Testing in the Trenches

Wolf Richter

What are these?

- Apache
- Cherokee
- lighttpd
- nginx
- Unicorn
- Tornado
- gws

Okay, what is this?

Liso

Web servers!

Web servers!

So, what's used to test them, can also test this:

Web servers!

So, what's used to test them, can also test this:

Liso

Idea 1: Stress Testing Tools

- apachebench – concurrency, GETs, HEADs, POSTs, custom header data
- Siege – bit more configurable with URLs file
- Read up online, find tests for web servers
- Run them on your Liso server
- If they work, cool
- If not, check to see if Liso supports them
- If not, cool
- Otherwise, you have work to do

Idea 2: Real World Browsers

- Chrome
- Firefox
- Safari
- Opera
- Konqueror
- Internet Explorer
- ...

Idea 3: Python Scripting

- Let libraries do it: `import urllib2`
- Rolling your own test suite:
  - Craft requests in files
  - Send via Python sockets
  - Check returned bytes

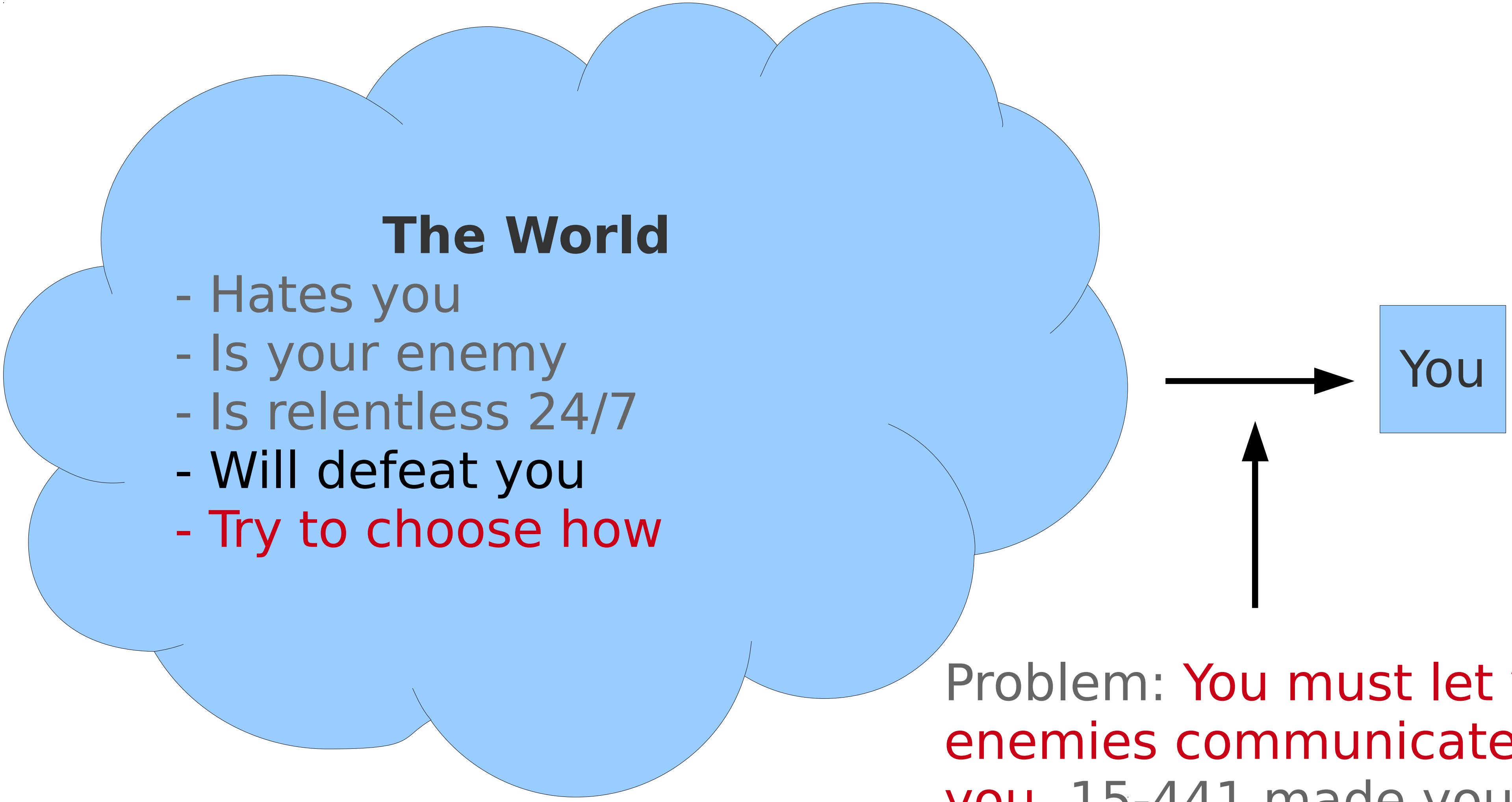
Testing: Think Evil, Be Evil

The World

- Hates you
- Is your enemy
- Is relentless 24/7
- Will defeat you
- Try to choose how

