Denial of Service

Denial of Service

So far, we have looked at attacks that are about making a system do something that it **shouldn't**

- jumping to the wrong code via *buffer overflow*
- running user input as code via *code injection*
- leaking information through *side channels*

A **denial of service (DoS)** attack is about preventing a system from doing what it **should**

- making a server unresponsive to requests
- preventing an OS from running useful processes

Denial of Service

So far, we have looked at attacks that are about making a system do something that it **shouldn't**

- jumping to the wrong code via *buffer overflow*
- running user input as code via *code injection*
- leaking information through *side channels*

A **denial of service (DoS)** attack is about preventing a system from doing what it **should**

- making a server unresponsive to requests
- preventing an OS from running useful processes

} by keeping the system busy or just crashing it

Why DoS Attacks?

Some possible reasons:

- Disadvantage a competitor
- Revenge
- Extortion

... and because it's not that difficult

Denial of Service

In general, a DoS involves using up some resource

- processing time
- network bandwidth
- memory
- ...

Attacker's cost needs to be ≪ victim's cost

Using Up an OS Resource

```
while (true) {
  fork();
}
```

A **fork bomb** like this tends to be a short-lived bug instead of an attack!

Resources used up: process IDs, thread-scheduler time

Why it "works":

easier to ask for a new process than to create one

Network Attacks

Sending many requests from host to server is unlikely to create DoS: server's resources are likely on par with host's

Successful attacks usually depend on one of these:

- using a small request to provoke an expensive response
- using a cheap resource to occupy a rare resource
- exploiting bystander hosts to offload costs
- exploiting compromised bystander hosts to form a **botnet**

Expensive Response

Be careful about providing public interfaces to arbitrary computation

Examples:

- Handin server that runs homework submissions
- Directory service that allows arbitrarily complex queries
- File-format conversion service

Any service like these will need sandboxing and time/memory limits

Occupying a Rare Resource

The HTTP protocol does not put a limit on a request size but a server can easily impose a sensible limit

Less easily handled: a request that comes in slowly, occupying a socket resource meanwhile

```
GET /index.html HTTP/1.1
```

Occupying a Rare Resource

The HTTP protocol does not put a limit on a request size but a server can easily impose a sensible limit

Less easily handled: a request that comes in slowly, occupying a socket resource meanwhile

```
GET /index.html HTTP/1.1
Host: www.cs.utah.edu
```

Occupying a Rare Resource

The HTTP protocol does not put a limit on a request size but a server can easily impose a sensible limit

Less easily handled: a request that comes in slowly, occupying a socket resource meanwhile

```
GET /index.html HTTP/1.1
Host: www.cs.utah.edu
Accept-Encoding: gzip,
```

Occupying a Rare Resource

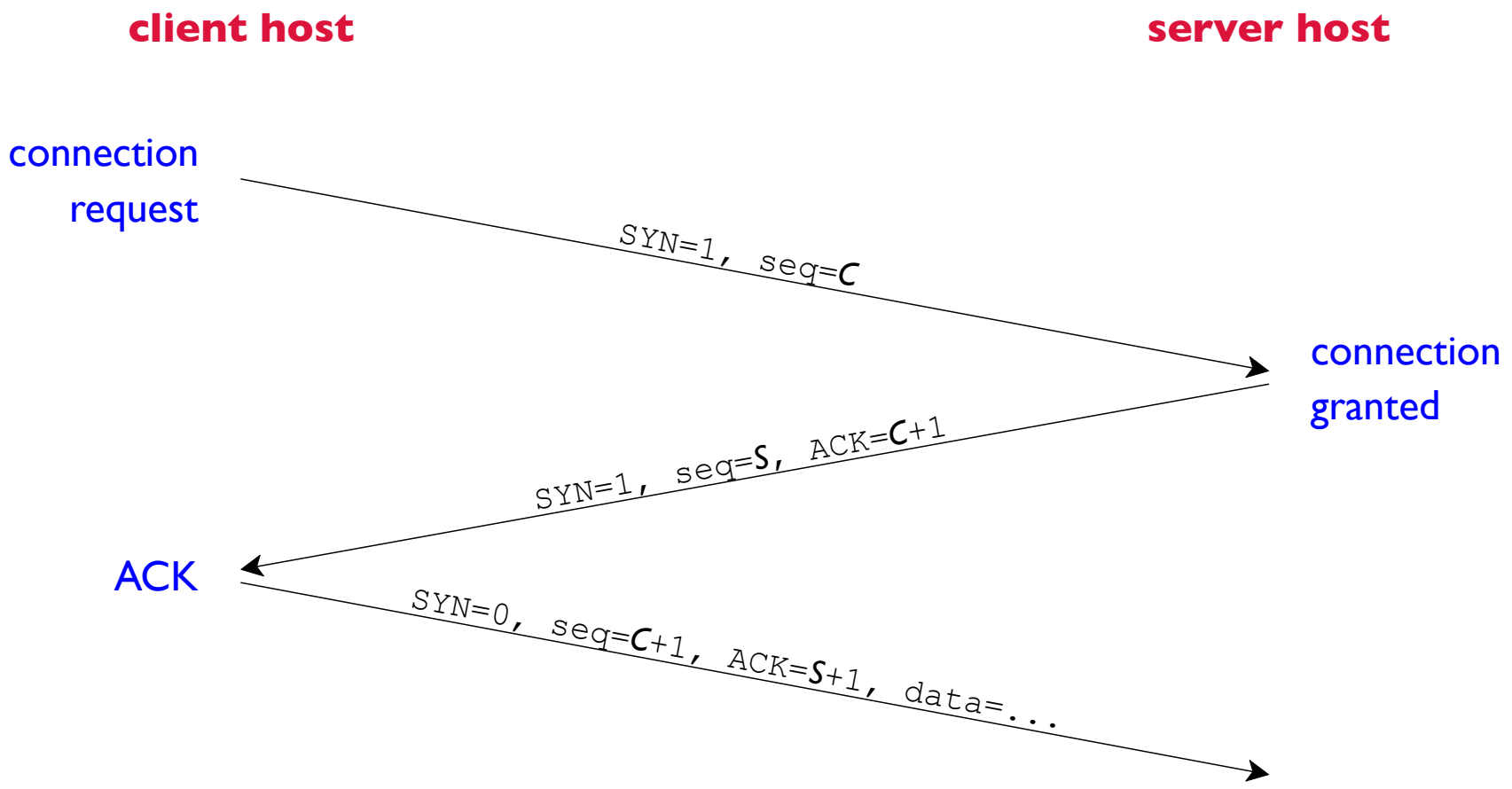
The HTTP protocol does not put a limit on a request size but a server can easily impose a sensible limit

Less easily handled: a request that comes in slowly, occupying a socket resource meanwhile

