Internet Nails

Marcus J. Ranum
Chief of Security
mjr@tenablesecurity.com

For want of a nail….

…the shoe was lost

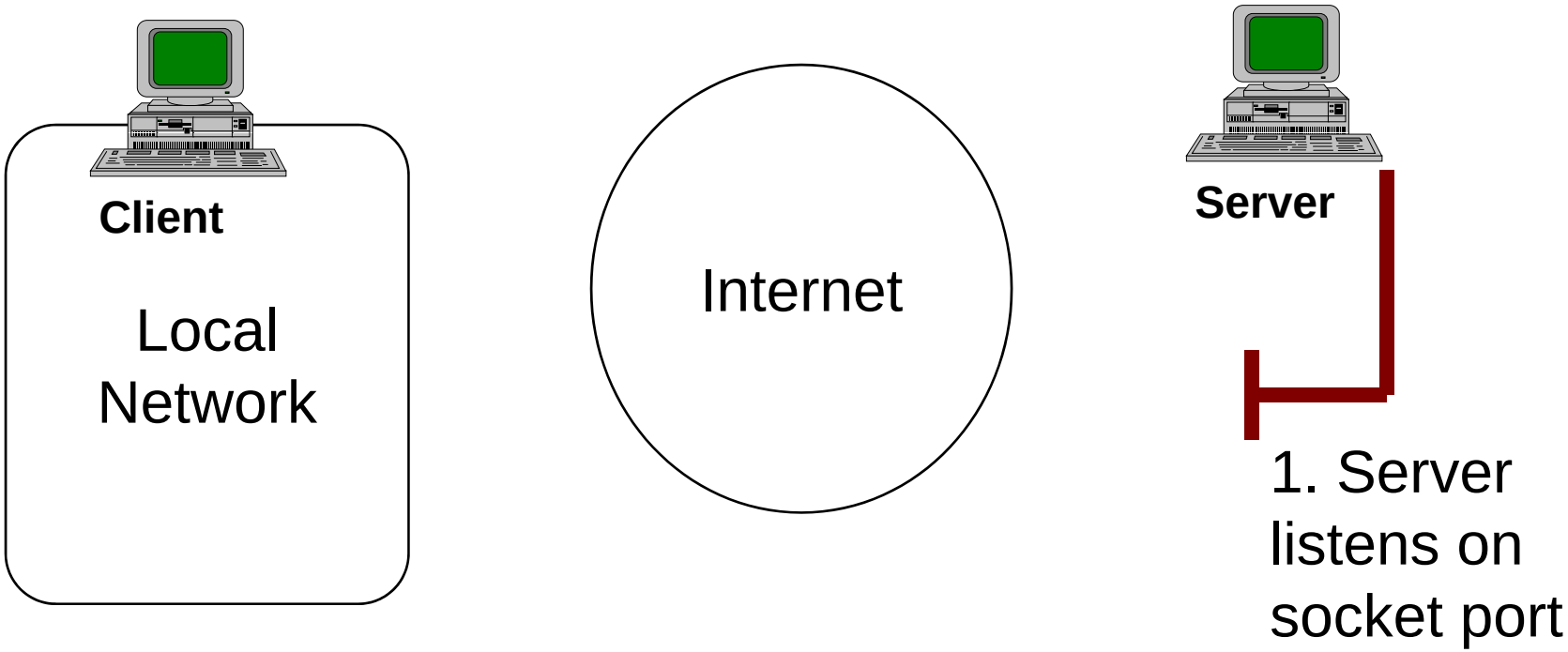
- For want of a shoe the horse was lost
  - For want of a horse, the knight was lost
    - For want of a knight the battle was lost

- I want to illustrate how software's tendency to *accrete* can have interesting side-effects
  - Can cost ***huge*** amounts of time and money
  - Potentially become the "achilles' heel" of entire culture?

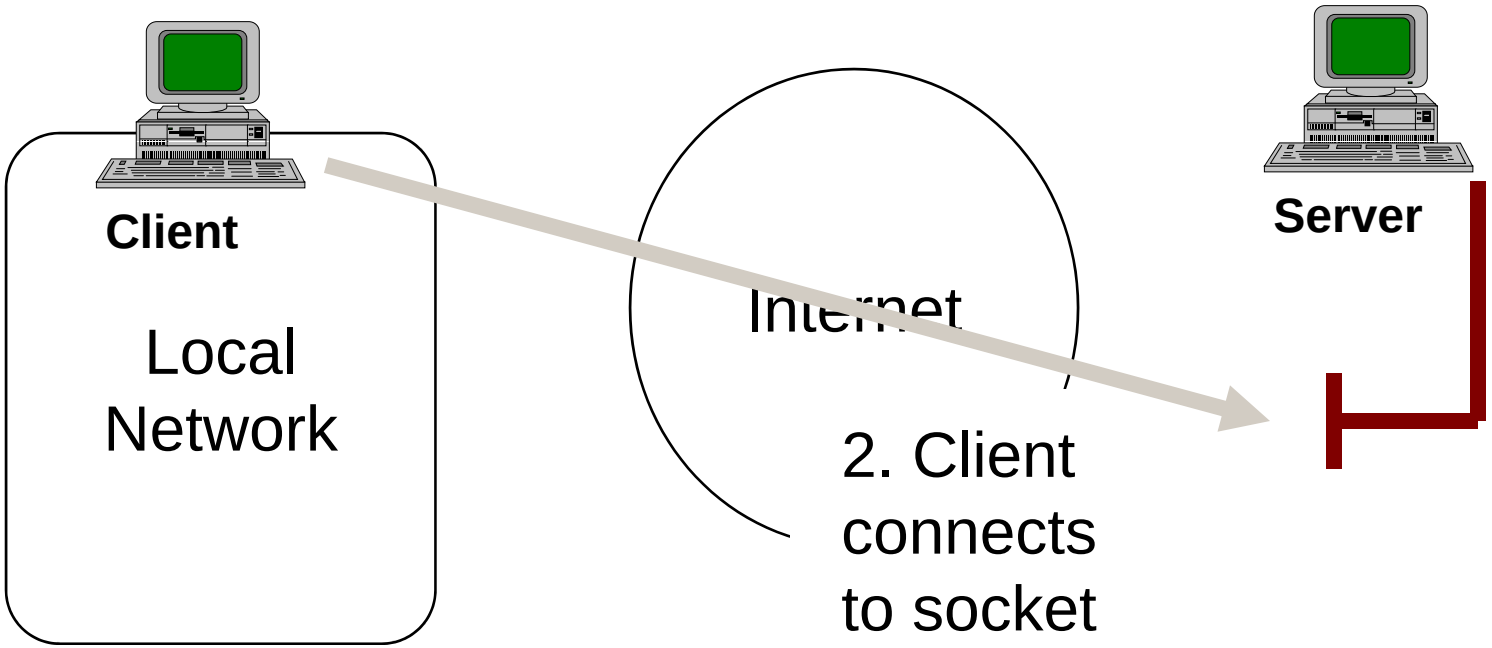
In the beginning there was…

- Circa 1971 - FTP - used for file transfer between early ARPANet nodes
  - Early versions of Email were carried over FTP
- Circa 1976 - host-to-host protocol / NCP
  - The protocol before what became TCP (of TCP/IP) the network layer that drives today's Internet