You

Testing: Think Evil, Be Evil



The World

- Hates you
- Is your enemy
- Is relentless 24/7
- Will defeat you
- Try to choose how

You

Problem: You must let your enemies communicate with you. 15-441 made you.

Networked Application Testing

- Always start with the previous picture
- Analyze interactions with The World
- That thin line in? port 80, or 443
- The World sends you bytes
- Think, what happens when I get:
  - Good bytes – designed to work normally
  - Arbitrarily bad bytes – designed to break me
  - Completely random bytes – !@#$()*&##($*)

Taint Analysis – Kinda

- We aren't formal, we don't care
  - Formal verification – would be nice
  - Also, can't explore every possibility, but...
- We want a back-of-the-envelope approach
  - Start with thought experiments
  - The World, the thin line in, and You
  - Then make these happen in real life
  - Leave absolutely nothing to chance
  - Know what your server will do in every case

Leaving Nothing to Chance

- Think outside the box, many scenarios
- Check especially corner cases
- If you expect 4096 sized buffers
    - You better be checking 4095
    - and 4097
    - And 8192+, 200MB+...different client apps...etc.
    - We already know The World will...
    - **Never, ever, ever expect something from:**
    - What you think, or code you read

Oh, I know what that does.
or, in that case my code will...

Oh, I know what that does.
or, in that case my code will...

How wonderful, you can compile, link, execute, and simulate clients with the x86 component of your human brain...OH WAIT!

No, hell no.

No, hell no.

You better make a minimal test case.
And then run it.

No, hell no.

You better make a minimal test case.
And then run it.

Know what will happen by making it happen.

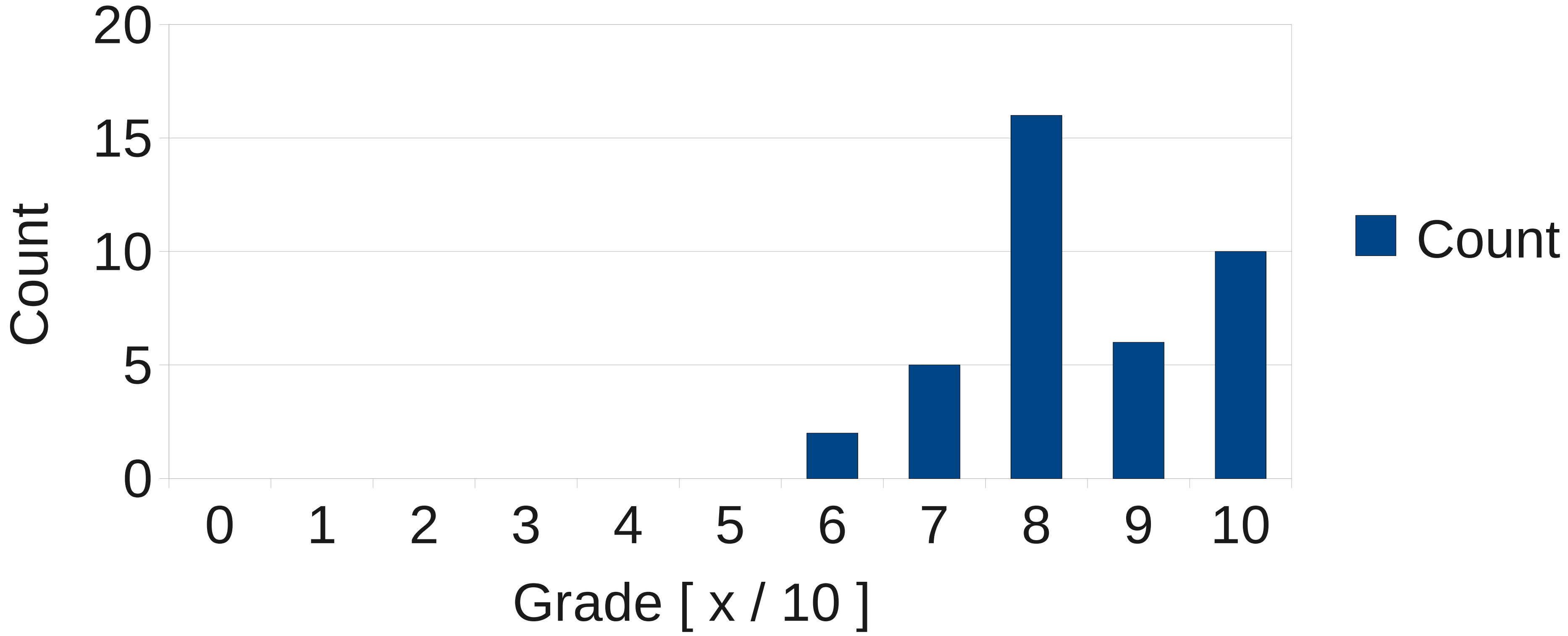
My advisor, Mahadev Satyanarayanan, is an Experimental Computer Scientist