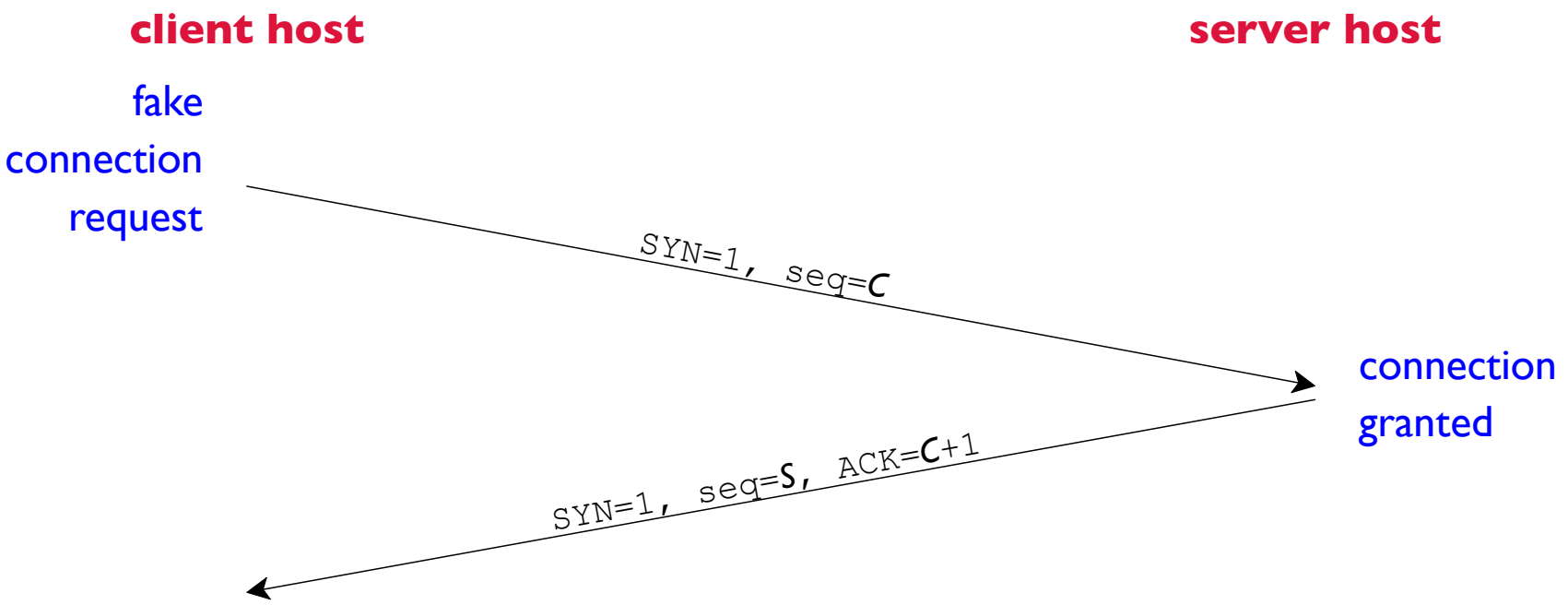
```
GET /index.html HTTP/1.1
Host: www.cs.utah.edu
Accept-Encoding: gzip, deflate
```

Slowloris tries to keep a server's connections occupied while sending it as little data as possible and as infrequently as possible

