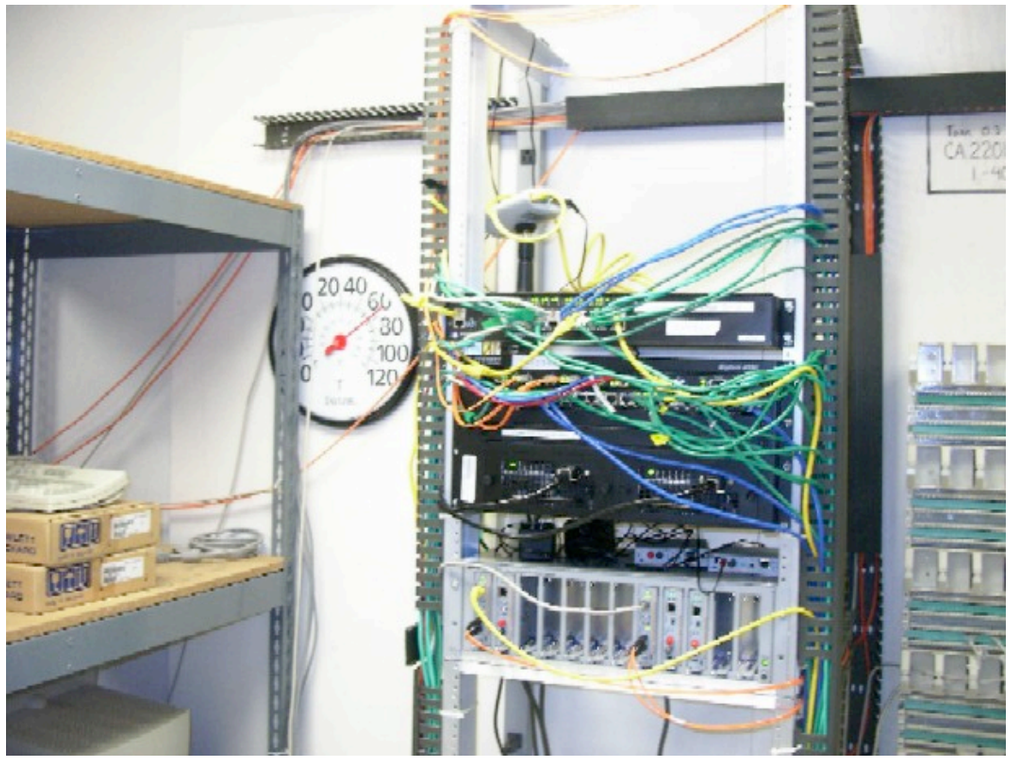
Network Protocol Testing in FreeBSD and in General



George Neville-Neil
FreeBSD Project
Neville-Neil Consulting

EuroBSD 2007

What is Protocol Testing?

- Network Protocols are hard to write
- Mistakes are often made
- How do we find mistakes in complex systems?
- Design various tests and carry them out over time

Why is protocol testing hard?

- Networked systems are non-deterministic
- Subject to message loss
- Time synchronization problems
- Requires lots of machines
  - Or instances of machines
- Knock on effects
- People find “systems thinking” hard
  - Which is why multi-threaded programming is hard

What measurements matter?

- Correctness
  - Do we get packet X when we expect it?
- Latency
  - How long does does a packet take to transit the system?
- Bandwidth
  - How many packets can we send per second?

