The Packet Shell

Steve Parker

sparker@eng.sun.com

PSH

- Requirements
- Examples
- Architecture
- PSH Language
- Current State
- What Next?

Why a Packet shell?

- Conflicting requirements

- Host should always act “right”
- But if all hosts act “right”, never see how hosts react if a peer misbehaves

➡ Don’t want to use protocol stack to test protocol stack!

- Need to reproduce test cases

- Current TCP tests predicated on:
    - “If data gets thru, it works”
    - Maybe watch with snoop, but how many code paths actually exercised?
    - Replay from capture would be great!

Requirements

- Simple, interpretive language
- Natural expression per protocol
- Allow per-protocol extensions
  - e.g. IP address → hostname
- Don’t require everything to be specified

Requirements (cont.)

- Easy to construct packet from scratch
- Easy to modify existing packet
- Reasonably efficient
- End-user extensible
  - Allow new protocol interpreters at run time

Example

- Send two TCP SYNs
  With differing sequence numbers

Should see a TCP RST

Example Script

1222902.case1

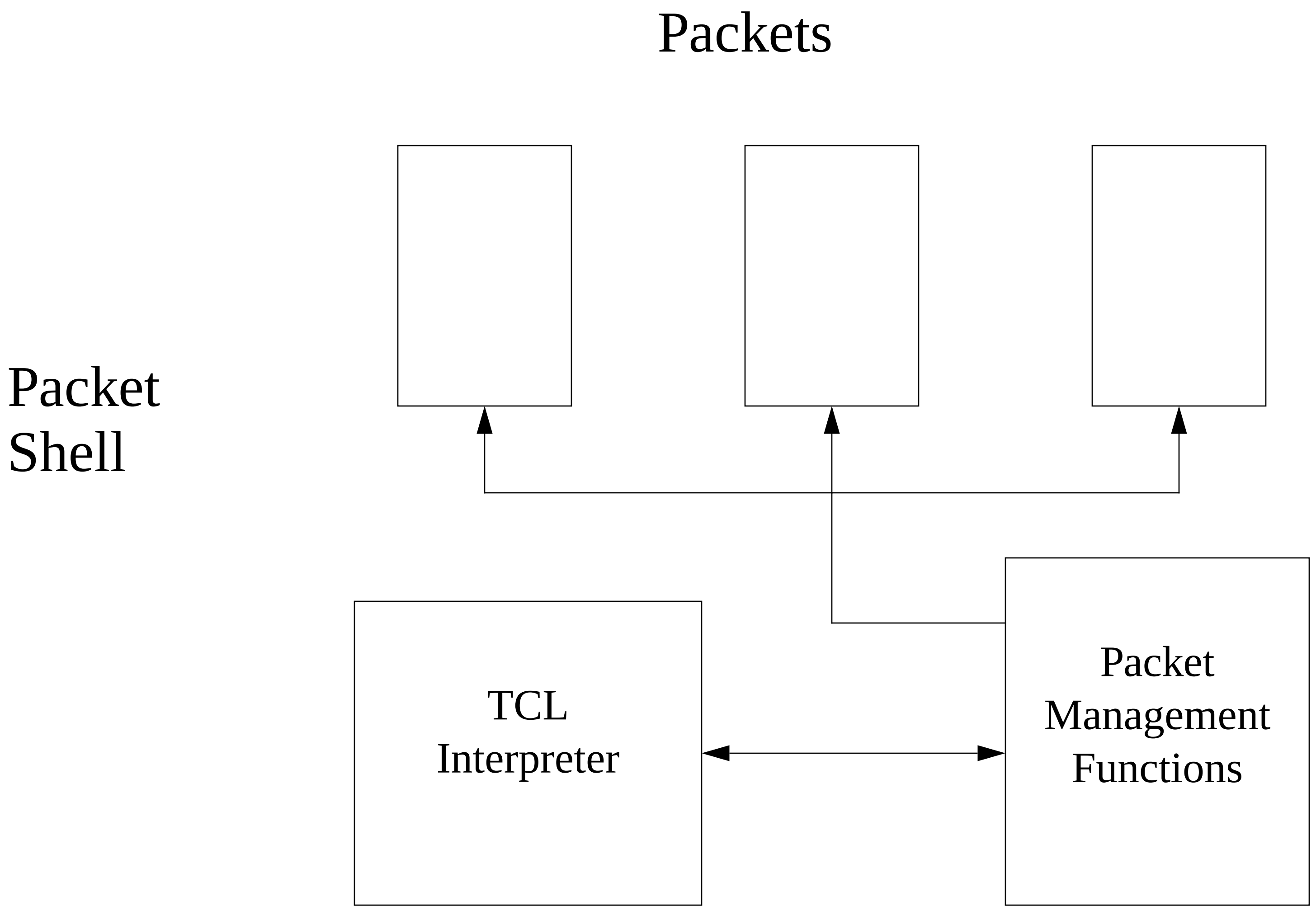
```
#
# Test that TCP survives (and should RST) a connection
# which sends two consequtive SYNs with different
# sequence numbers

# Open end-point 't' for communication
popen tcp t localhost 333 localhost echo

# Construct a prototype SYN packet
pinit tcp tcpsyn
pset tcpsyn tcp sport 333
pset tcpsyn tcp dport echo
pset tcpsyn tcp seq 55555
pset tcpsyn tcp flags SYN
pset tcpsyn tcp cksum

# Add TCP mss options
pset tcpsyn tcp addopt mss 0x5b4

# Initialize a packet to receive replies
pinit tcp tcprcv
```

```
# Send first SYN
t.send tcpsyn

# Now collect and print all packets until we go
# five seconds (5000ms) without anything further
while {[t.recv tcprcv 5000] == "packet"} {
      puts [plist tcprcv]
}

# Increment the sequence number by two
pset tcpsyn tcp seq [expr [pget tcpsyn tcp seq]+2]
pset tcpsyn tcp cksum

# Send the second SYN
t.send tcpsyn

# Report the next 10s worth of packets
while {[t.recv tcprcv 10000] == "packet"} {
      puts [plist tcprcv]
}

pclose tcp t
```