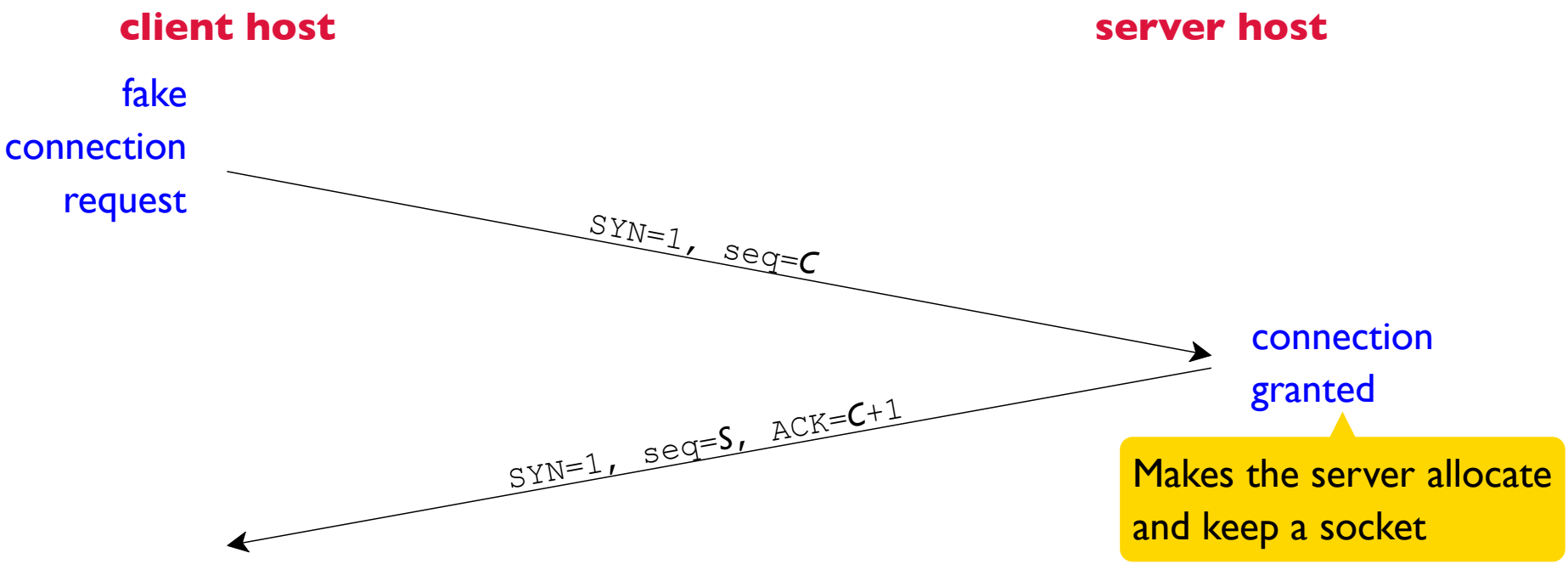
Occupying a Rare Resource: TCP SYNACK



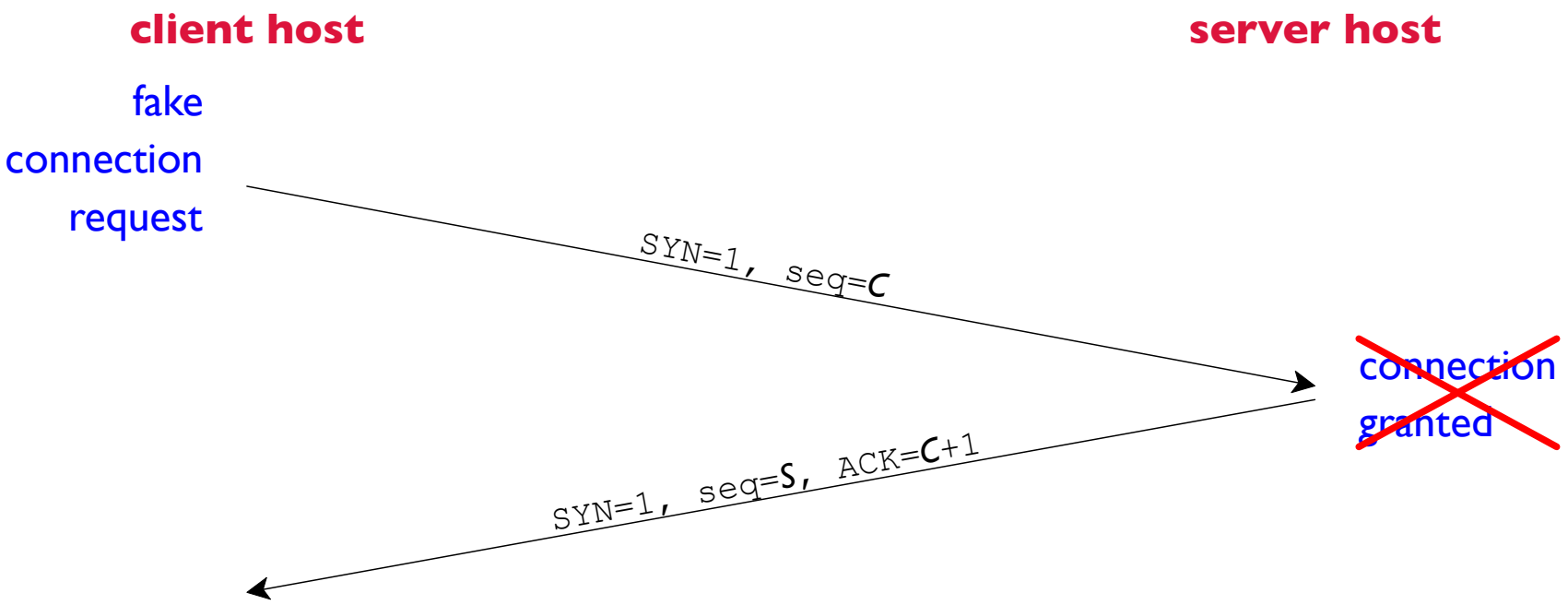
Occupying a Rare Resource: TCP SYNACK



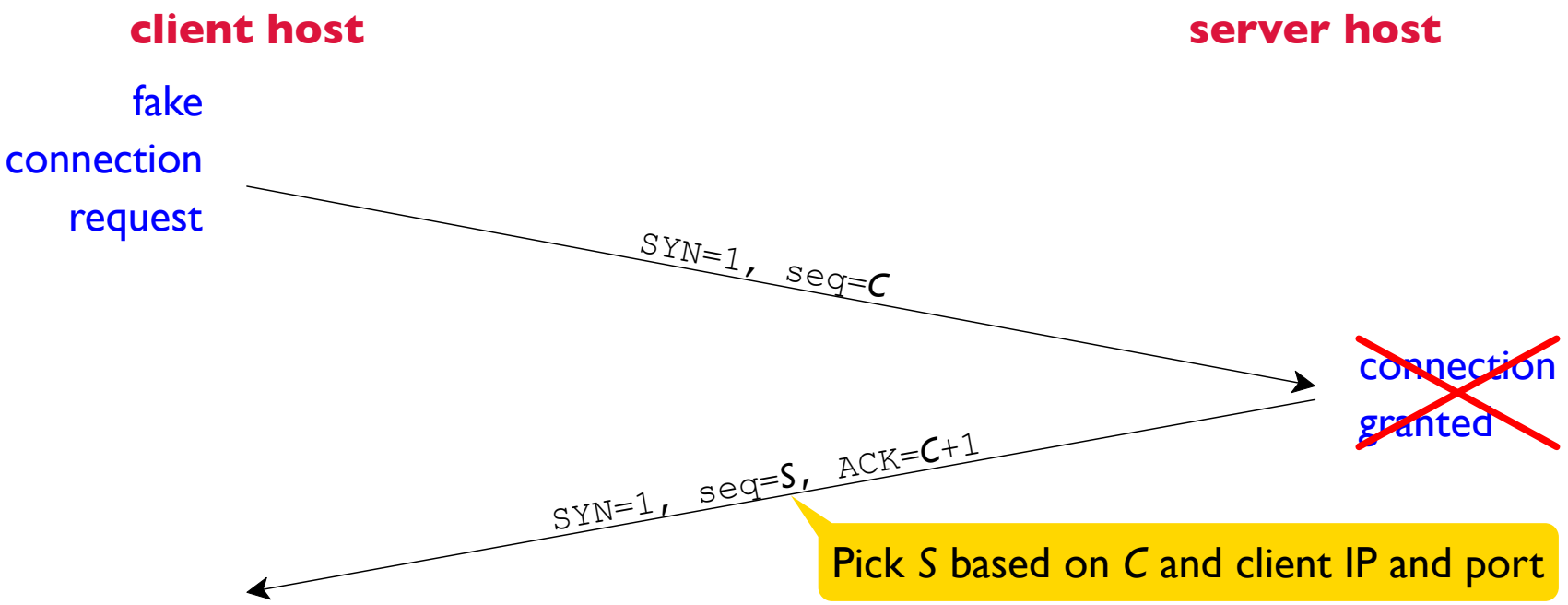
Occupying a Rare Resource: TCP SYNACK



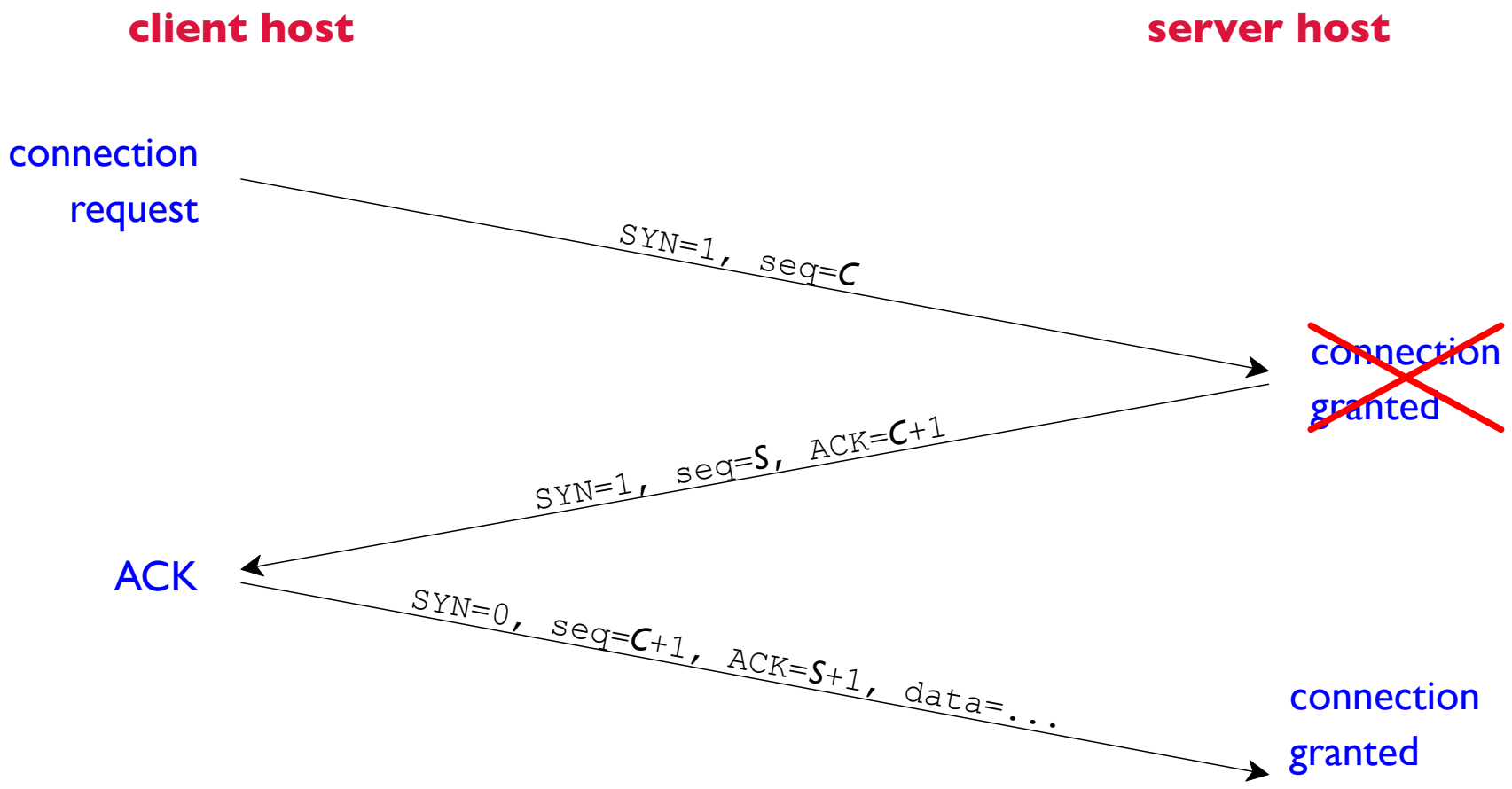
Occupying a Rare Resource: TCP SYNACK



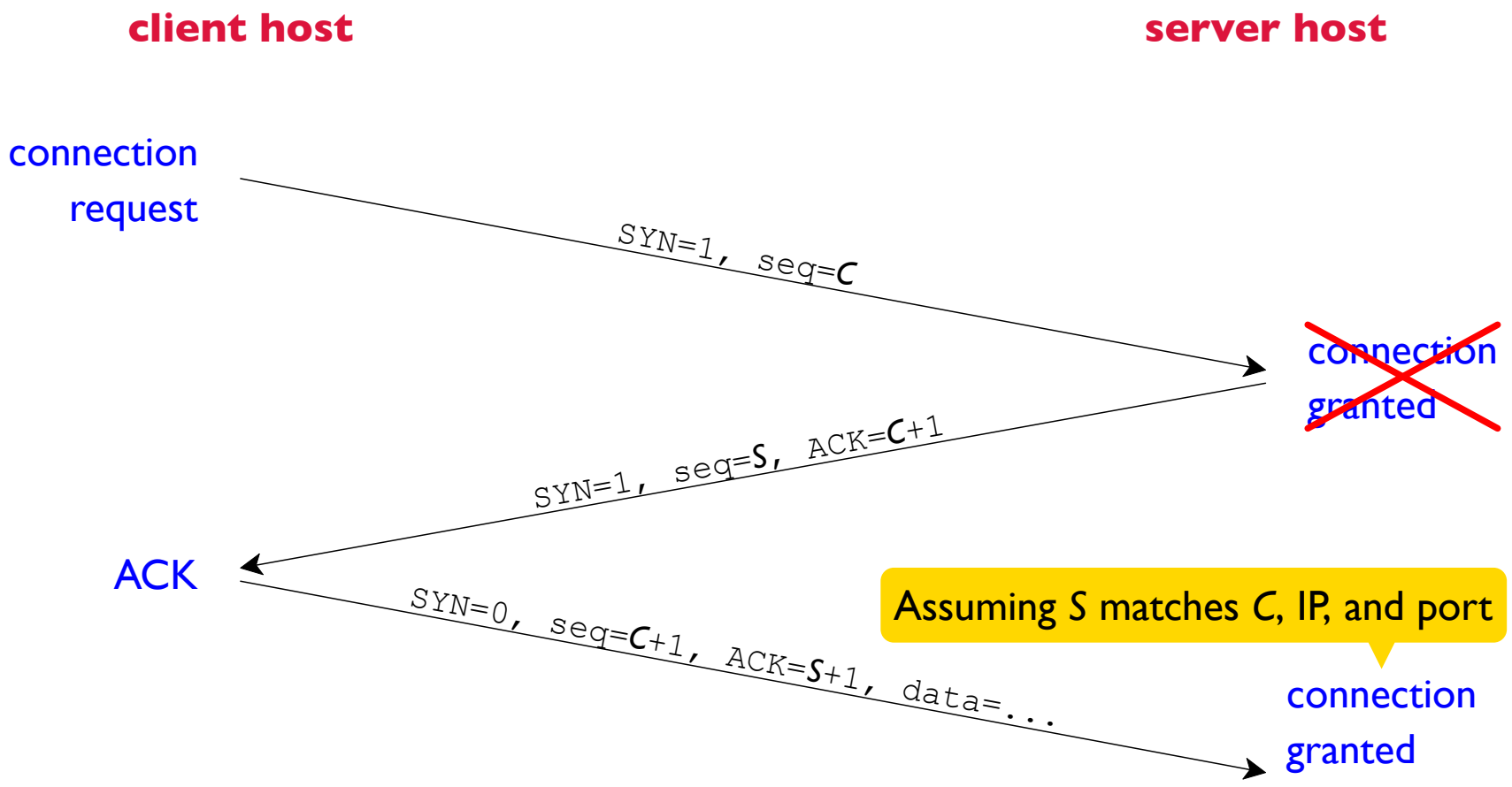
Occupying a Rare Resource: TCP SYNACK



Occupying a Rare Resource: TCP SYNACK



Occupying a Rare Resource: TCP SYNACK



Exploiting Bystanders

