# Raw TCP/IP Socket Communication

DotNext St. Petersburg 2020

StephenCleary.com

# Who is this guy?


StephenCleary.com

O'REILLY®

Second Edition

# Concurrency in C# Cookbook

## Asynchronous, Parallel, and Multithreaded Programming

Stephen Cleary

# Who is this guy?

Raw TCP/IP communication with:

- Automatic Guided Vehicles
- Printing presses
- Hot backup systems
- GUI clients
- Bridge devices (TCP/IP to/from serial or unusual networks)
- Oil pipeline inspection tools
- Clients: GM, Syracuse News, Estee Lauder, RR Donnelley, Ricoh, BlueScope Steel. All 24x7 automated systems.

# Scope

The first rule of raw TCP/IP socket communication: Don't.

- HTTP / WebSockets / SOAP.
- SignalR / ASP.NET Core.

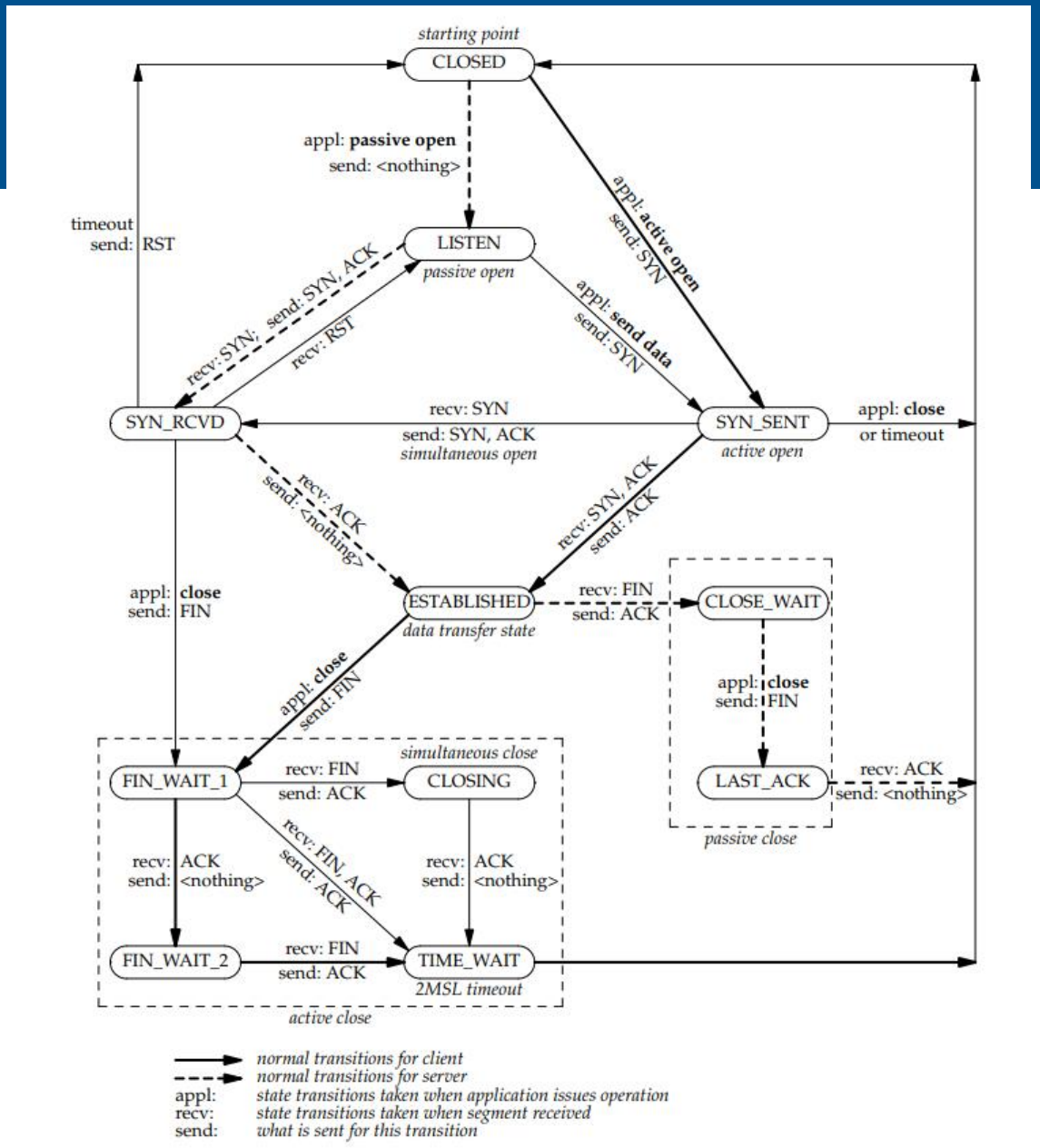
Raw TCP/IP sockets exist primarily for devices.