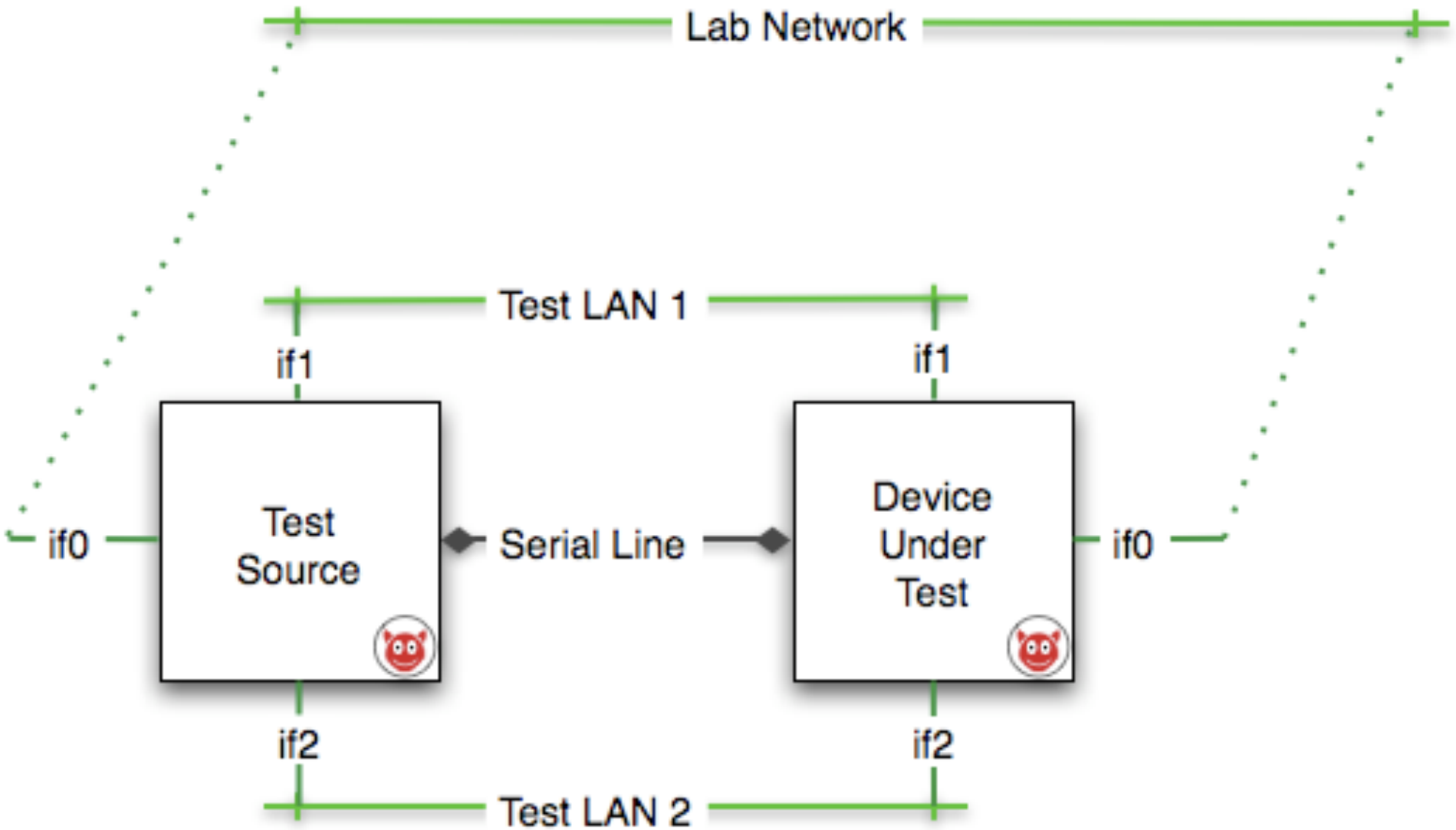
Some Types of Protocol Testing

- Specification Inspection
  - Finds faults in the design
  - Most powerful exploits are here
- Code Inspection
  - Finds faults by reading implementation
- Protocol Conformance Testing
  - Use real packets to ensure the implementation conforms to the spec
- Fuzzing Tools
  - Submit random junk in order to trigger a fault

Some Network and Protocol Tests

- Hey, it works!
  - "Hey, I just opened an HTTP connection to www.freebsd.org and I got the page!"
- ANVL, and NetBits
  - Commercial products costing tens of thousands of dollars
- TAHI
  - Open source from the Wide Projecn
  - Used for IPv6 and IPsec testing
- Netperf
  - Used for performance testing
- NetPIPE
  - A better performance tester

A Typical Test Rig



gnn@freebsd.org

Terminology

- Device Under Test (DUT)
  - The system you're testing
- Test Host (Controller)
  - The system that is sending and receiving test packets
- Test LAN (or Link)
  - The LAN across which test packets travel
  - Usually has to be quiescent
- Lab Network
  - The connection to the real world
  - Used to get new images, code, files etc.