One problem with making a request is that you have to deal with the answer

... unless you pretend to be someone else asking

Spoofing is using someone else as the "from" in a message that you write

email address, IP address, etc.

Exploiting Bystanders

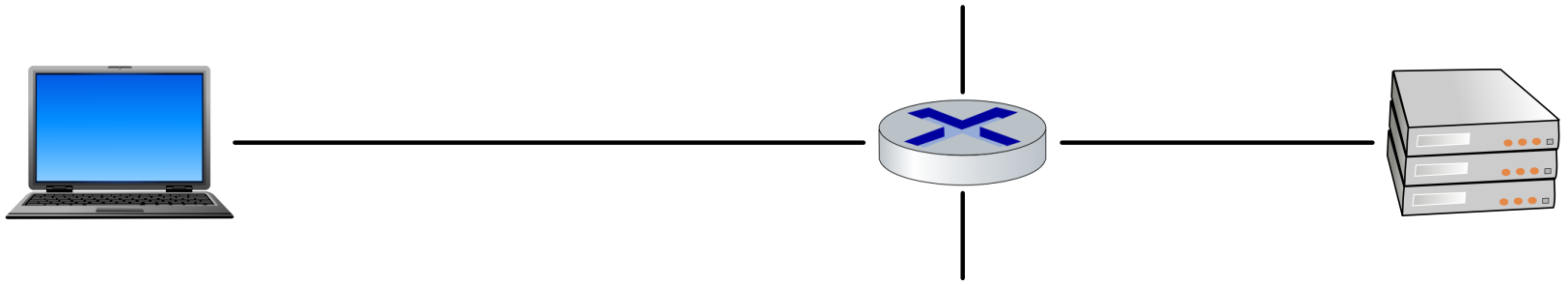
Attack a DNS server:

Send many DNS requests

result can be much larger than request

Spoof return address

results go to other parts of the network



Exploiting Bystanders

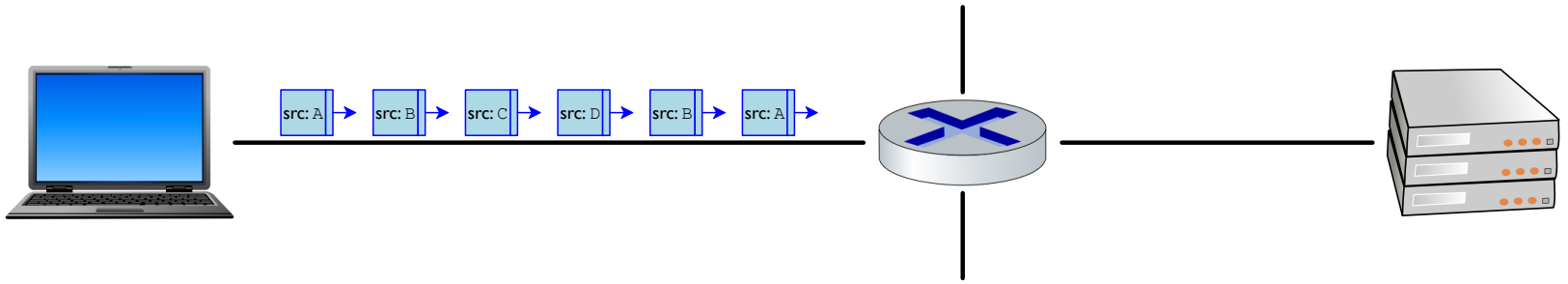
Attack a DNS server:

