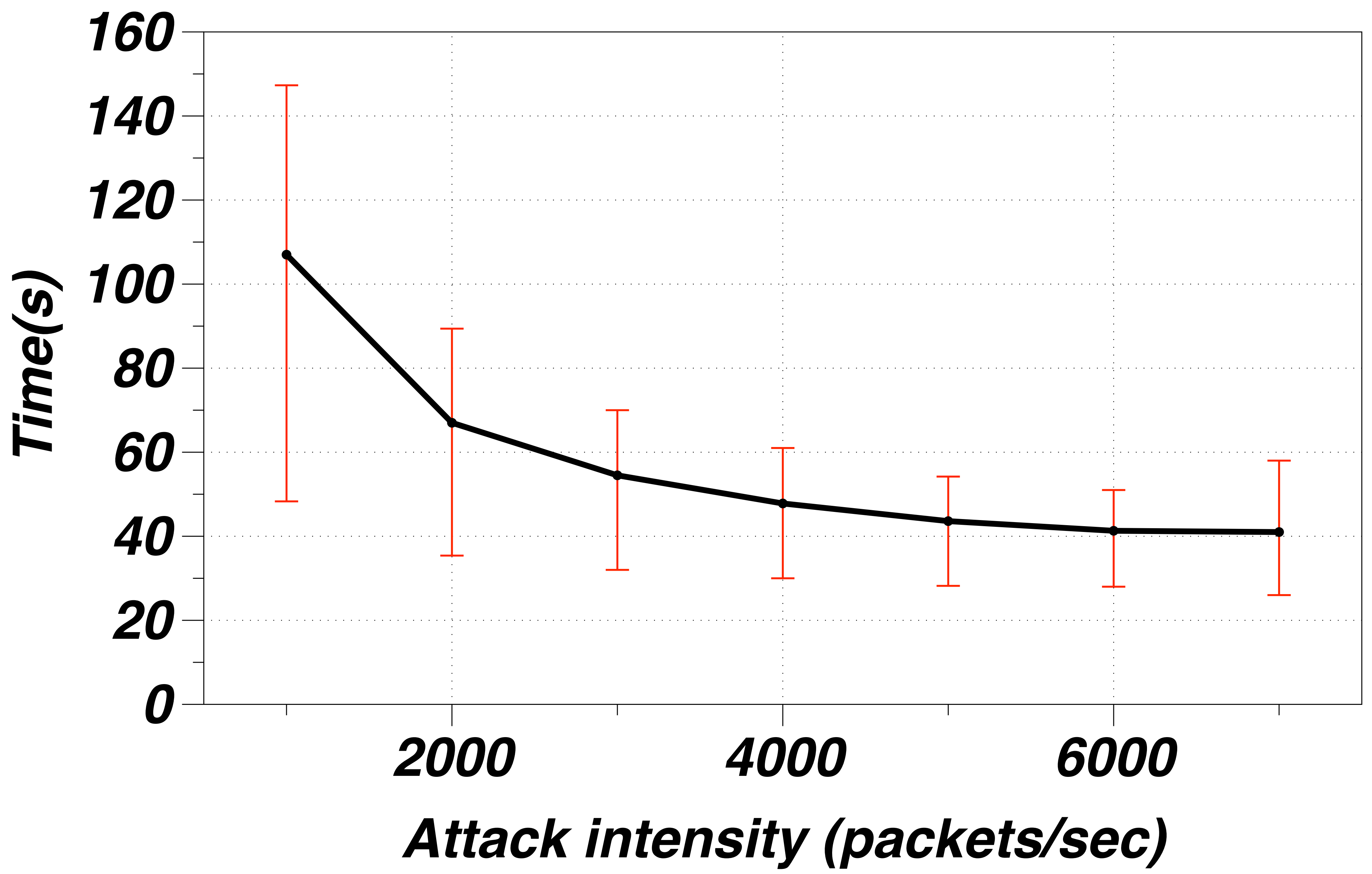