The Problem

- Writing network protocol code is hard
- Testing network protocols is as hard as writing the protocol in the first place
- Most current systems are incomplete
  - Only support a small number of packets
  - Not extensible
  - Written in write-once languages
- Proprietary systems are expensive and incomplete
  - ANVL
  - SmartBits

What is a poor open sourcer to do?

- Write your own!
- Make it open source, of course!
- It can't be **that** hard, can it?

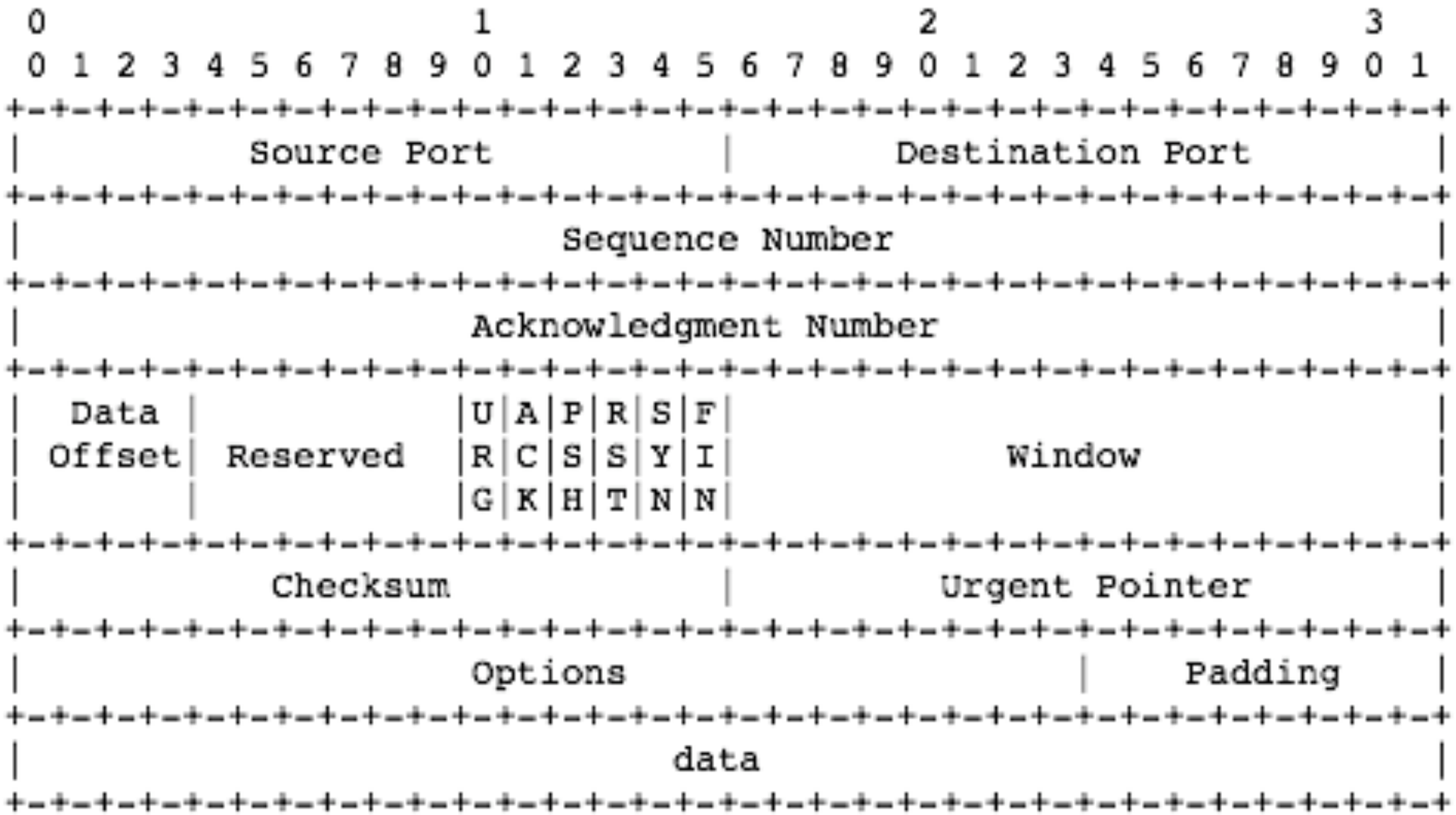
Packet Generation Tools in C

- Using the socket(2) API to generate specific packets is difficult to impossible
- The socket(2) API is designed with “normal” communication in mind
- Writing packets directly to bpf(4) is an error prone and lengthy process
- You wind up re-implementing most of the network stack
  - Which is > 100,000 lines just for IPv4!
- A middle ground is possible

Packet Construction Set

- A Python library for packet construction
- Easy creation of packets
- Gives a more natural syntax for packet manipulation
- BSD Licensed
  - http://pcs.sf.net

We need to get from this...