Send many DNS requests

result can be much larger than request

Spoof return address

results go to other parts of the network



Exploiting Bystanders

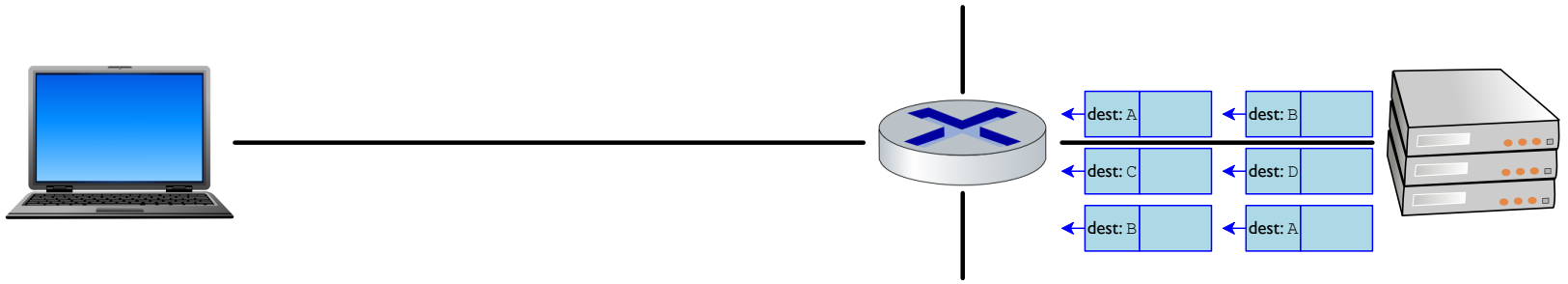
Attack a DNS server:

Send many DNS requests

result can be much larger than request

Spoof return address

results go to other parts of the network



Exploiting Bystanders

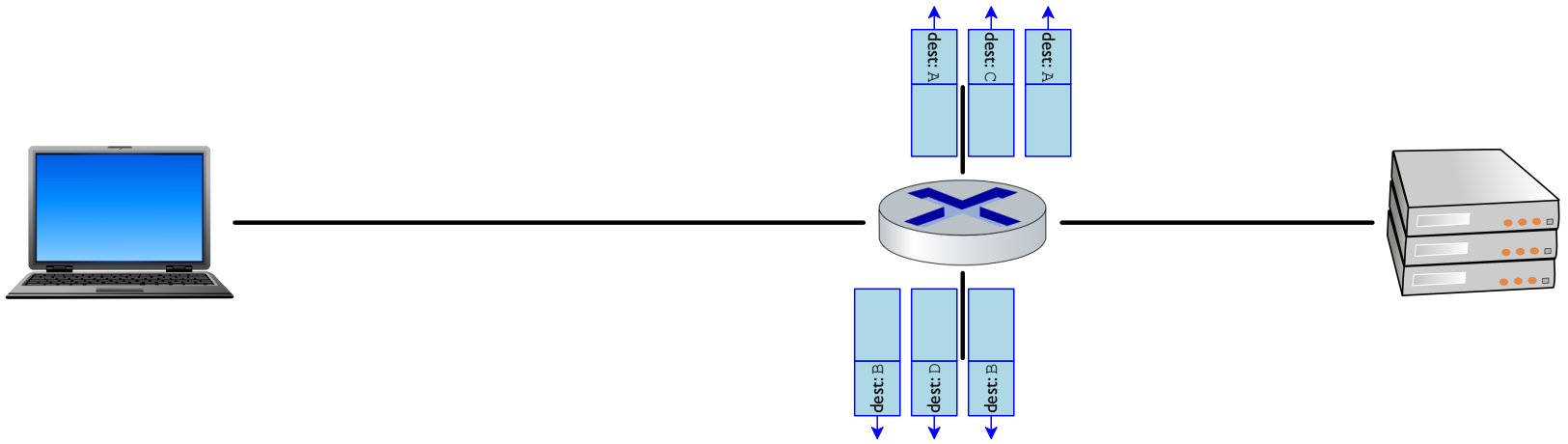
Attack a DNS server:

Send many DNS requests

result can be much larger than request

Spoof return address

results go to other parts of the network



Exploiting Bystanders

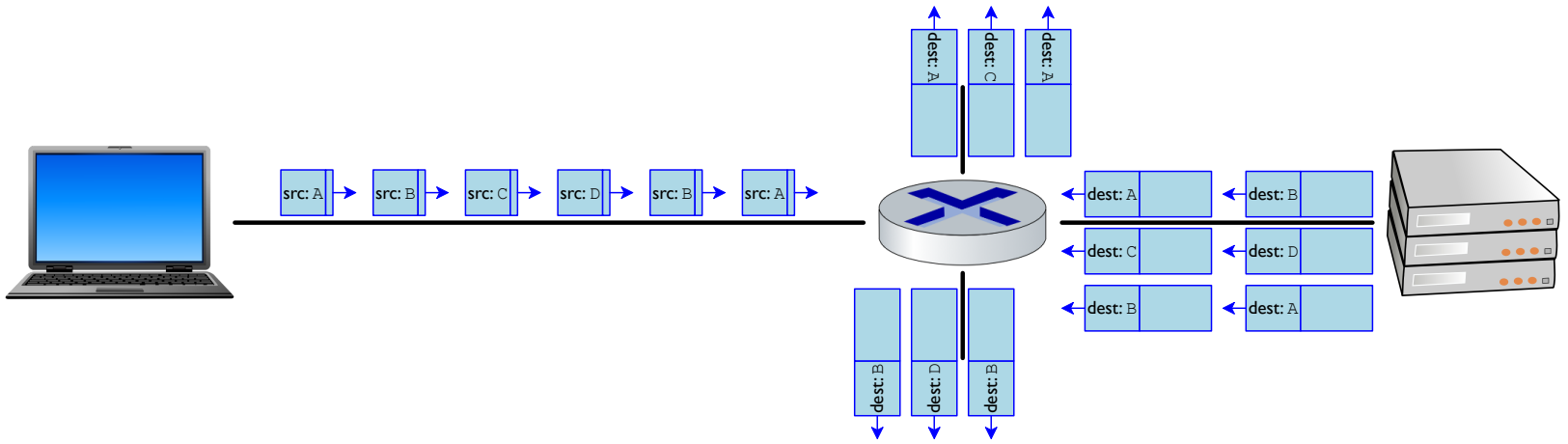
Attack a DNS server:

Send many DNS requests

result can be much larger than request

Spoof return address

results go to other parts of the network



Exploiting Bystanders

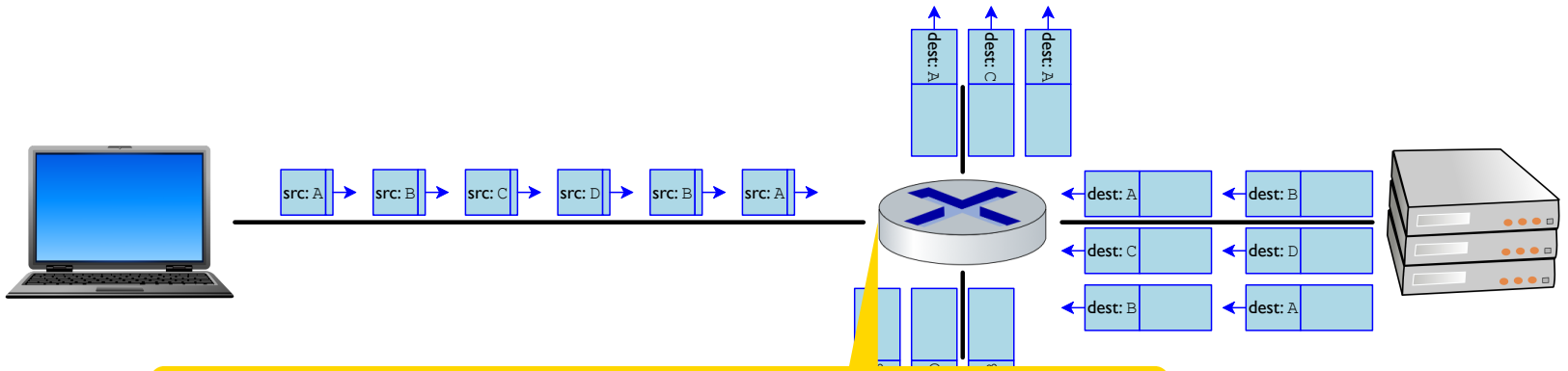
Attack a DNS server:

Send many DNS requests

result can be much larger than request

Spoof return address

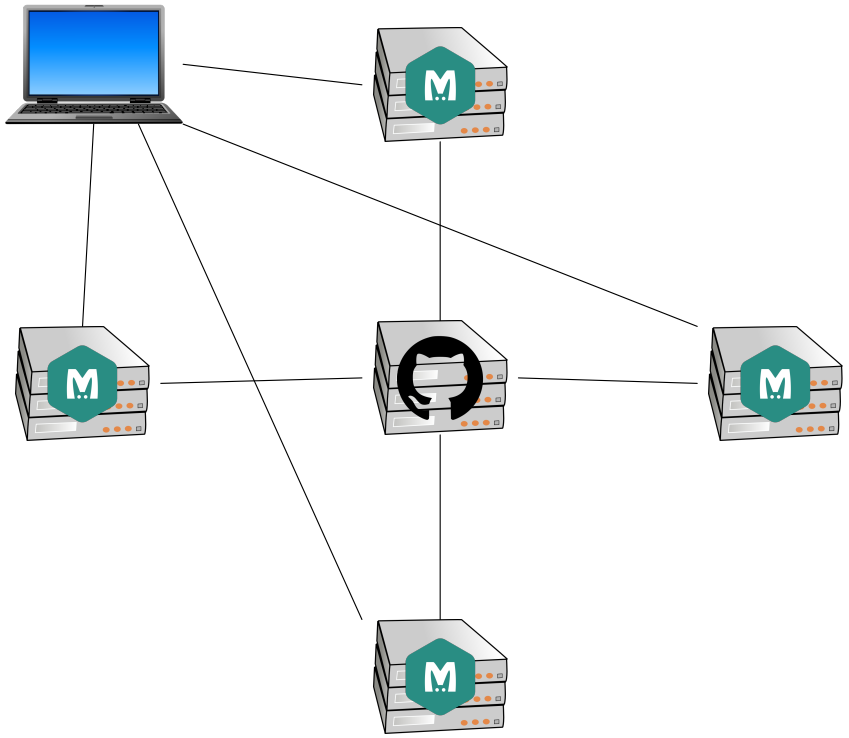
results go to other parts of the network



Possible solution: check packet src before forwarding

Exploiting Bystanders

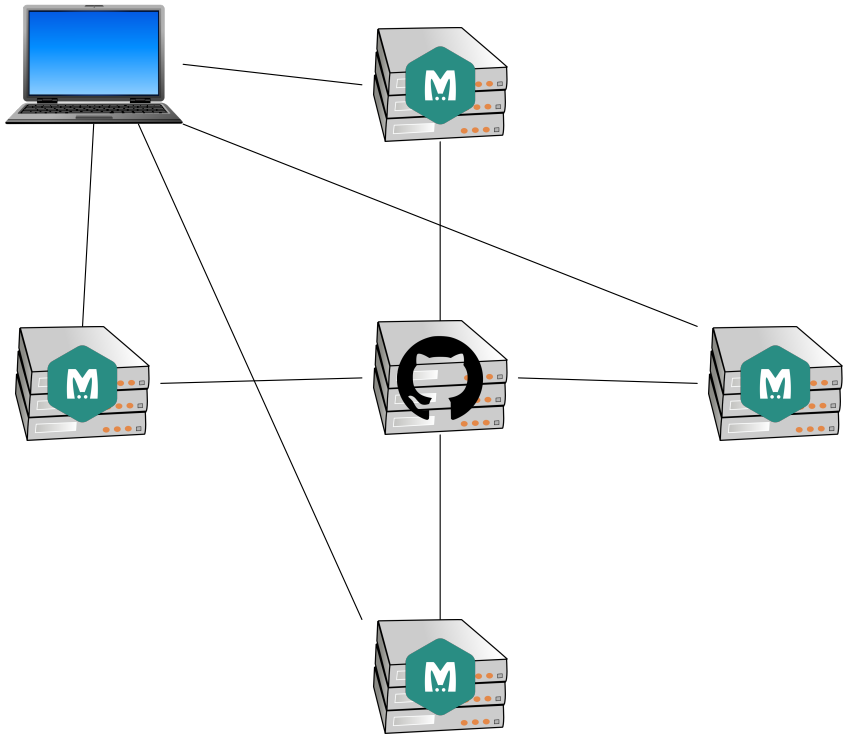
A 2018 GitHub attack used `memcached` requests,
and spoofed the request src to send all results to GitHub



Exploiting Bystanders

answer can be *much* larger than query

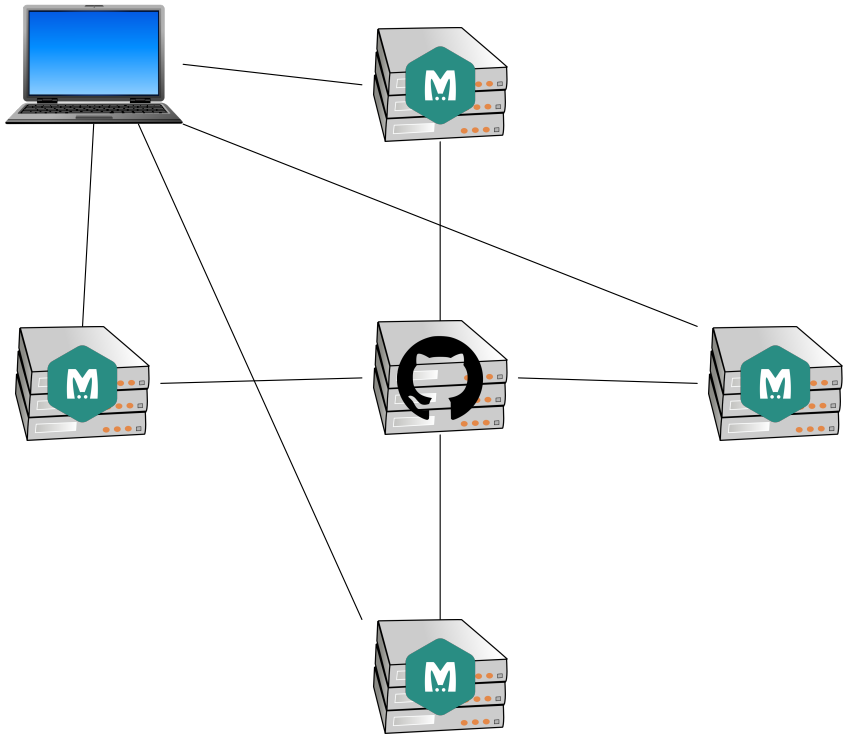
A 2018 GitHub attack used `memcached` requests,
and spoofed the request src to send all results to GitHub



Exploiting Bystanders

answer can be *much* larger than query

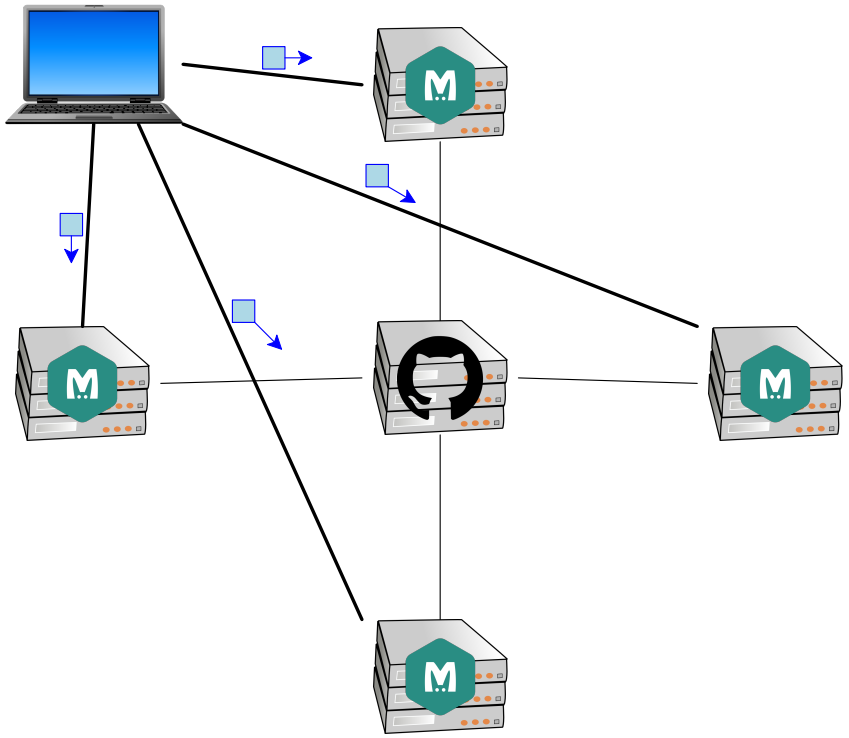
A 2018 GitHub attack used `memcached` requests,
and spoofed the request src to send all results to GitHub