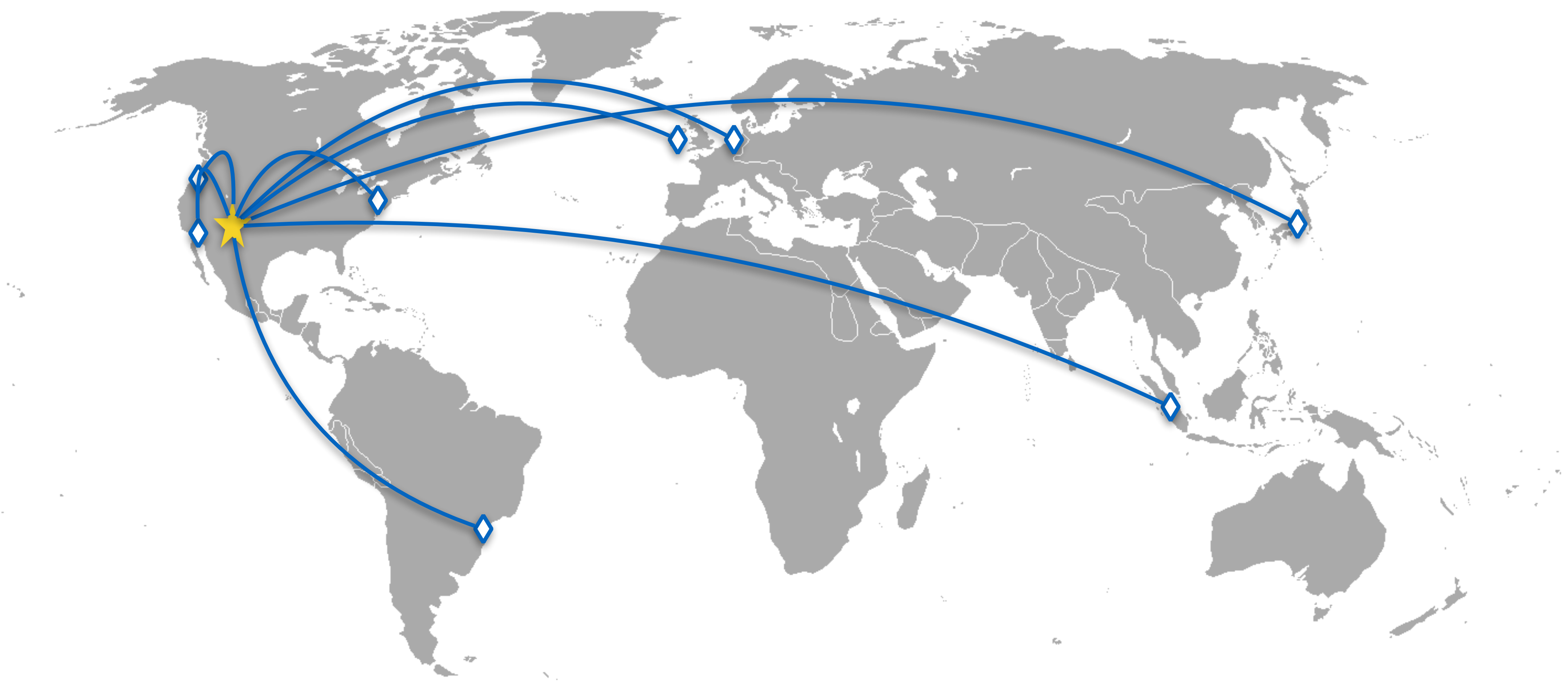