- Generic protocols are complex and difficult to implement on constrained hardware.
- Even when they can be implemented, they're inefficient.

Goal: Implement (or fix!) raw TCP/IP socket protocols.

# Out of Scope

- UDP
- Packet details (FIN/ACK/RST):
  - Stevens, Volume 1
  - Also state diagram from Volume 2
    - E.g., TIME_WAIT




StephenCleary.com

# Definition: “Application Protocol”

- All communication above the TCP/IP layer for a given application.
  - E.g.: POP, SMTP, IMAP, FTP, SSH
  - “Custom Application Protocol” – the details of how your apps communicate.

# TCP/IP Semantics

StephenCleary.com

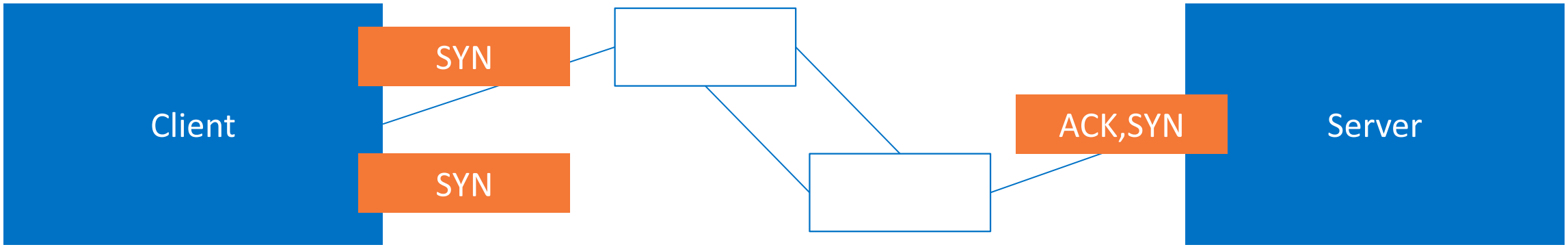
# What TCP/IP Provides

- “Connection” (but not whether a connection is viable)
  - Client connects to server.
  - Connected sockets don’t have an orientation.
- Reliability
  - Acknowledgements, Checksums, Retransmission, Discarding duplicates.
- Ordering
  - Packets sorted on receipt

# TCP/IP Abstraction: Connection

No “Known Connection State”

- Corollary: These members are useless: `Socket.Connected`, `TcpClient.Connected`



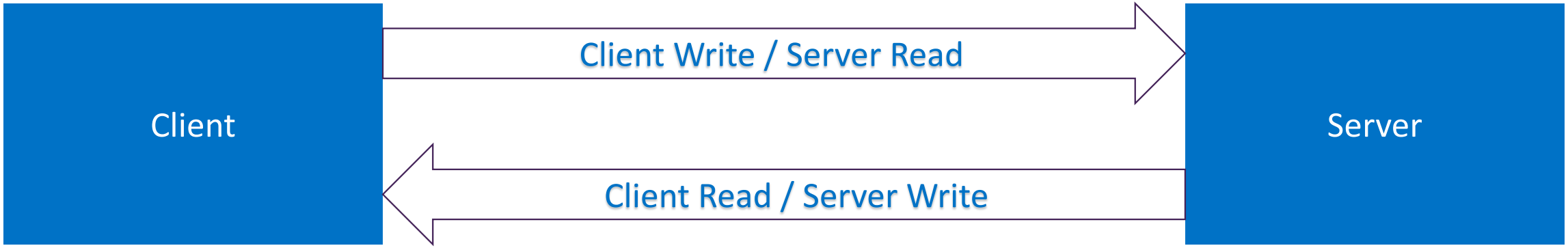
# TCP/IP Abstraction: Stream

TCP/IP provides a s packets.