Exploiting Bystanders

answer can be *much* larger than query

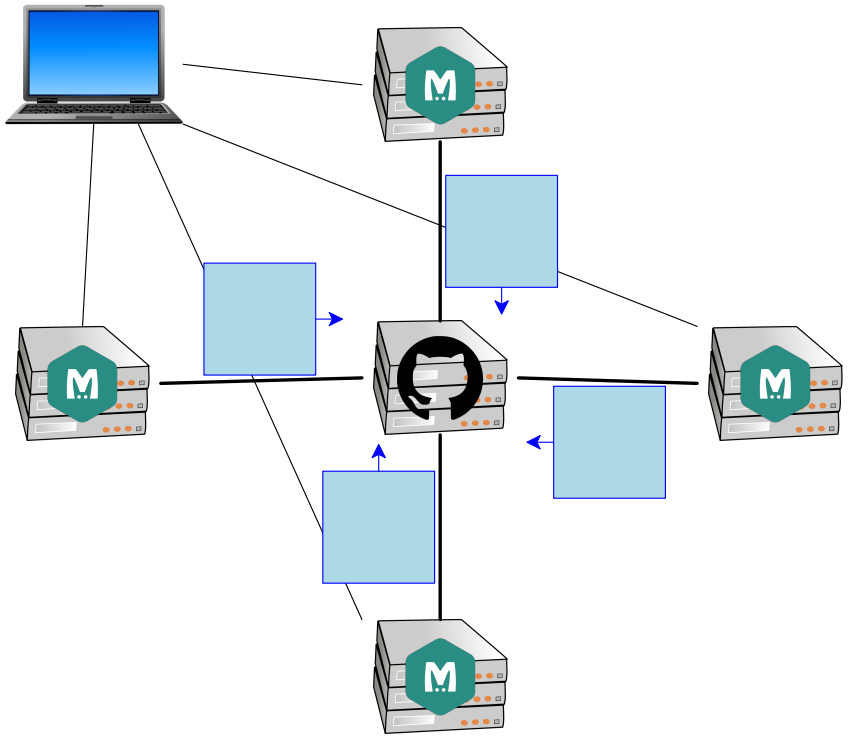
A 2018 GitHub attack used `memcached` requests,
and spoofed the request src to send all results to GitHub



Exploiting Bystanders

answer can be *much* larger than query

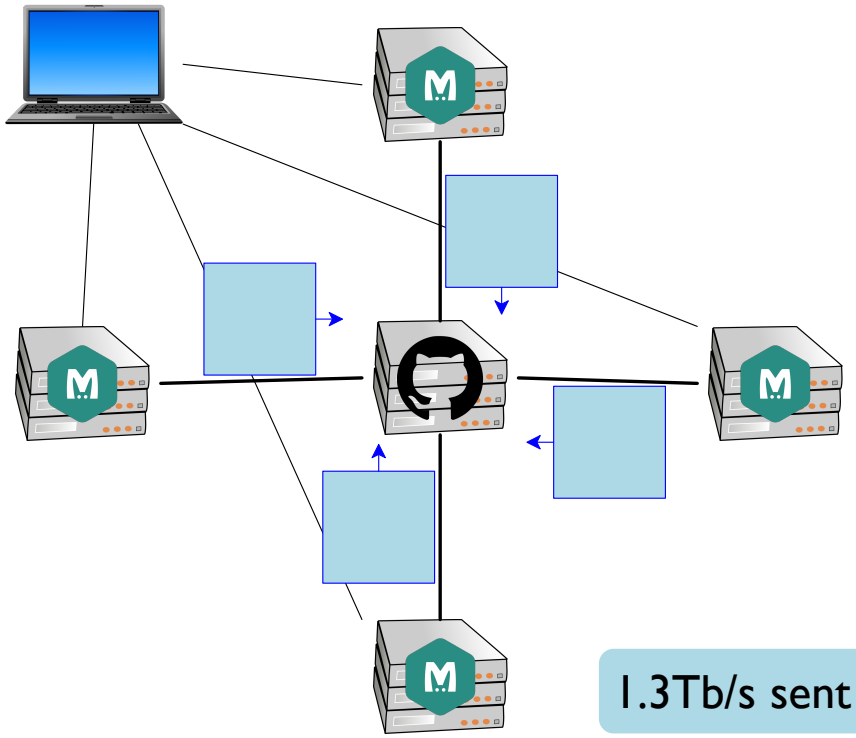
A 2018 GitHub attack used `memcached` requests,
and spoofed the request src to send all results to GitHub



Exploiting Bystanders

answer can be *much* larger than query

A 2018 GitHub attack used `memcached` requests,
and spoofed the request src to send all results to GitHub

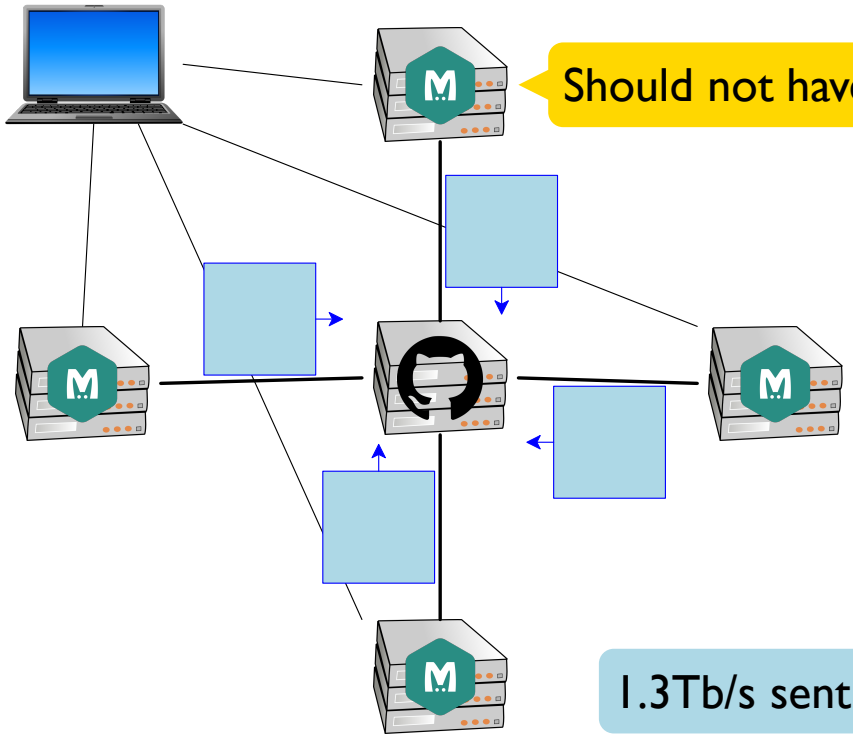


1.3Tb/s sent to GitHub for 15 minutes

Exploiting Bystanders

answer can be *much* larger than query

A 2018 GitHub attack used `memcached` requests,
and spoofed the request src to send all results to GitHub