Fig2. Attack intensity impact on time to succeed

Case Study: Termination Attack

- Setting: client and attacker at different part of campus
- EC2: 8 different regions
  - Success rate: 96%
  - Attack time: ~42s

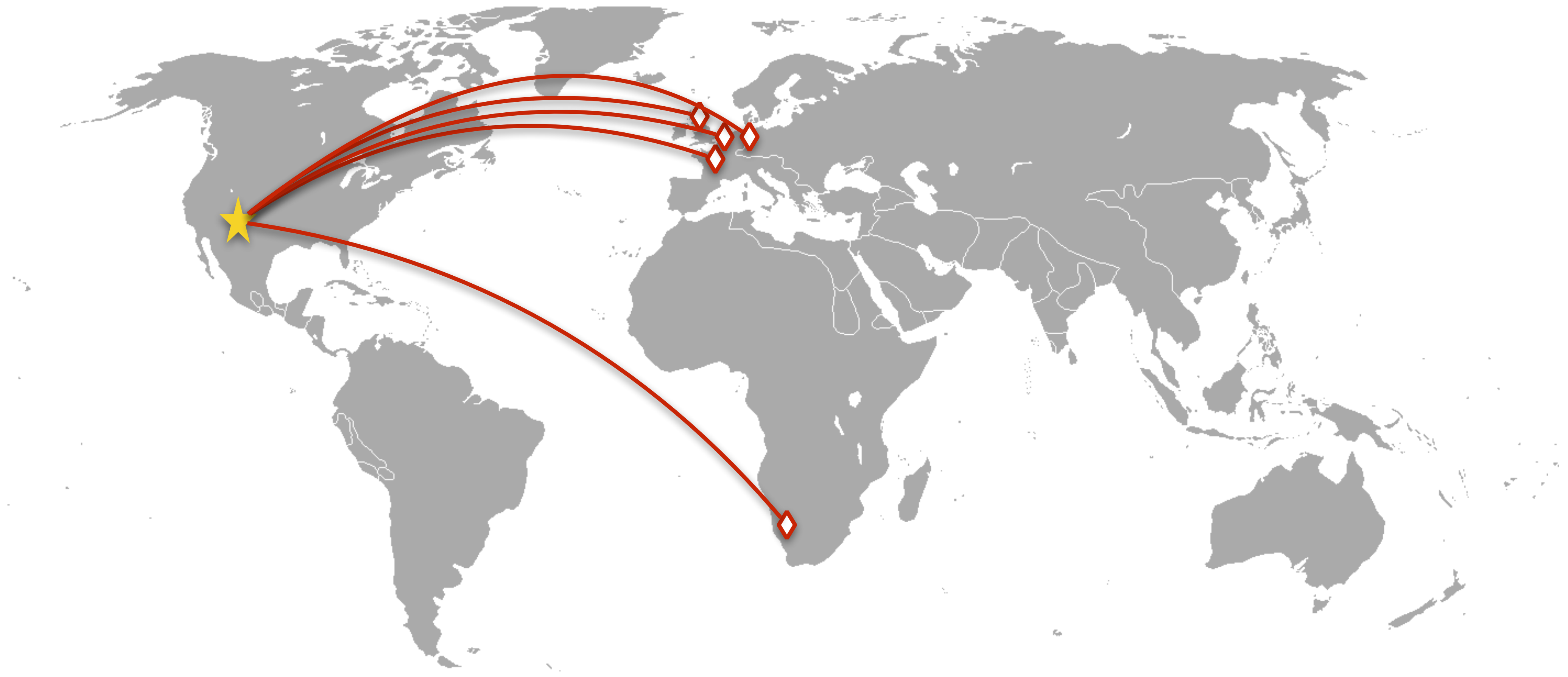


Location	Success Rate	Avg # of rounds with loss	Avg % of rounds with loss	BW (pkts)	Time Cost (s)
US West 1	10/10	0	0	5000	48.00
US West 2	9/10	1.0	1.91%	5000	58.00
US East	10/10	0	0	5000	32.00
EU German	9/10	0.3	0.67%	5000	48.00
EU Ireland	10/10	0	0	5000	35.20
Asia 1	10/10	0	0	5000	51.00
Asia 2	9/10	1.7	5.34%	5000	36.67
South America	10/10	0	0	5000	45.70

Table 1: SSH connection reset results

Evaluation: Hijacking Attack

- Setting: client and attacker at different part of campus
- Tor: 8 different regions
  - Success rate: 89%
  - Attack time: ~61s
-