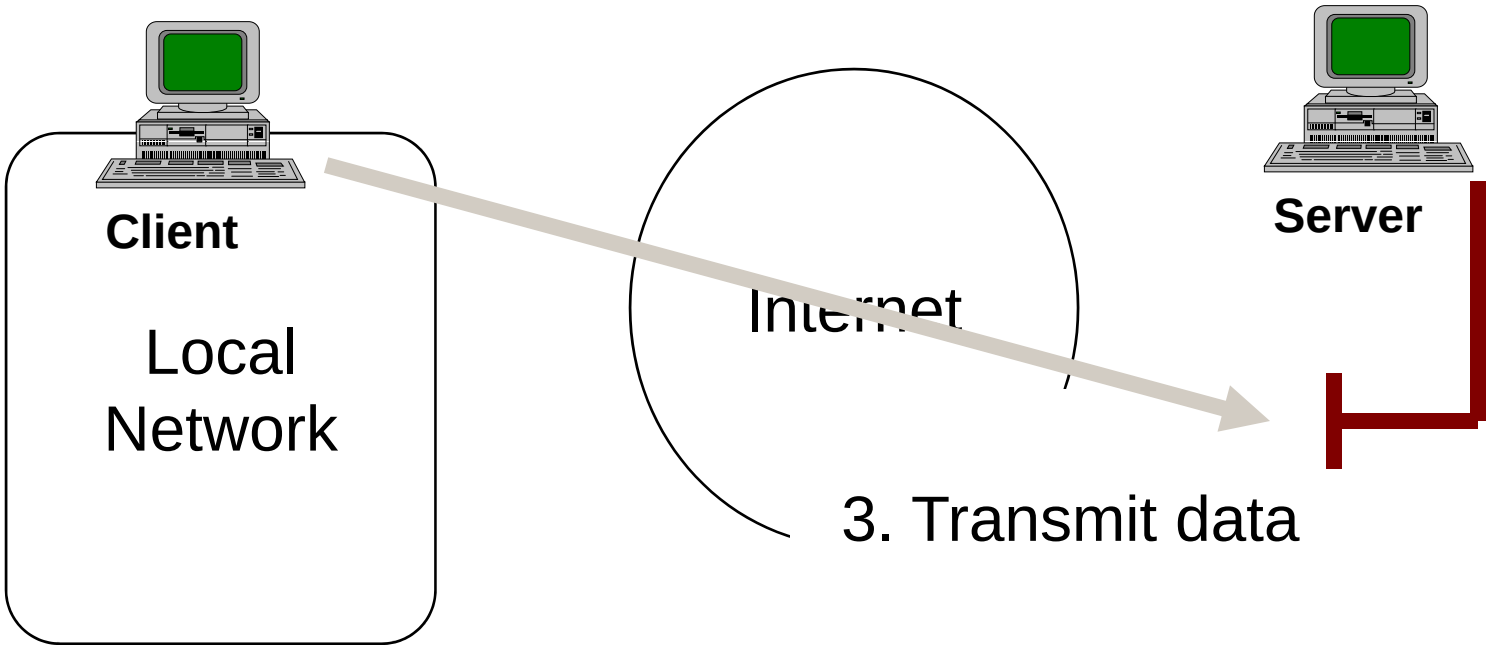
Network Sockets



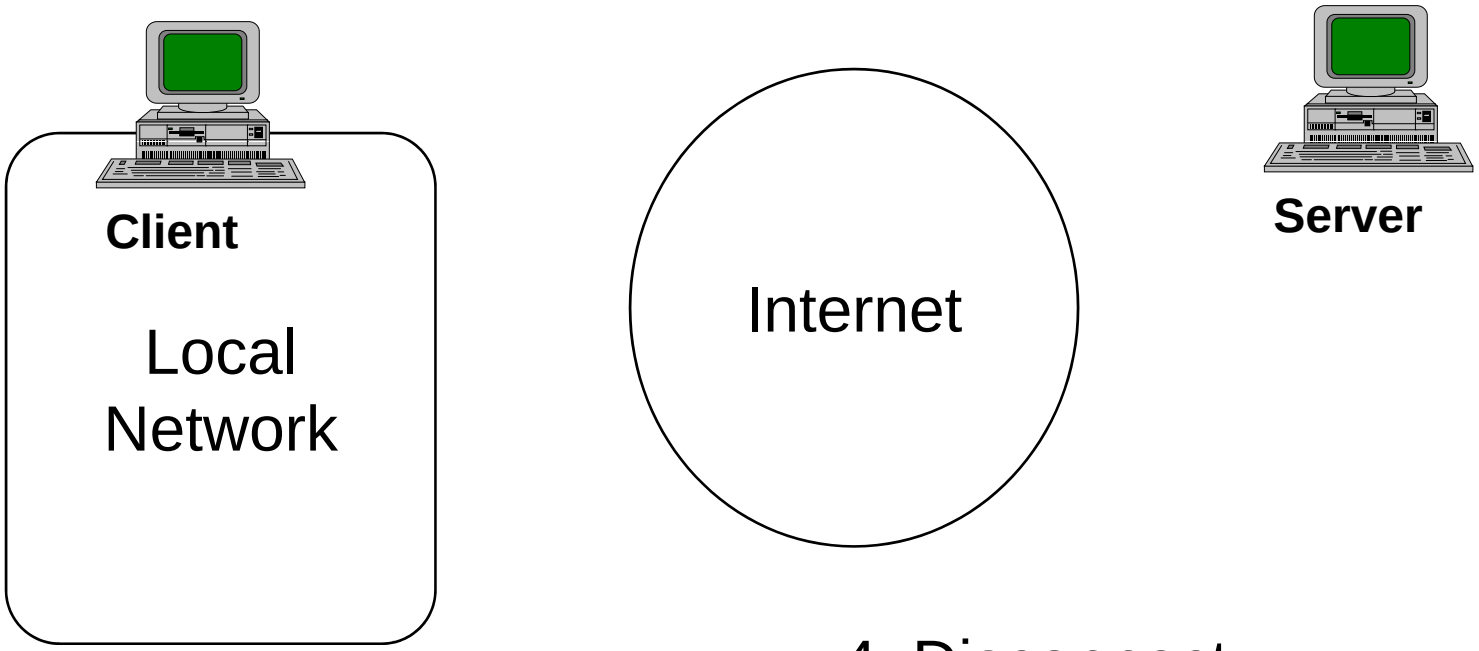
Network Sockets



Network Sockets

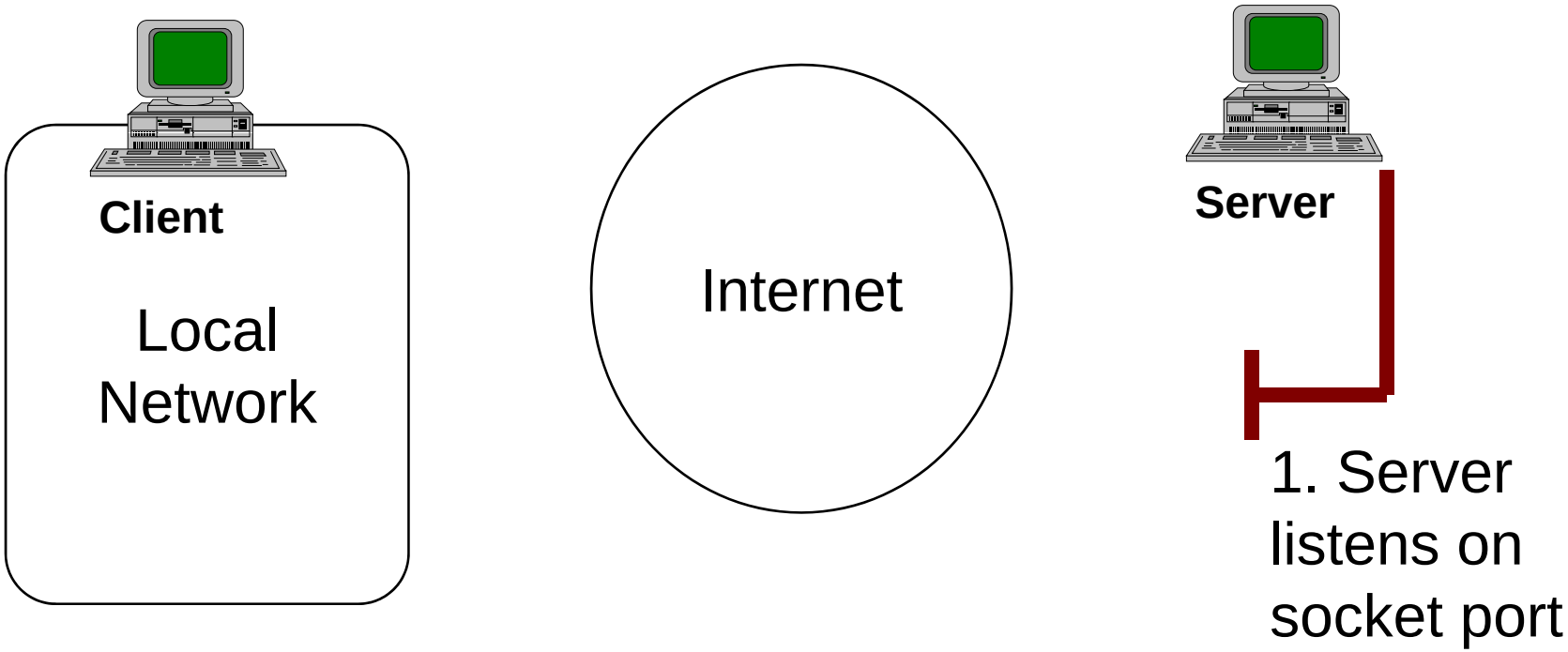


Network Sockets

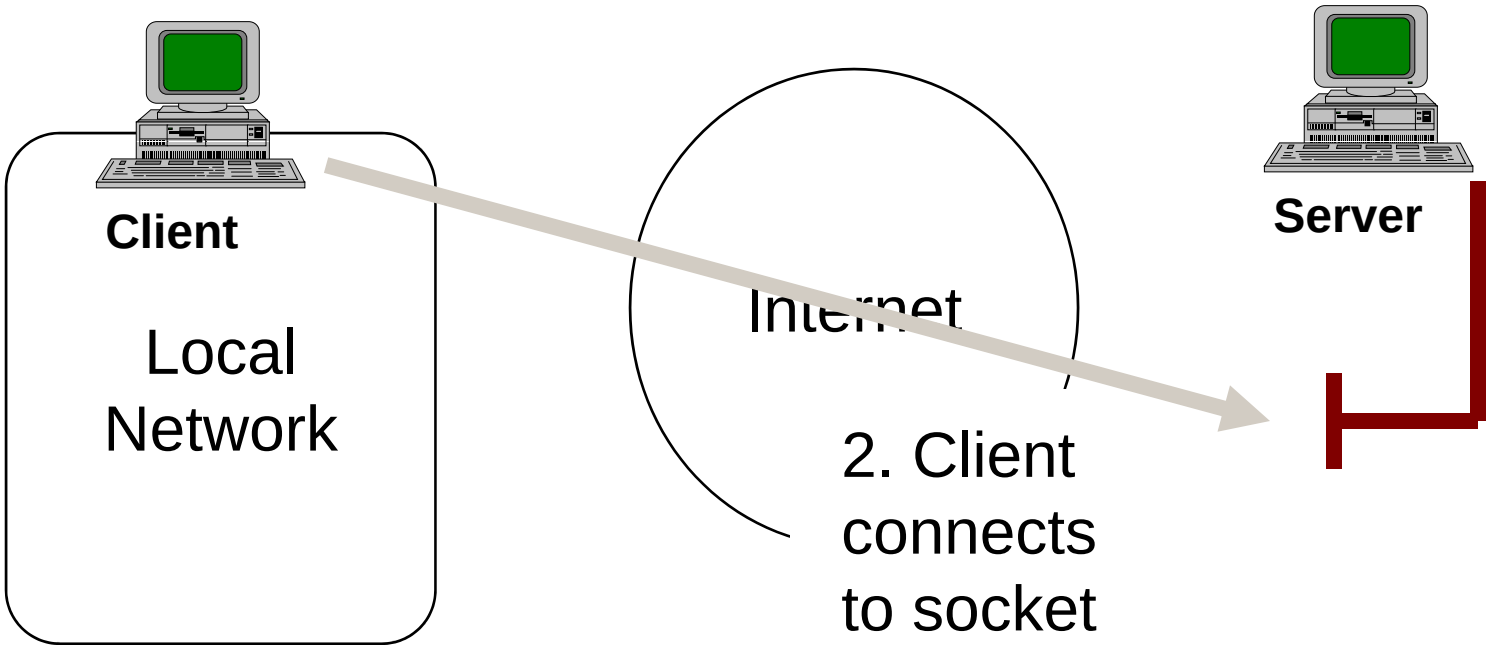


4. Disconnect

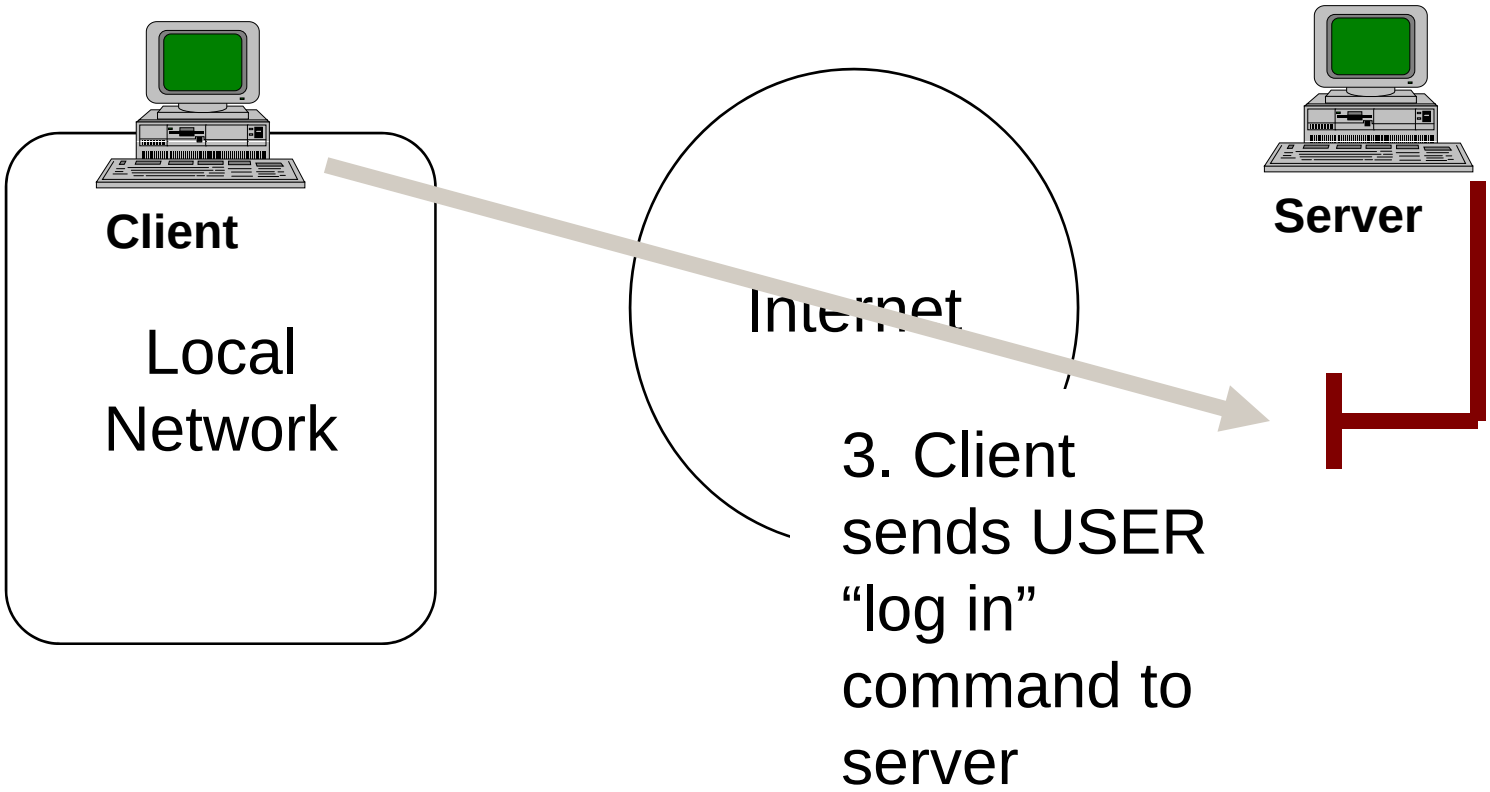
FTP: How it Works



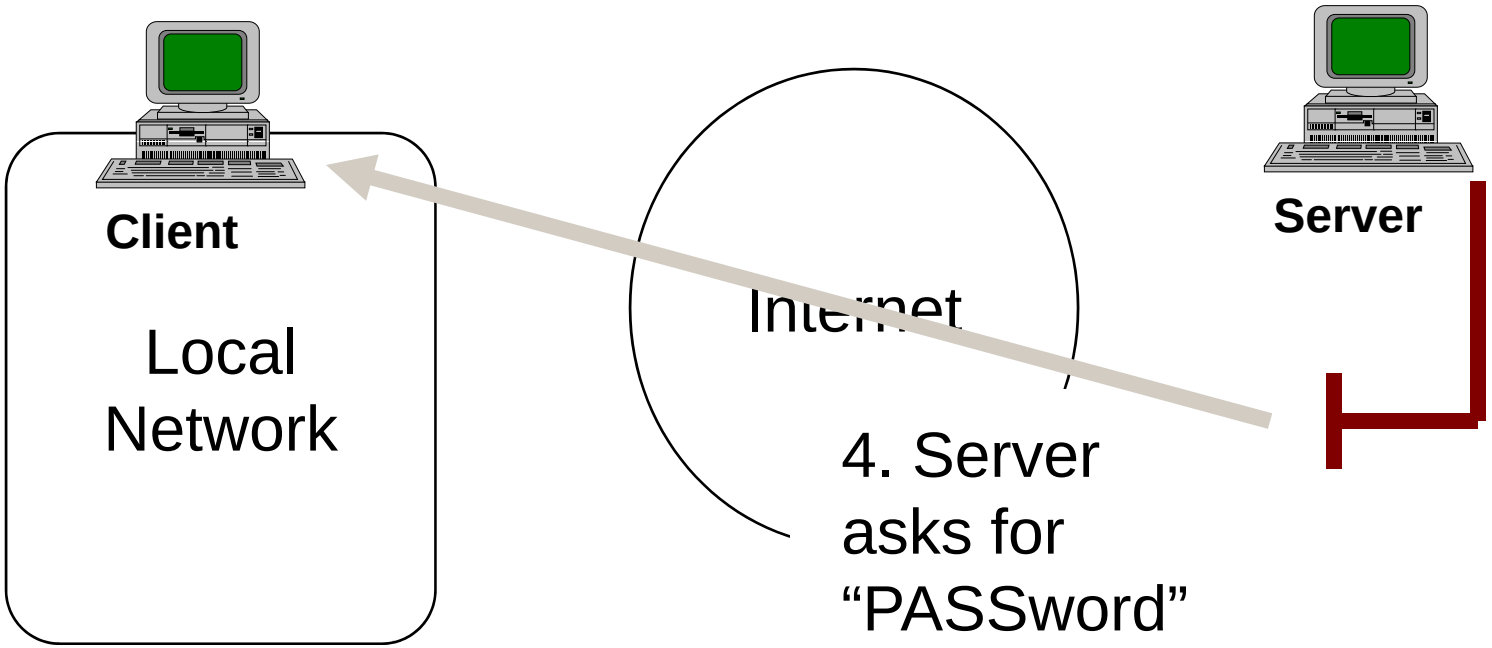
FTP: How it Works