To this!

```
len = sizeof (struct ip) + tlen;
bzero(ipov->ih_x1, sizeof(ipov->ih_x1));
ipov->ih_len = (u_short)tlen;
ipov->ih_len = htons(ipov->ih_len);
th->th_sum = in_cksum(m, len);

if (th->th_sum) {
    tcpstat.tcps_rcvbadsum++;
    goto drop;
}
```


er, I mean this!

```
tcppacket.checksum = tcppacket.cksum(ip)
```


Advantages of PCS

- Easy to specify new packet formats
- Natural way of setting and getting packet fields
- Written in a well known language
  - Scripting languages are easier to “play” in
- Modular
- Well Documented

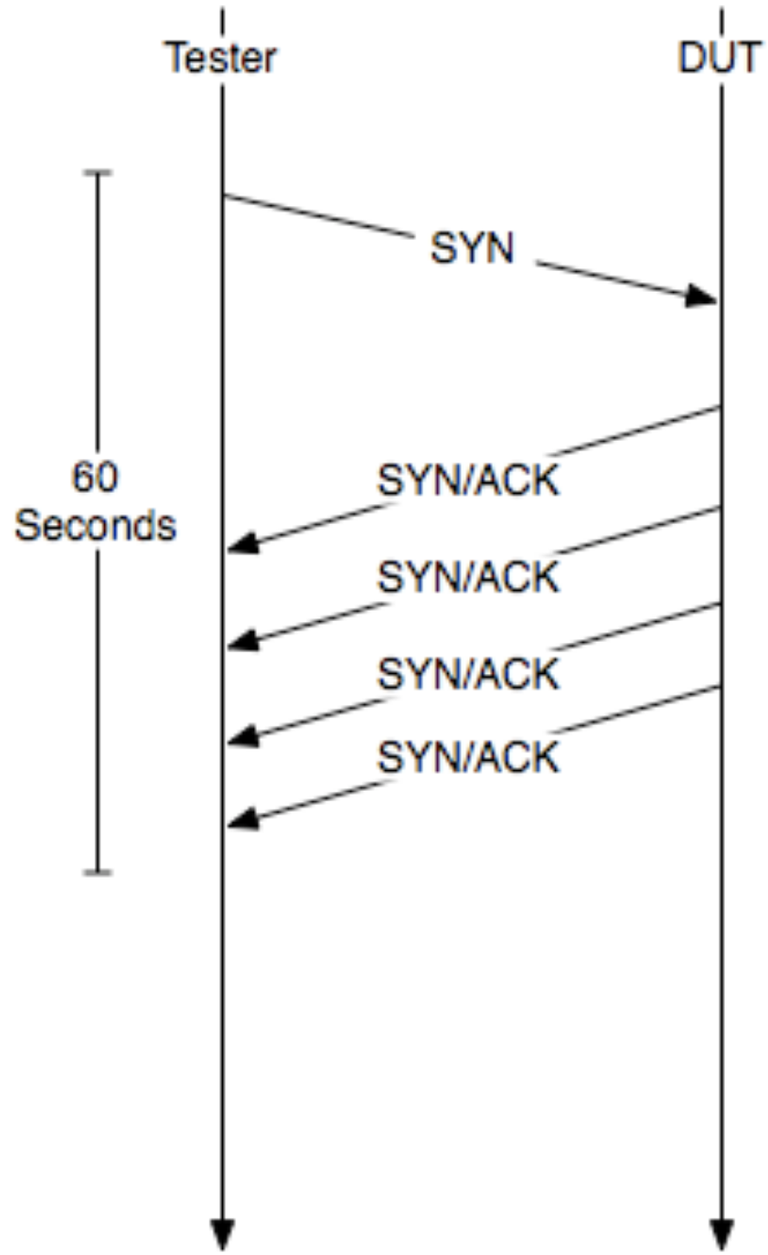
TCP Packet in PCS

```
sport = pcs.Field("sport", 16)
dport = pcs.Field("dport", 16)
seq = pcs.Field("sequence", 32)
acknum = pcs.Field("ack_number", 32)
off = pcs.Field("offset", 4)
reserved = pcs.Field("reserved", 6)
urg = pcs.Field("urgent", 1)
ack = pcs.Field("ack", 1)
psh = pcs.Field("push", 1)
rst = pcs.Field("reset", 1)
syn = pcs.Field("syn", 1)
fin = pcs.Field("fin", 1)
window = pcs.Field("window", 16)
checksum = pcs.Field("checksum", 16)
urgp = pcs.Field("urg_pointer",16)
```