There is *no way* to ket".

# TCP/IP Abstraction: Stream

Actually *two* independent streams: read and write.



# TCP/IP API

StephenCleary.com

# Berkeley API

I generally prefer Berkeley/WinSock or low-level wrappers

- I.e., Socket over TcpClient/TcpListener.
- This lets you more easily apply lessons learned from any Berkeley system.
- However, .NET types are unusually good quality.
- But any abstraction can introduce bugs, especially for unusual corner cases in complex systems.

But ignore the UDP parts.

- Remember: streams, not packets.

# Berkeley API – Common Mistakes

Writes:

- Complete when data is copied to the OS.
  - Successful write does *not* mean the other side got it. Or will *ever* get it.

Reads:

- Complete when data arrives or there is an error.
  - Successful read *does* mean these bytes are the next ones in the stream.
  - Be aware a read-completion of 0 bytes is the normal way to detect EOS.

# Designing Application Protocols for TCP/IP

StephenCleary.com

# Protocol Patterns

- Stream-based
  - E.g., audio/video transmission, file transfer.

- Message-based
  - Command/Response: one side sends Command; other side sends Response.
    - Can be bidirectional.
  - Poll/Status: a simple form of Command/Response.
  - Subscribe/Event: one side sends Subscribe/Unsubscribe; other side sends Event.

# Write It Down: Formal Specification

Even if it's not "your" protocol.

Formal documentation:

- MUST, MAY, and SHOULD from RFC 2119.
- Build on established standards.
  - Clarify: bytes vs characters (encoding), endianness.

First contact:

- Explicitly state which side is the server.

# Write It Down: Formal Specification

Choosing a port

- Configurable if possible.
- IANA: 0-1023 is off limits; use 1024-49151.
- Avoiding ephemeral port conflicts:
  - IANA (modern OSes): 49152 to 65535
  - Old Linux/Windows: 1025-5000

# Application Protocol Versioning

Enabling backwards compatibility is easier now than later!

Include version number in messages.

Active version should be negotiated, not assumed.

Don’t over-engineer; i.e., just send a list and choose one.

# What TCP/IP Provides (Review)

- "Connection" (but not whether a connection is viable)
  - Client connects to server.
  - Connected sockets don't have an orientation.
- Reliability
  - Acknowledgements, Checksums, Retransmission, Discarding duplicates.
- Ordering
  - Packets sorted on receipt