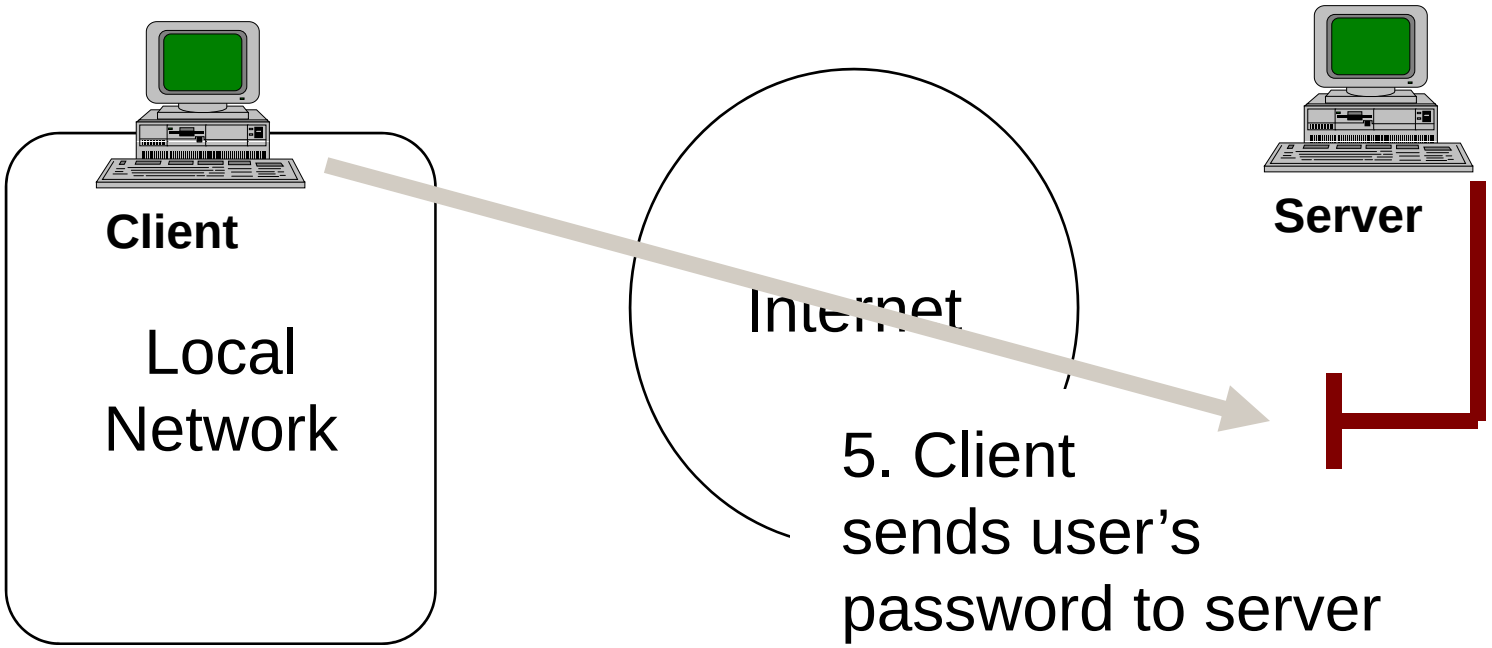
FTP: How it Works



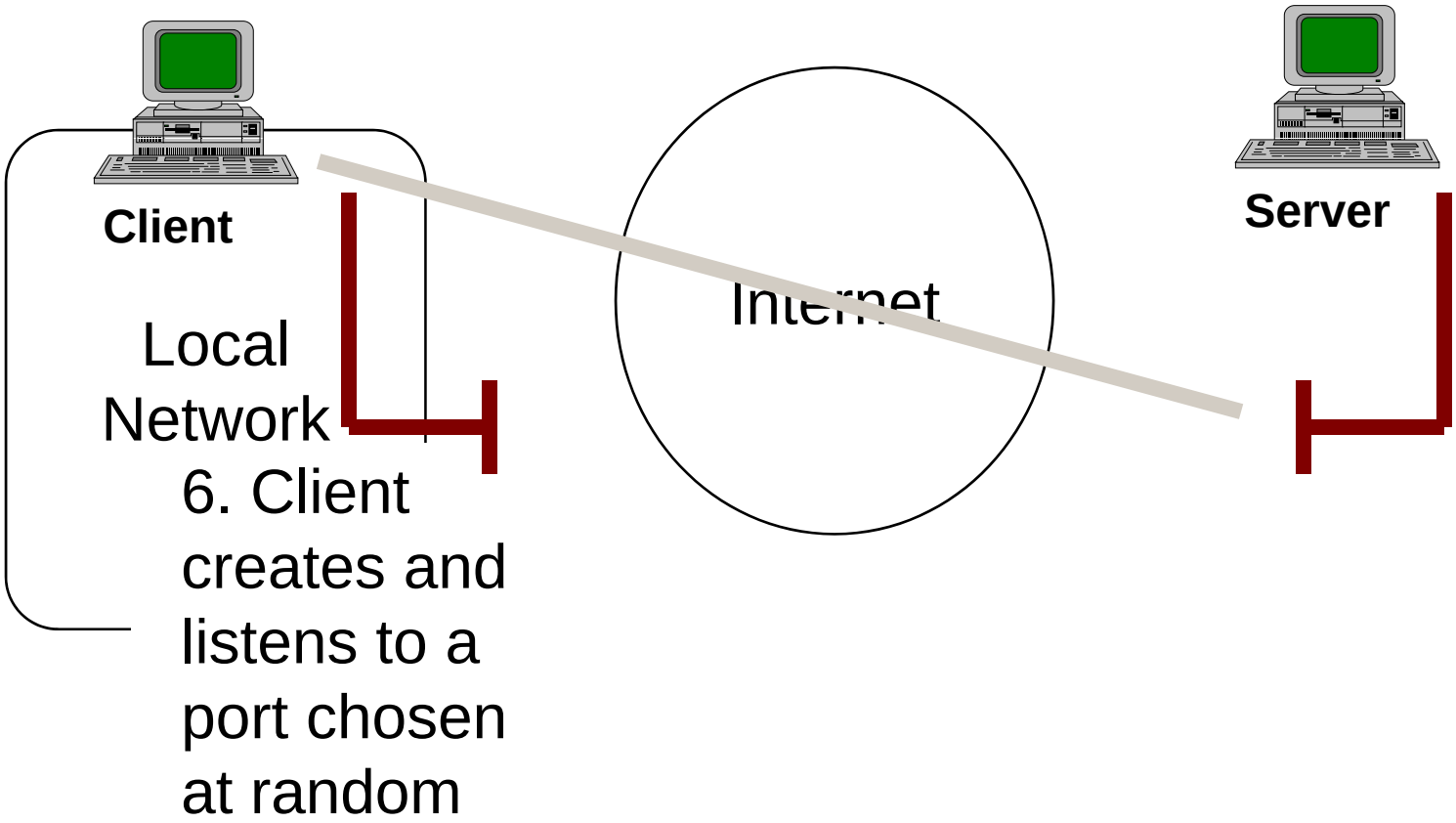
FTP: How it Works



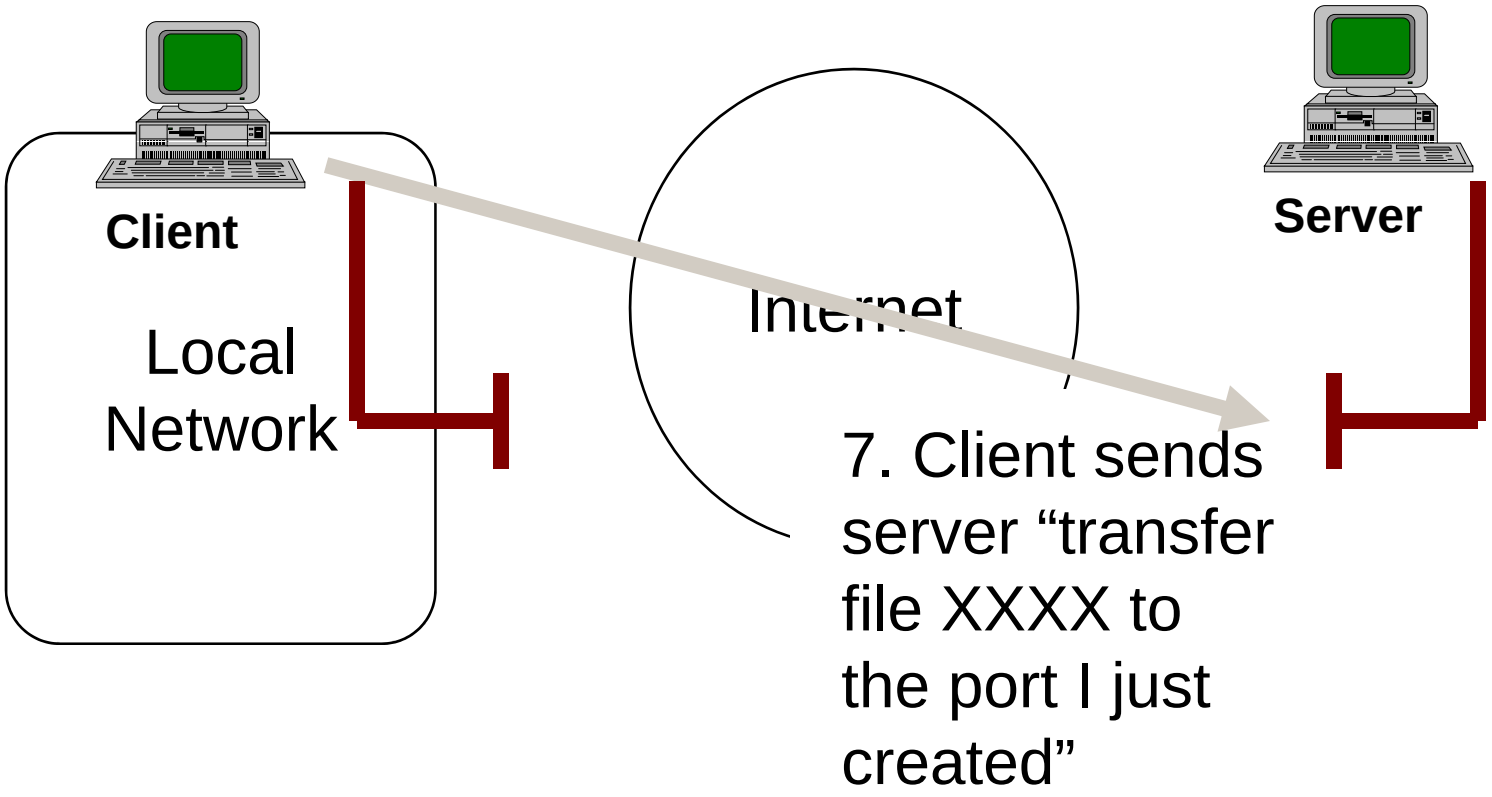
FTP: How it Works



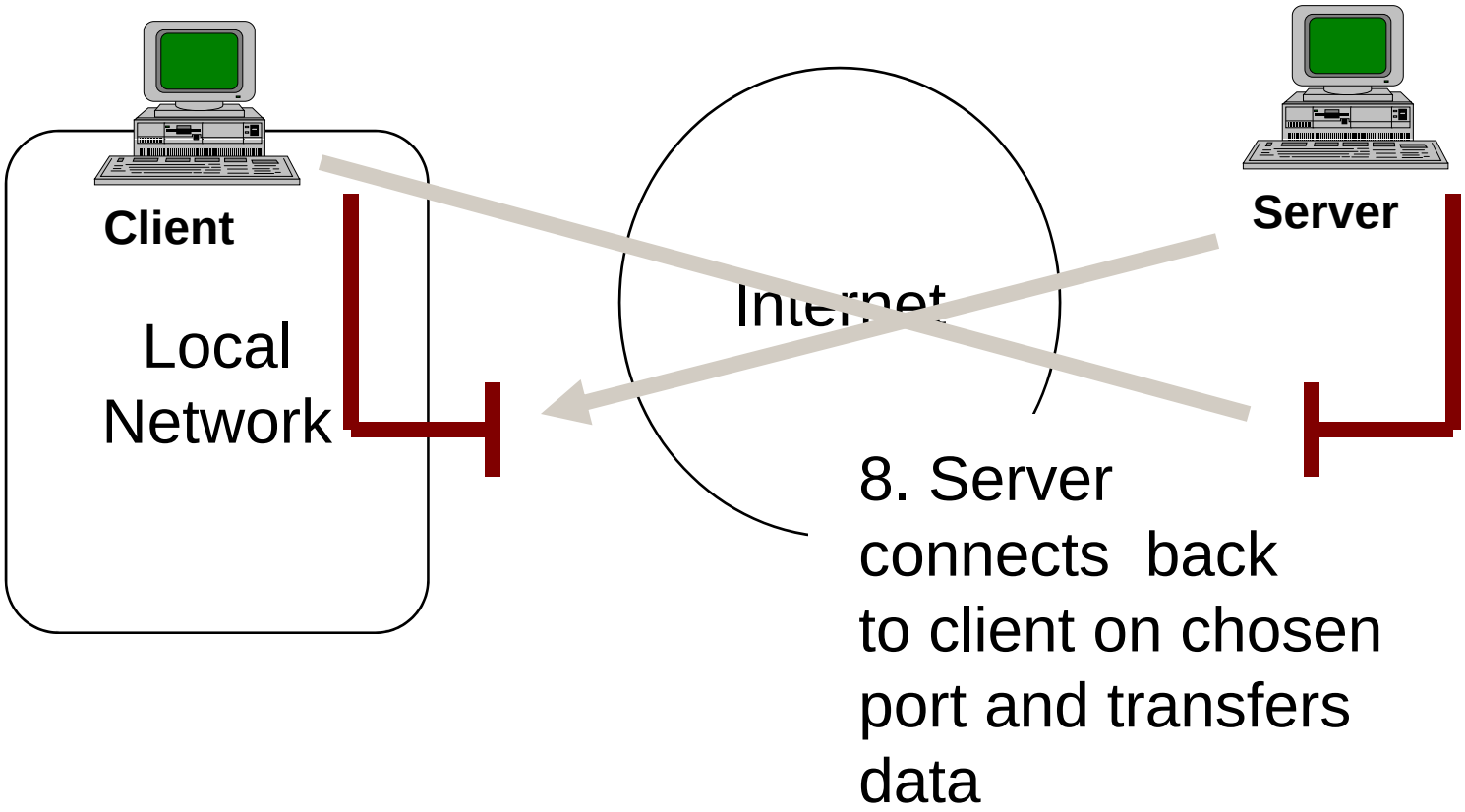
FTP: How it Works



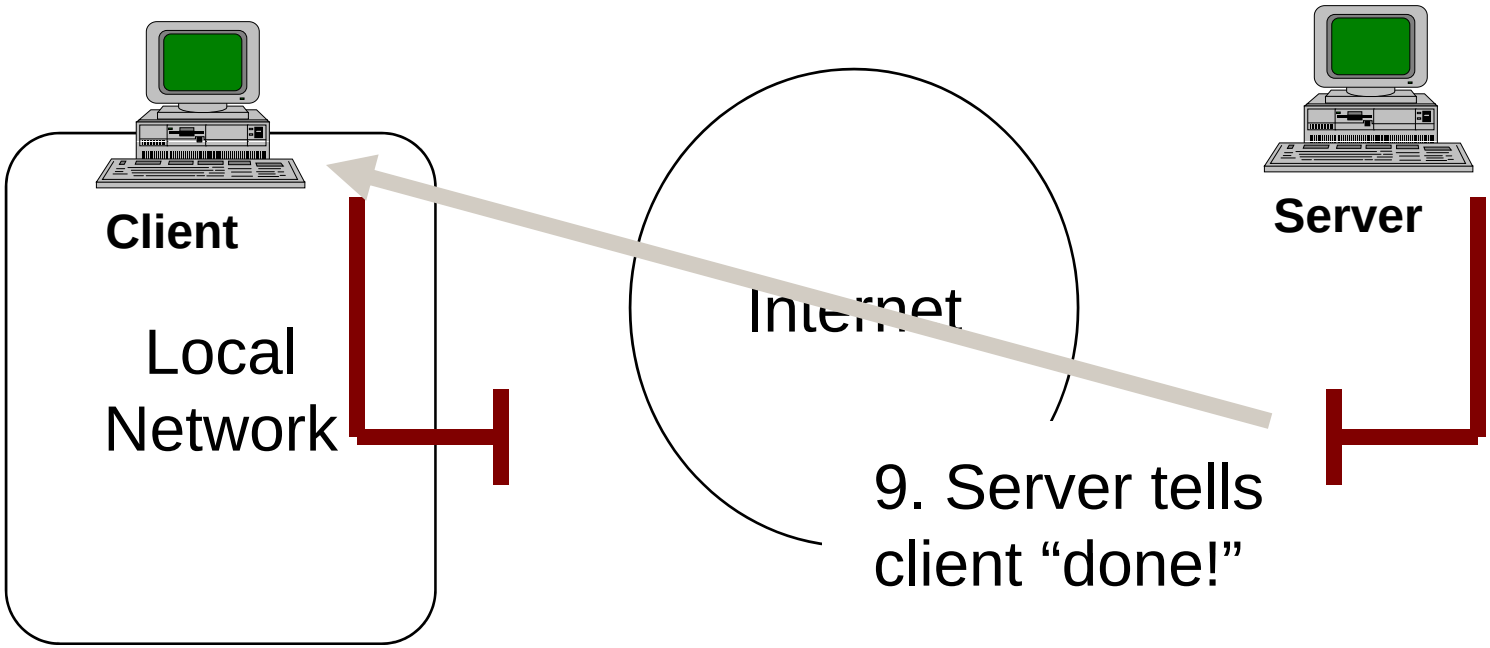
FTP: How it Works



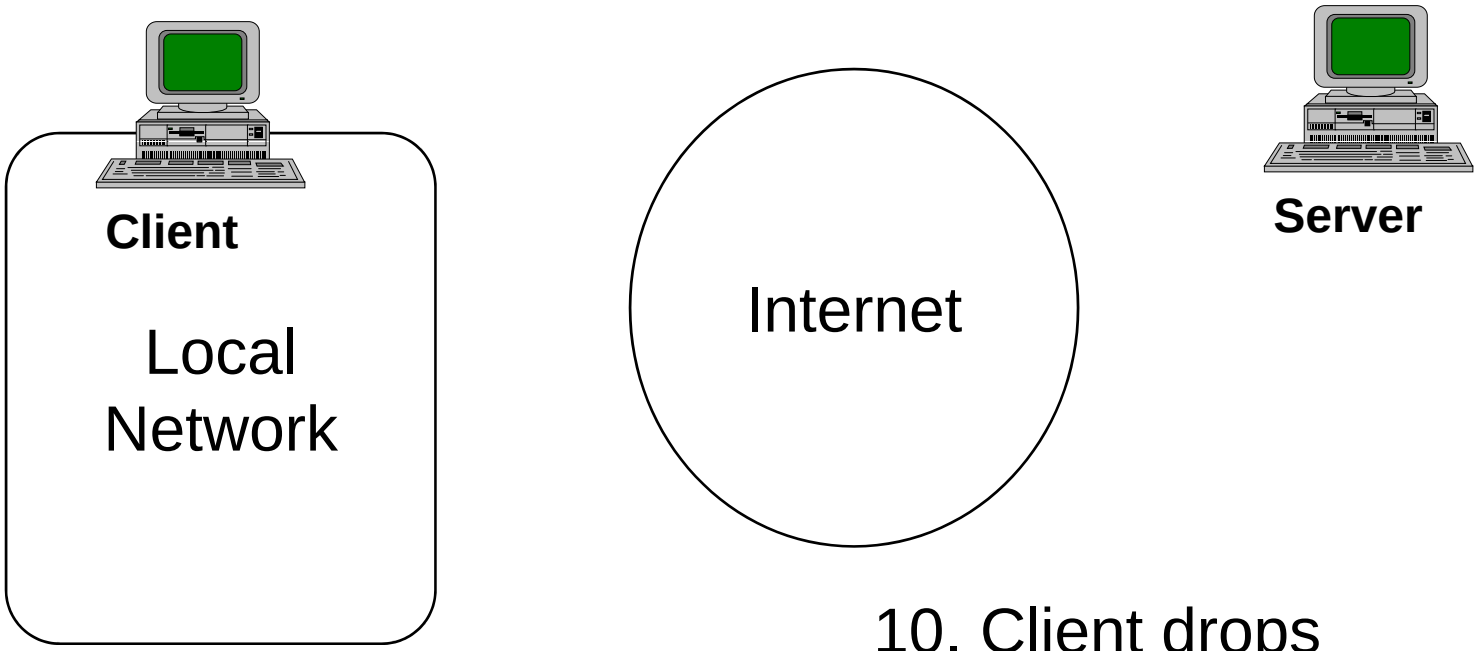
FTP: How it Works



FTP: How it Works