I liked that about him. Maybe that says something about me as well.

So...

- This leads to robustness
- This leads to well-tested network code
- This leads to happy 100% in 15-441
- Think outside the box
- Test like crazy—as much as possible

Project 1 Checkpoint 1
Grade Distribution



The Wall of Shame

angx	15-441-project1
apodolsk	proj1
chunhowt	chunhowt-441-p1
ebreder	ebrederp1
hanl1	hanl1-p1-441
jchee	jchee-p1
jwloh	jwloh-p1
kailili	network-project1-kaili
moz	moz-project1
mengh	meng-project1
phoskins	phoskins-441-1
rggonzal	p1
siyoungo	siyoungo1
tbach	tbach-441-p1
xuanzhan	xuanzhan-p1
yueyuan	yy-441-proj1
zhuojil	zhuojil-p1

The Wall of Shame

angx	15-441-project1
apodolsk	proj1
chunhowt	chunhowt-441-p1
ebreder	ebrederp1
[illegible]	[illegible]
phoskins	phoskins-441-1
rggonzal	p1
siyoungo	siyoungo1
tbach	tbach-441-p1
xuanzhan	xuanzhan-p1
yueyuan	yy-441-proj1
zhuojil	zhuojil-p1

Review Reading Instructions:

“Name your project using this scheme (to avoid name collisions):
<andrewid>-15-441-project1”

Okay, we didn't detail the whole form, partly our fault; it was confusing :p

Leaderboard: Chaos Master

anandsur	00:17.92
chunhowt	00:18.18
ebreder	00:20.74
spradhan	00:21.06
adityaa1	00:22.26
kdalmia	00:22.97
abi	00:33.47
rggonzal	00:35.79
tbach	00:38.81
mteh	00:52.09

800 client connections;
random 50 write/read 32 Kibibytes;
two 5% chance disconnect events;
repeat for 100 trials

Leaderboard: BW King

abi	00:01.77
spradhan	00:01.79
anandsur	00:01.85
mteh	00:01.86
rggonzal	00:01.88
chunhowt	00:01.89
ebreder	00:01.89
tbach	00:01.98

minjaele replay.test 115.91 megabytes
Estimated: 3-6 memmove, disk write → 1.6 - 1.7 seconds

Numbers to Think About

- Select on 500 tcp fd's: 14.4491 microseconds
- Simple syscall: 0.2252 microseconds
- STREAM copy bandwidth: 3493.08 MB/sec
- Socket bandwidth using localhost: 2584.65 MB/sec
- Estimated disk write bandwidth: 79.1 MB/sec
- 116MB / 2584.65 MB/sec = .045 seconds (transfer)
- 116MB / 3493 MB/sec = 0.03 seconds (mem movement)
- *[3-6] = 0.09 – 0.18 seconds
- 116MB / 79.1 MB/sec = 1.47 seconds
- 1.47 + 0.045 + [0.09 – 0.18] = 1.6 – 1.7 seconds

Numbers to Think About

- Select on 500 tcp fd's: 14.4491 microseconds
- Simple syscall: 0.2252 microseconds

Thank you lmbench and dd.

- 116MB / 3493 MB/sec = 0.03 seconds (mem movement)
- *[3-6] = 0.09 – 0.18 seconds
- 116MB / 79.1 MB/sec = 1.47 seconds
- 1.47 + 0.045 + [0.09 – 0.18] = 1.6 – 1.7 seconds

Fork me on GitHub

GitHub:

Git it, got it, good.

```
git clone git://github.com/theonewolf/15-441-Recitation-Sessions.git
```