# What TCP/IP Does Not Provide

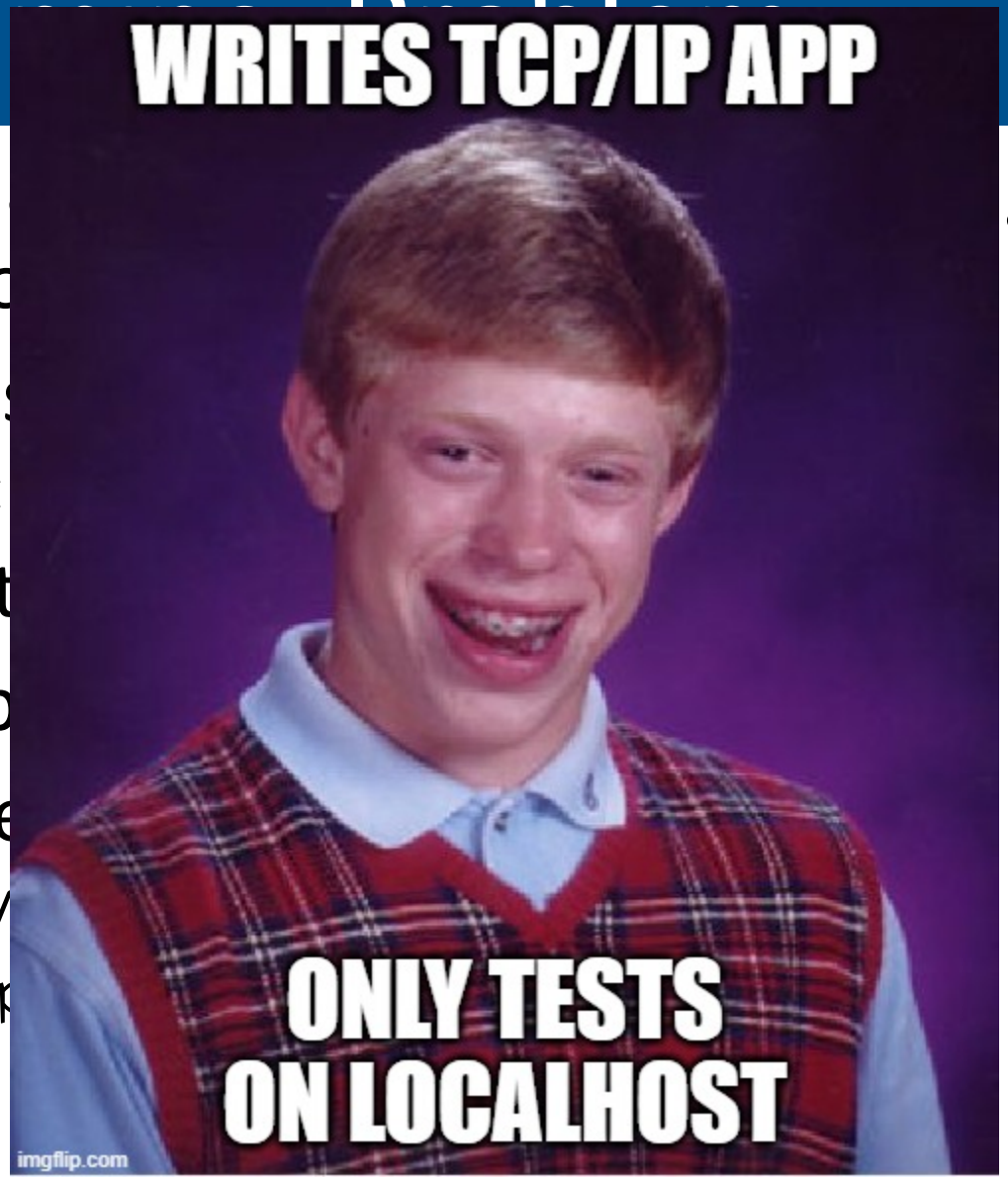
- Known Connection State
  - You cannot know whether a connection is viable.
- Message Boundaries
  - Your abstraction is a stream of bytes, not packets.

# Designing Application Protocols for TCP/IP: The Two Tricky Bits

StephenCleary.com

# Message Fra

- Most protocols h ge", e.g., Command, Respo
- TCP *does not* pres
  - Stream of bytes, not
- "Send": place byt
- "Receive": read b
- "Send" and "Rece
  - 1:N or N:1 or even M
  - Loopback testing imp




StephenCleary.com

# Message Framing: Solutions

Length Prefixing:

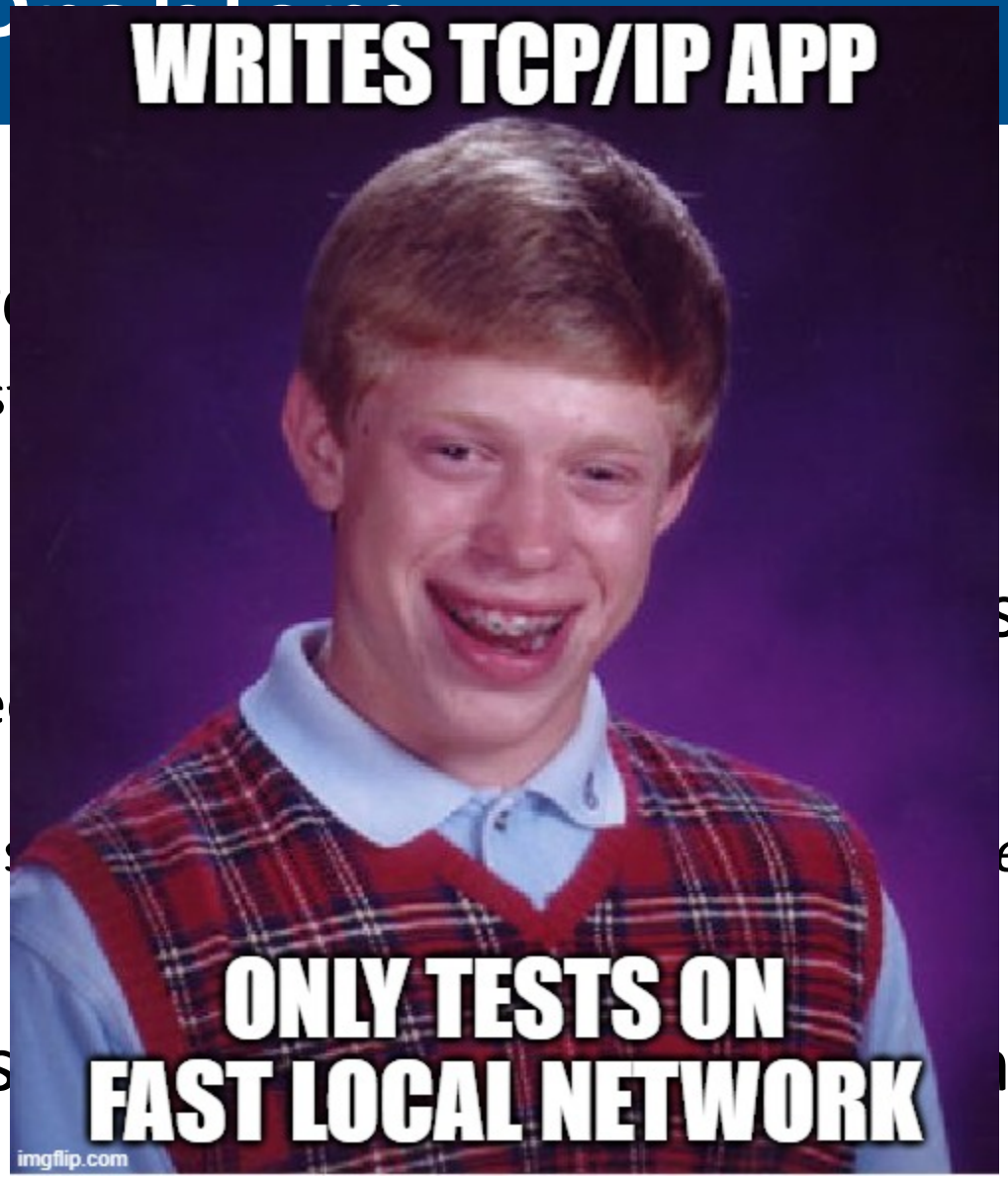
- Binary number indicating length of message.
- Document: length and endianness.

Delimiters:

- May require escaping message data to avoid delimiters.
- Requires flexible buffers on receiving side.

# Keepalives: P

- TCP is an “idle pr
  - No packets are trans
- TCP will detect a s sent.
  - Reliability guarantee
  - Most protocols only
    the sending side see *en*.
- Windows detects otifies apps!




StephenCleary.com

# Keepalives: Solution

- Wrong solutions: ping, second connection.

- Correct solution:
  - Periodic noop message (“ping” / “heartbeat” / “keepalive”)
  - TCP has a “keepalive” option, but:
    - Stack support, router support, 2 hours, systemwide settings.

- Periodic sends *must* be done on *both* sides.
  - If you *can’t* (i.e., it’s not your protocol), then you have to use an idle timer.

# Lessons Learned: HTTP

StephenCleary.com

# History of HTTP

HTTP 1.0

- One connection per request (!)
- Text-based
- Super flexible: Headers and Body
  - Header might specify body length (Content-Length)

Implementation:

- Requires very flexible buffering.
- Also requires an idle timer to detect idle connections.
- Very inefficient. Impossible for some devices.

# History of HTTP

HTTP 1.1

- Connections can be reused, but only one request at a time.
- Still text-based
- Still super flexible: Headers and Body
  - Header might specify body length (Content-Length)

Implementation:

- Still requires very flexible buffering.
- Still requires an idle timer to detect idle connections.
- Slightly more efficient. Still impossible for some devices.

# History of HTTP

HTTP 2

- Multiplexing: multiple simultaneous bidirectional requests.
- Binary
- Content-Length is optional but binary.
- Ping message to detect idle connections.

Implementation:

- Only requires flexible buffering if Content-Length is missing.
- Much more efficient. Possible for more devices.