FTP: How it Works



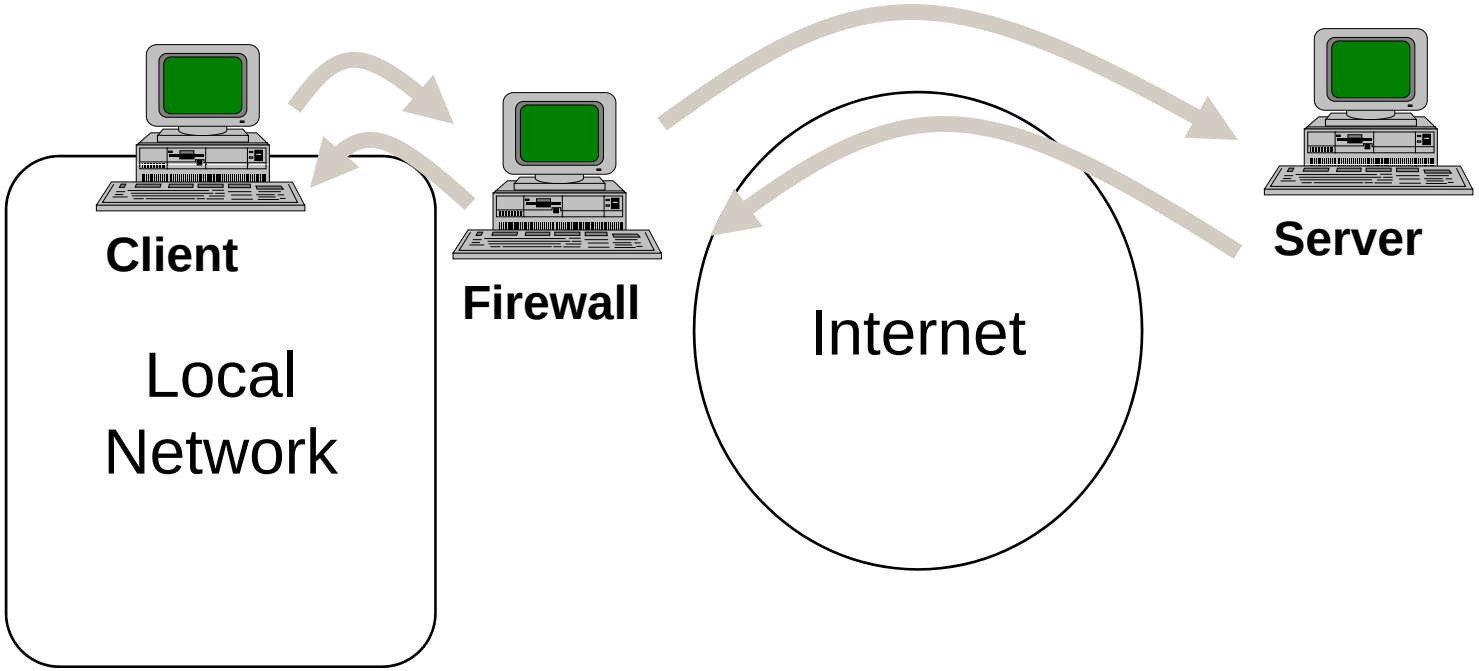
10. Client drops connection to server

Why that Matters

- In a word: firewalls
  - Existing router technology allowed basic blocking traffic based on origin and destination

  … But the “call back” where the remote server connects ***back into*** the originating client was outside of what the routers’ simple abilities could handle