Syncache timer test

- Tester sends a SYN packet to the DUT
- The DUT is supposed to put the SYN in its syncache
- The DUT then sends a SYN/ACK back to the Tester to continue the handshake process
- The syncache has a timer which controls how often to send the SYN/ACK
- The DUT **must** send 4 packets back to the Tester in under 60 seconds

A Network Time Diagram



Setting up IP

```
ip = ipv4()
ip.version = 4
ip.hlen = 5
ip.tos = 0
ip.id = 13000
ip.flags = 0
ip.offset = 0
ip.ttl = 64
ip.checksum = 0
ip.protocol = IPPROTO_TCP
ip.src = self.src
ip.dst = self.dst
```


Setting up TCP

```
tcppacket = tcp()
tcppacket.sport = self.sport
tcppacket.dport = self.dport
tcppacket.sequence = 42
tcppacket.offset = 5;
tcppacket.syn = 1
tcppacket.push = 0
tcppacket.window = 4096
tcppacket.checksum = 0
```


Setup Ethernet and Checksums

```
ether = ethernet()
ether.src = "\x00\x07\x43\x04\x01\x77"
ether.dst = "\x00\x17\x31\xef\x61\xdb"
ether.type = 2048

ip.length = len(ip.getbytes()) + len
(tcppacket.getbytes())
ip.checksum = ip.cksum()
tcppacket.checksum = tcppacket.cksum(ip)

packet = Chain([ether, ip, tcppacket])
```


Transmission

```
instream = PcapConnector("cxgb4")

instream.setfilter("ip");

output = PcapConnector("cxgb4")

out = output.write(packet.bytes,
len(packet.bytes))
```


Reception and Test

```
set_alarm(60)
syncount = 0

while syncount < 4 and alarm_triggered ==
0:
    newpkt = instream.readpkt()
    newpkt = newpkt.data.data
    if newpkt.dport == tcppacket.sport and
       newpkt.ack_number ==
       tcppacket.sequence + 1):
         syncount += 1

if alarm_triggered == 0:
    print "test passed"
```


Some Statistics

- PCS Test is less than 60 lines of code
- This test was written by Kip Macy in about 15 minutes
- A similar test in C was 600 lines
- Similar code in the kernel is hundreds of lines and not easy to extract or re-use

What makes a good test?

- Focused testing on a single variable
- Reproducible
- Understandable
- Easy to integrate with other tests

PCS Currently Supported Packets

- Ethernet, Loopback
- ARP
- IP (v4 and v6)
- ICMP, ICMPv6
- Neighbor Discovery
- AH, ESP (incomplete)
- UDP, TCP
- DNS, HTTP (incomplete)

Future Work

- More packets
  - 802.11
  - A more complete DNS
  - Appletalk
  - Better support of HTTP
- More tests
  - TCP Conformance Test Suite Project (kmacy@freebsd.org)
  - IPv6 and IPsec Conformance (TAHI replacment)
  - Fuzzers
- A more comprehensive test framework
- Integration to the FreeBSD regression tests
- More tools based on PCS
  - The Packet Debugger

Questions?

- Main Pages
  - http://pcs.sf.net
  - http://pktdbg.sf.net
- Mercurial Repo Access (Read Only)
  - http://www.neville-neil.com/hg/PCS
  - http://www.neville-neil.com/hg/PacketDebugger
- Ports
  - /usr/ports/net/py-pcs
- Contribute!
  - Patches, bugs, requests
  - https://sourceforge.net/projects/pcs/