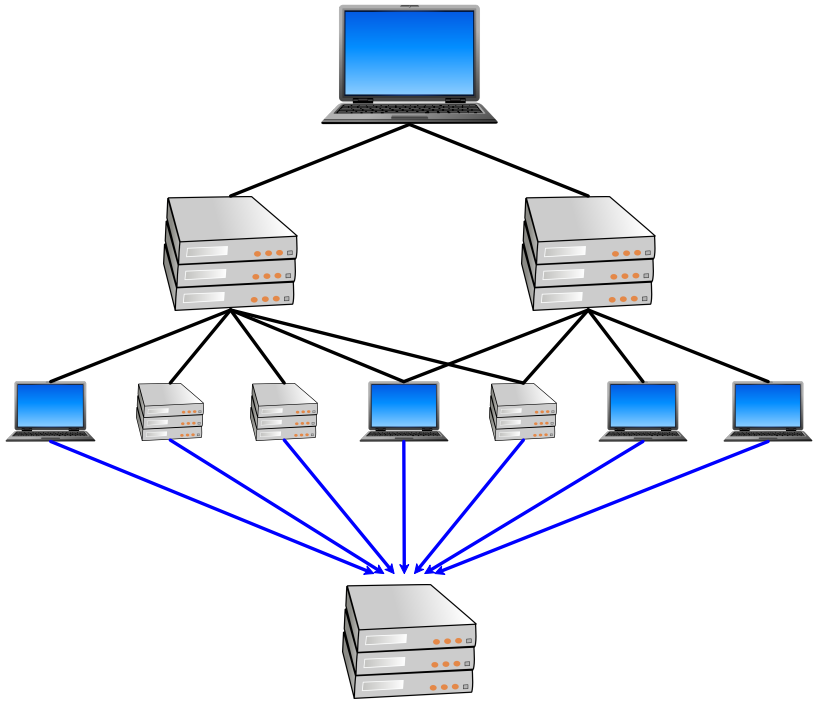


Should not have been publicly accessible

1.3Tb/s sent to GitHub for 15 minutes

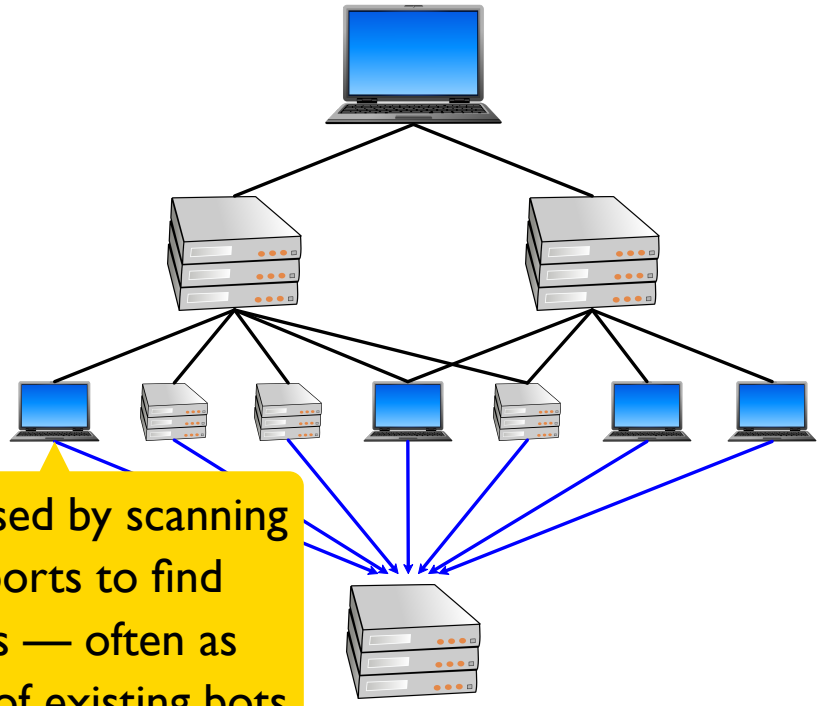
Compromised Bystanders

A **botnet** is a large collection of hosts that have been compromised and made to run external code



Compromised Bystanders

A **botnet** is a large collection of hosts that have been compromised and made to run external code



botmaster

command and control servers

rootkit-owned bots

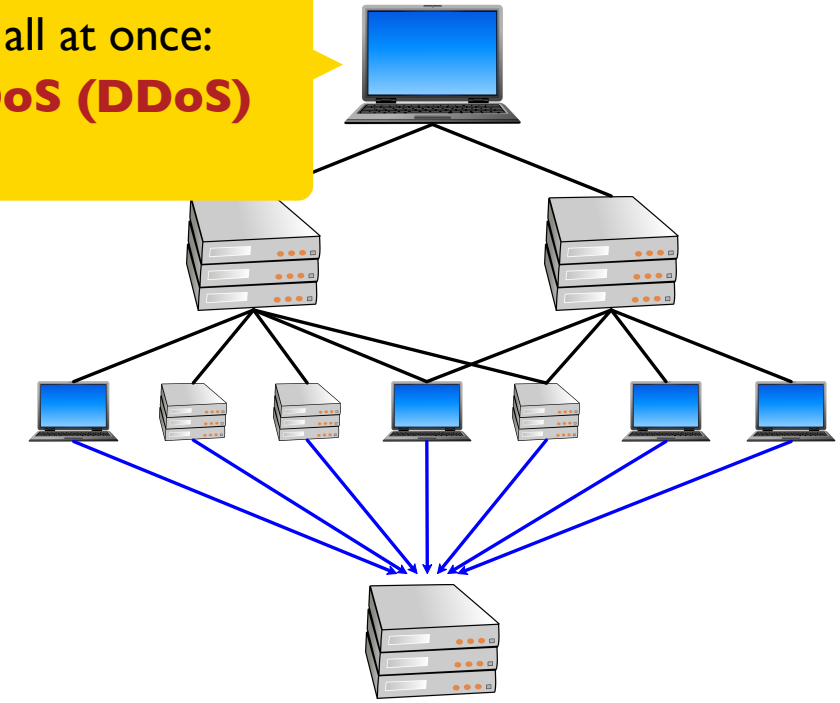
victim

Gradually amassed by scanning addresses and ports to find vulnerable hosts — often as a periodic task of existing bots

Compromised Bystanders

A **botnet** is a large collection of hosts that have been compromised and
l code

Can direct bots to send messages to the victim host all at once: a **distributed DoS (DDoS)** attack



botmaster

command and control servers

rootkit-owned bots

victim

Compromised Bystanders

Early botnet days: many home computers suddenly on the internet with poor security configurations

Newer home OSes tend to be configured and maintained more defensively

Modern bots: **internet of things (IoT)** devices

cameras, refrigerators, routers, etc.

IoT devices tend to run stripped-down Linux with a default password

Mirai Botnets

Since 2016, botnets based on **Mirai** malware have launched DDos attacks and otherwise exploited (e.g., cryto mining) IoT devices

Try a web search on "mirai botnet news"

Bot-recruiting approach:

- scan addresses and ports by sending TCP SYN probes
- try logging into discovered hosts using a directory of default passwords
- clean out other malware(!)
- periodically scan peers to reinfect them in case of reboots

Defenses

Primary tool against botnets is traffic classification to detect and reject malicious packets

This is difficult, because botnet traffic by design looks like normal traffic

Other strategies:

- timeouts to prevent SYN flooding and Slowloris
- **rate-limiting** to prevent a client from making too many requests
- traffic limiting through **firewalls**
- anti-spoofing routing

Summary

A **denial of service (DoS)** attack prevents a server from doing its job

A **distributed denial of service (DDoS)** attack solves the problem of scaling requests to exceed a server's capacity

A **botnet** is a way to scale and distribute an attack without having to pay for it