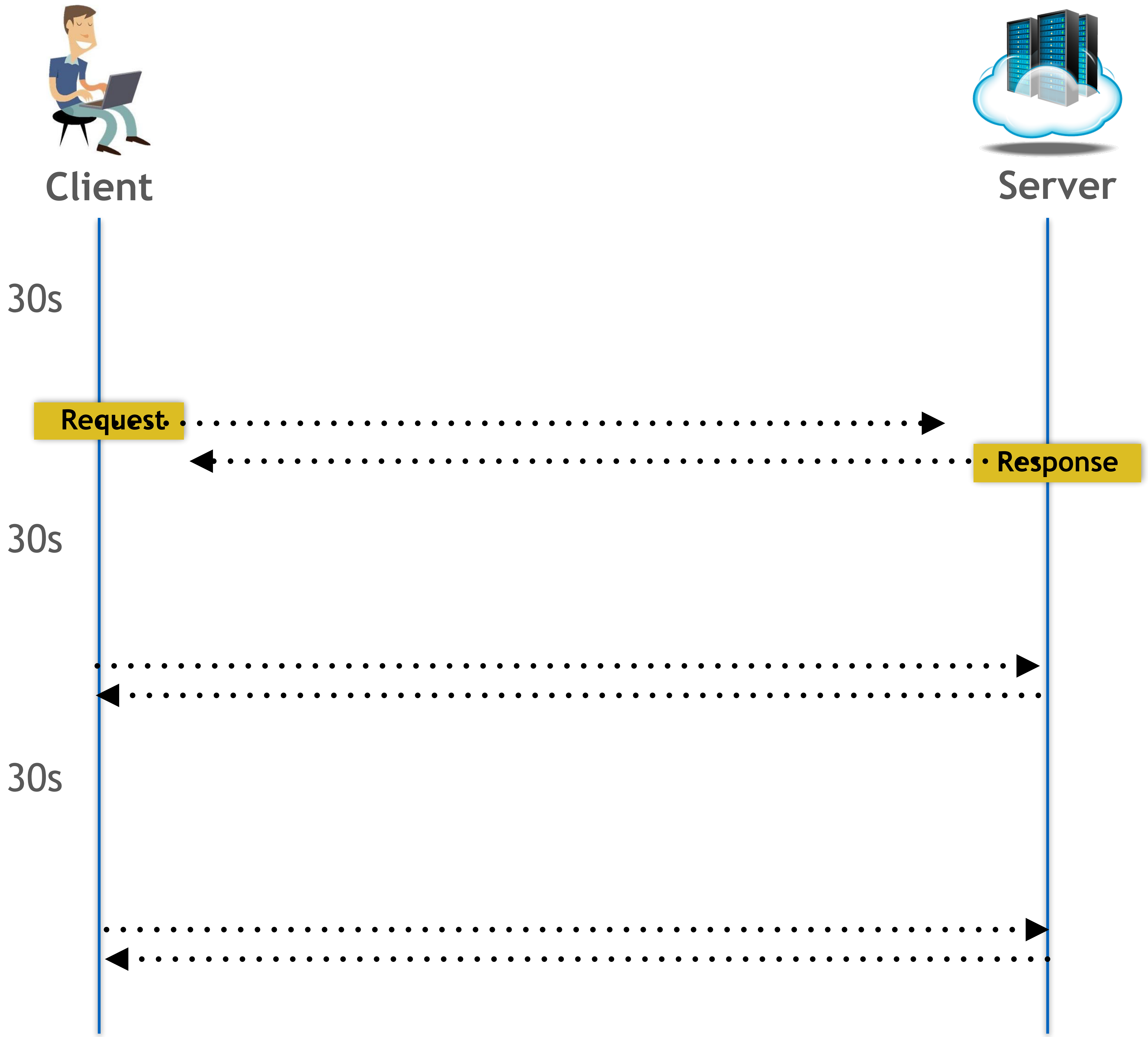
Node	Target	Success Rate	Avg # of rounds with loss	Avg % of rounds with loss	BW (pkts)	Time Cost(s)
62.210.x.x	FR	8/10	1.9	4.58%	4000	46.36
89.163.x.x	DE	9/10	4.0	7.97%	4000	49.08
178.62.x.x	GB	7/10	3.2	4.20%	4000	53.00
198.27.x.x	NA	10/10	0.8	1.45%	4000	59.86
192.150.x.x	NL	8/10	4.1	5.64%	4000	68.03
62.210.x.x	FR	6/10	2.5	5.85%	4000	49.57
89.163.x.x	DE	8/10	1.7	3.06%	4000	52.51
178.62.x.x	GB	8/10	6.0	8.15%	4000	78.35
198.27.x.x	NA	7/10	2.1	3.64%	4000	72.49
192.150.x.x	NL	6/10	5.5	7.14%	4000	79.42