Firewalls In The Middle



Firewalls: There's a Market

- 1991 First commercial firewall: $175,000
- 1994 Firewall industry: 3 vendors $12 million combined sales
- 1997 Firewall industry: 15 vendors $100 million
- 2009 Network Edge defense technologies: $1+billion (hard to even count)

What Is The Cost of a Nail?

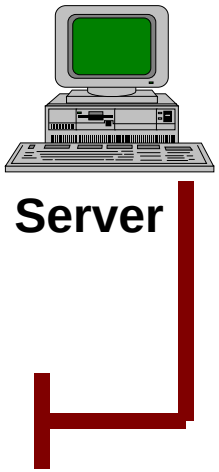
- It would have taken a good programmer two hours to fix FTP in 1975
- ***Hundred of millions*** of $ spent on firewalls between 1991 and 2009
- The problem is still there

... and ***so is FTP***

More about Sockets

- By the early 90's the Internet implementation was largely dominated by bsd/UNIX operating system and its derivatives
  - The network software layer in UNIX (aka: "the TCP/IP stack") is all basically the same code-base

Incoming Connection Limits

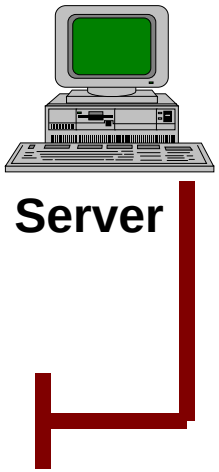


1. Server listens on socket port

Partial socket table (12 entries)

System socket table (2048 entries)

Incoming Connection Limits

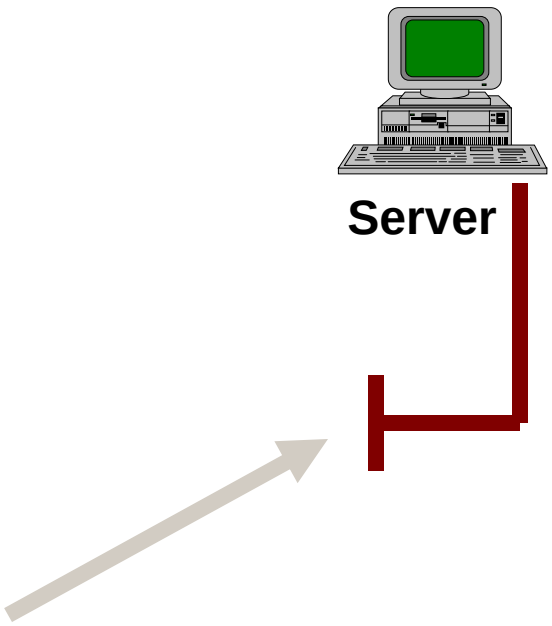


2. Incoming socket connection from remote client

Partial socket table (12 entries)