# Tips

# Error Handling

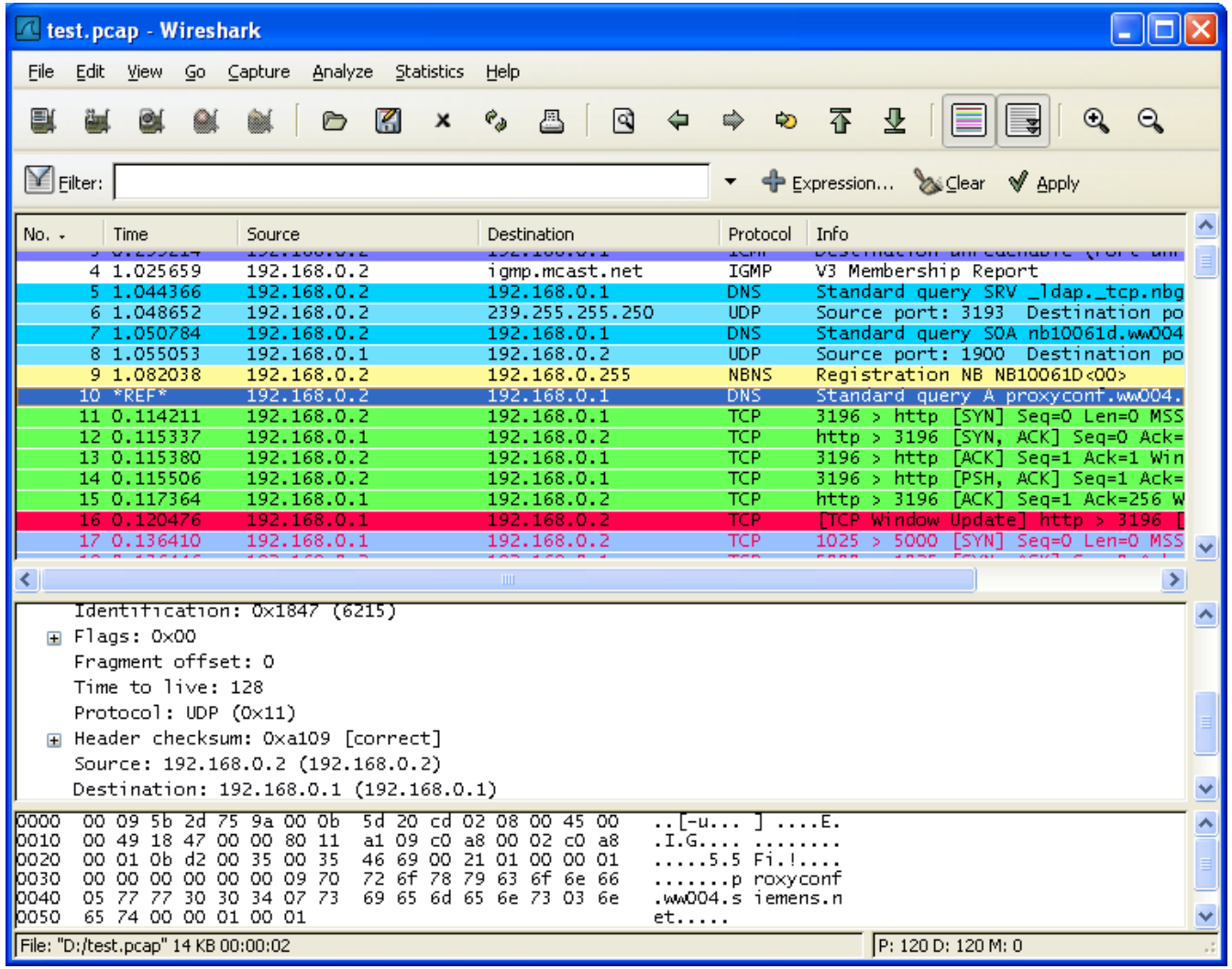
- All errors should close the connection.
  - Delay for some time before retrying.
- Allow abortive close.
  - If you're done communicating and you were about to close the connection anyway, then ignore any errors.
- Keep a continuous read operation to detect errors.
  - Writes complete successfully when data is copied to OS.

# Logging

- Have some way to log *everything*, from day 1.

# Inspection Tools: Wireshark

If you need Wireshark, also get TCP/IP Protocols Volume 1 (Stevens).