Table 2: Tor connection reset results (first half under browsing traffic and second half under file downloading traffic)

Evaluation: Hijacking Attack

- Target: long-lived TCP connection without using SSL/TLS
  - news website
  - advertisements connection

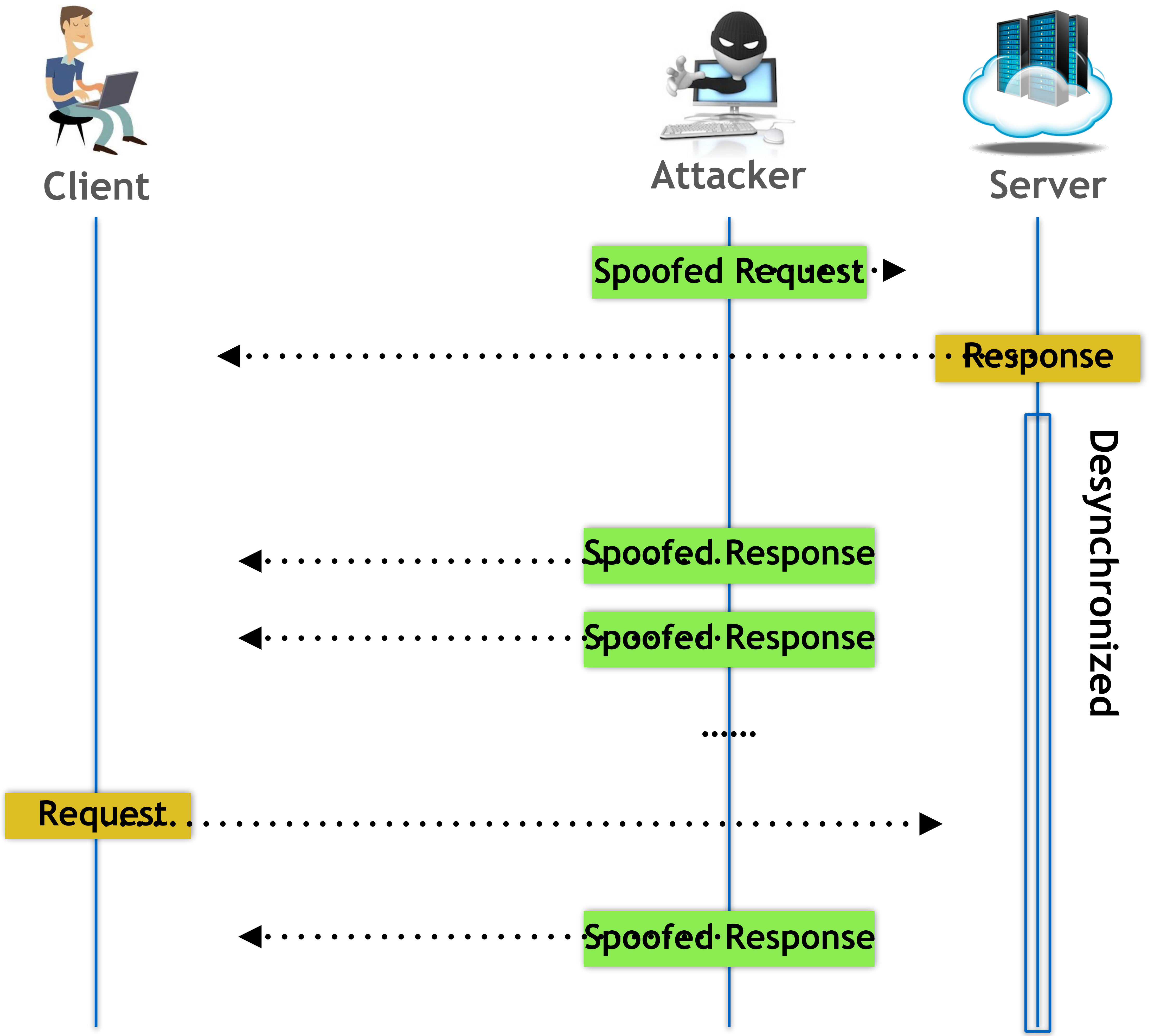
- Behavior at USAToday:
  - Client refreshes data periodically(30s)
  - Requests may vary during time