System socket table (2048 entries)

Incoming Connection Limits

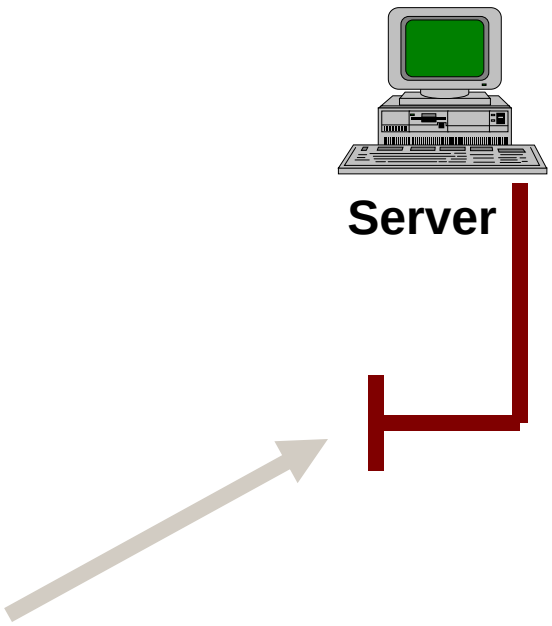


Partial socket table (12 entries)
11 left

3. Connection request is temporarily marked in partial socket table

System socket table (2048 entries)

Incoming Connection Limits



Partial socket table (12 entries)

4. Once connection is active it is moved to system socket table

System socket table (2048 entries)

What Does it Mean?

- 1995: Some versions of UNIX would crash and burn if they got too many connections at once
- Question: ***What else*** started happening in a big way around 1995?

World Wide Web

- In order to avoid the overloaded socket table problem, Tim Berners-Lee et al make HTTP protocol "stateless"
  - Each request is a separate connection
  - Accessing one web page might trigger 5, 10, 20 individual short-lived connections
  - The browser assembles all the responses into a coherent-looking document

Performance Hacks

- It turns out that making lots of short-lived connections is slow
  - So: browser coders hit on the clever idea: make 4 or 5 short-lived connections ***in parallel***!
  - Browsing gets much faster in return for higher load on the server and network

Incoming Connection Limits - 2

- Because of the load from many many short-lived connections hitting the servers, the socket management code is improved in IP stacks
  - Systems can now handle much larger (tens of thousands) of sockets, much faster

    I.e.: the ***original*** reason for doing short-lived connections is ***solved***

1997: La E-Deluge

- Internet commerce ***works!***

  ... Now, happens a very strange thing:
  - It turns out that “stateless” is not really so good

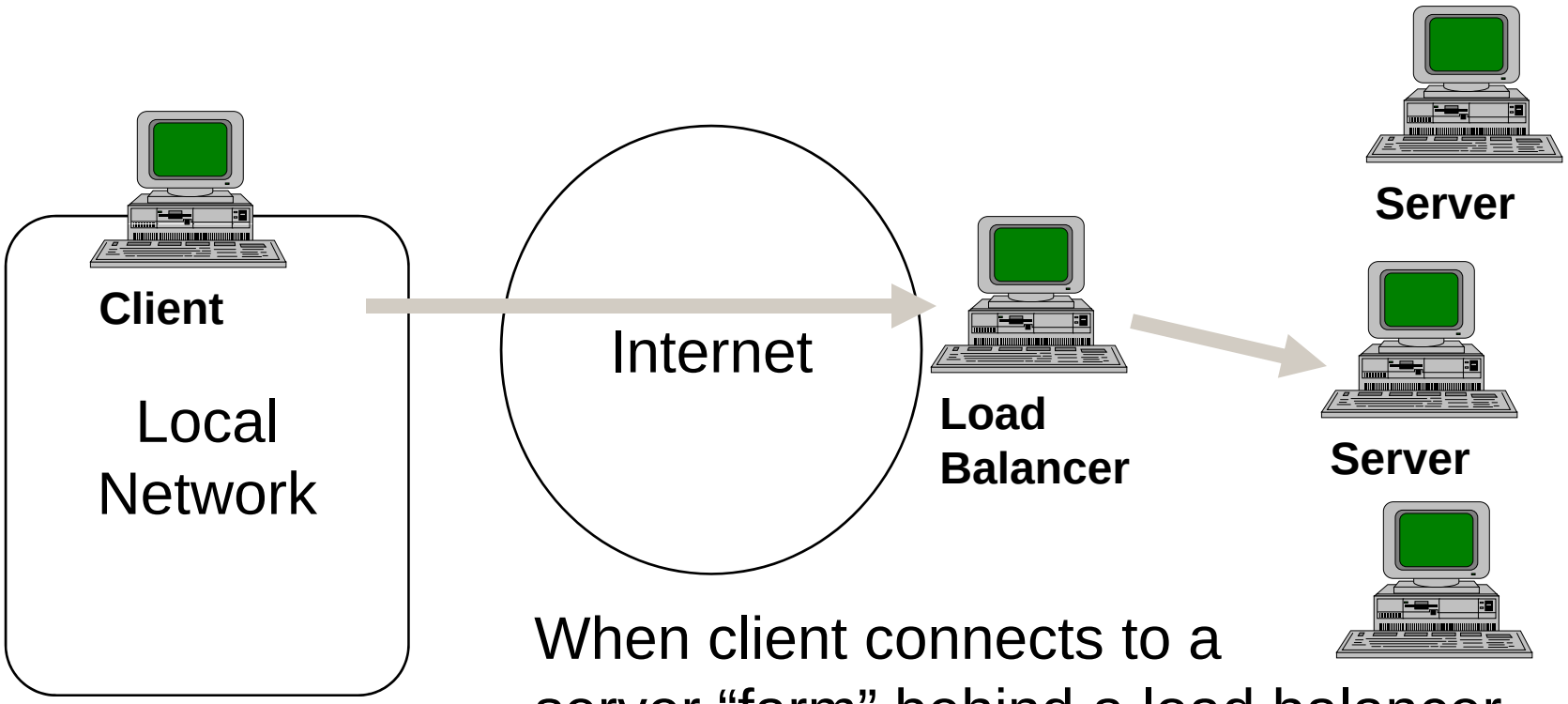
The State of StatelessNess

- Things that hold state:
  - Secure Sockets Layer (SSL)
  - Shopping carts
  - Website logins
  - Websites that keep track of what you've seen

L'Etat, C'est Moi!

- Software frameworks (PHP, Ruby, AJAX, .NET et al) all support models to re-introduce state in the form of "session management"
  - Uses a variety of cookies, tracking in the servers, etc
  - Programmers have to correctly code session management into their applications

State in a Box



When client connects to a server “farm” behind a load balancer, the load balancer must always direct the traffic to the same server to prevent breaking session state

The Bottom Line

- In order to ***re-achieve*** a capability that the underlying TCP/IP protocol ***already has***:
  - Hundreds of thousands of coder-hours will be spent by programmers who now have to deal with session management
  - Millions of dollars are being spent on load balancers and infrastructure that has extra “smarts” to handle state management

Failed States

- Some of those hundreds of thousands of coder-hours will be mis-spent
  - Thanks to flaws in the session management model we now have:
    - Session hijacking attacks
    - cross-site scripting
    - html injection attacks

Reminder

- The bug Berners-Lee et al went stateless to code around was ***fixed*** before the web went bigtime

Things You May Have Learned

- Software is a hugely connected enterprise: ***small mistakes over here can have huge impact elsewhere***
- Software evolves: ***intelligent design by an omniscient overseer - would be nice***
- Laziness + Genius + Momentum = ***unpredictably baroque systems***

What Matters

- Software is now becoming a major cost driver in most of the things humans build
  - We need to do better
  - We need to be much less concerned with “backwards compatibility”
  - We need to be much more aware of downstream consequences of ‘small’ design decisions