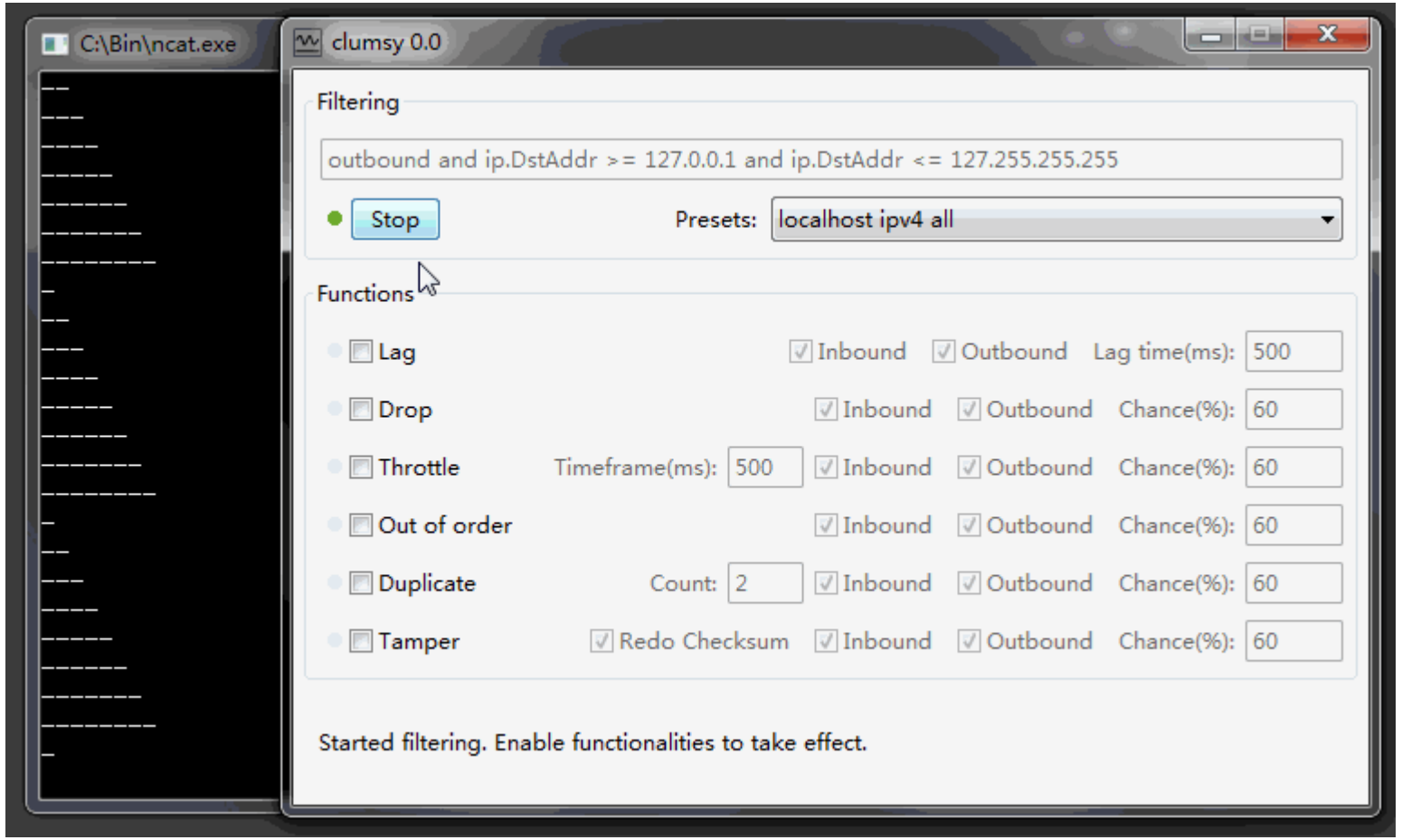



StephenCleary.com

# Testing Tools: Clumsy

Alternatives:

- TMNetSim
- dummynet




StephenCleary.com

# TL;DL

# The Key Information

## Design:

- Write the protocol down in a formal document.
- Message framing.

## Implementation:

- Always be reading
  - Detect errors ASAP.
- Periodically write
  - Detect half-open connections.
- Use asynchronous APIs for production servers
  - Thread per connection doesn't scale.

# Q&A Time!