Evaluation: Hijacking Attack

- Hijacking: the usatoday.com website
  - Desynchronization[1]
  - Injection

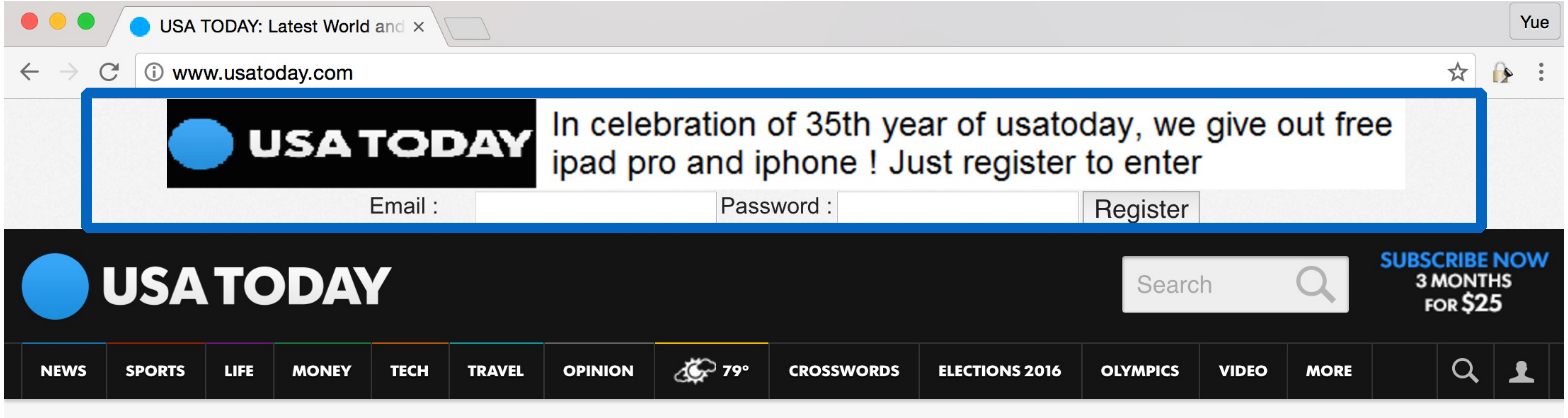
30s



[1]ABRAMOV, R., AND HERZBERG, A. Tcp ack storm dos attacks. Journal Computers and Security (2013).

Evaluation: Hijacking Attack

- Hijacking: the usatoday.com website



- Success rate of inferring the correct sequence and ACK number: 90%
- Success rate of injecting the phishing window: 70%
- Average Time Cost: 81.05s (with BW: 5000 pkt/s)

Outline

- Thread Model
- Background
- Vulnerability
- Our Attack
- Evaluation
- Defense & Conclusion

Defense & Conclusion

- Our defense scheme:
  - Add random noise to the channel (global challenge ACK rate limit)
  - Eliminate the side channel
  - Set sysctl_tcp_challenge_ack_limit to extremly large value[temporary]

Patched in Linux kernel 4.7 in July 2016

- Conclusion
  - Discovered a subtle yet critical flaw in the design and implementation of TCP in Linux 3.6+
  - Demonstrated blind off-path TCP attacks within ~1 minute
  - Proposed defense schemes

Thank you!
Q & A

Yue Cao
ycao009@ucr.edu