Packet Attributes

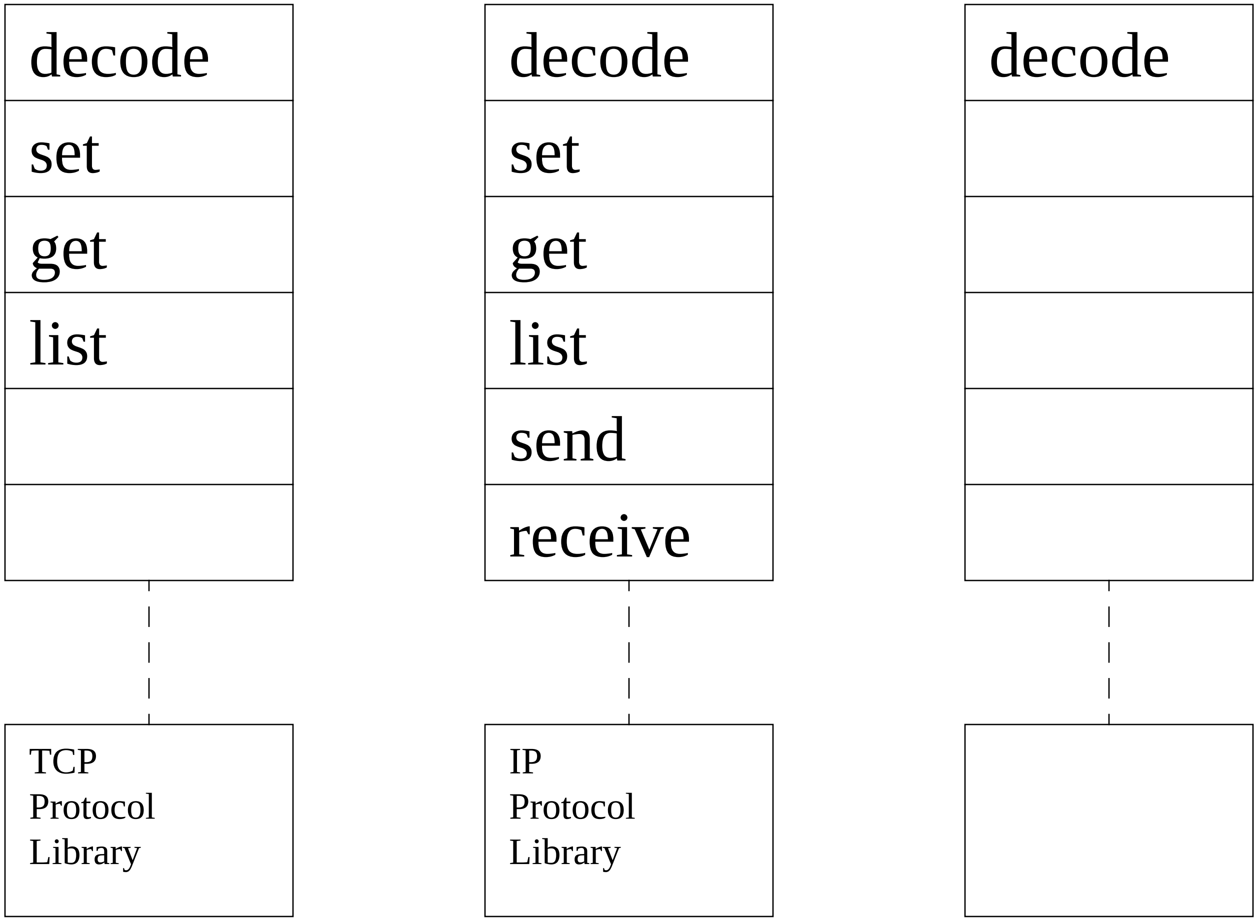
- length (r/w)
- protocol table (r/w)
  - List of (protocol, offset) pairs
- arrival time, if applicable (r/o) (TBD)

PSH Architecture



PSH Architecture

Protocol
Libraries



decode
set
get
list

TCP
Protocol
Library

decode
set
get
list
send
receive

IP
Protocol
Library

decode

Protocol Interpreter Interface

```
int  init(char *name, Tcl_Interp *interp);

int propen(ClientData clientData, Tcl_Interp *interp,
    int argc, char *argv[]);

int prclose(ClientData clientData, Tcl_Interp *interp,
    int argc, char *argv[]);

int pinit(ClientData clientData, Tcl_Interp *interp,
    struct packet *pckt, int argc, char *argv[]);

int pfree(ClientData clientData, Tcl_Interp *interp,
    struct packet *pckt, int argc, char *argv[]);
```

Protocol Interpreter Interface

```
int next_proto(struct packet *pckt, int element, char **name
    int *next_offset);


int get(int element, ClientData clientData, Tcl_Interp *interp,
    struct packet *pckt, int argc, char *argv[]);


int set(int element, ClientData clientData, Tcl_Interp *interp,
    struct packet *pckt, int argc, char *argv[];


int list(int element, ClientData clientData, Tcl_Interp *interp,
    struct packet *pckt, int argc, char *argv[]);
```

Packet Actions

- Packet buffer allocation

```
pinit   protocol   name    [proto-specific-options]
pfree    name
```

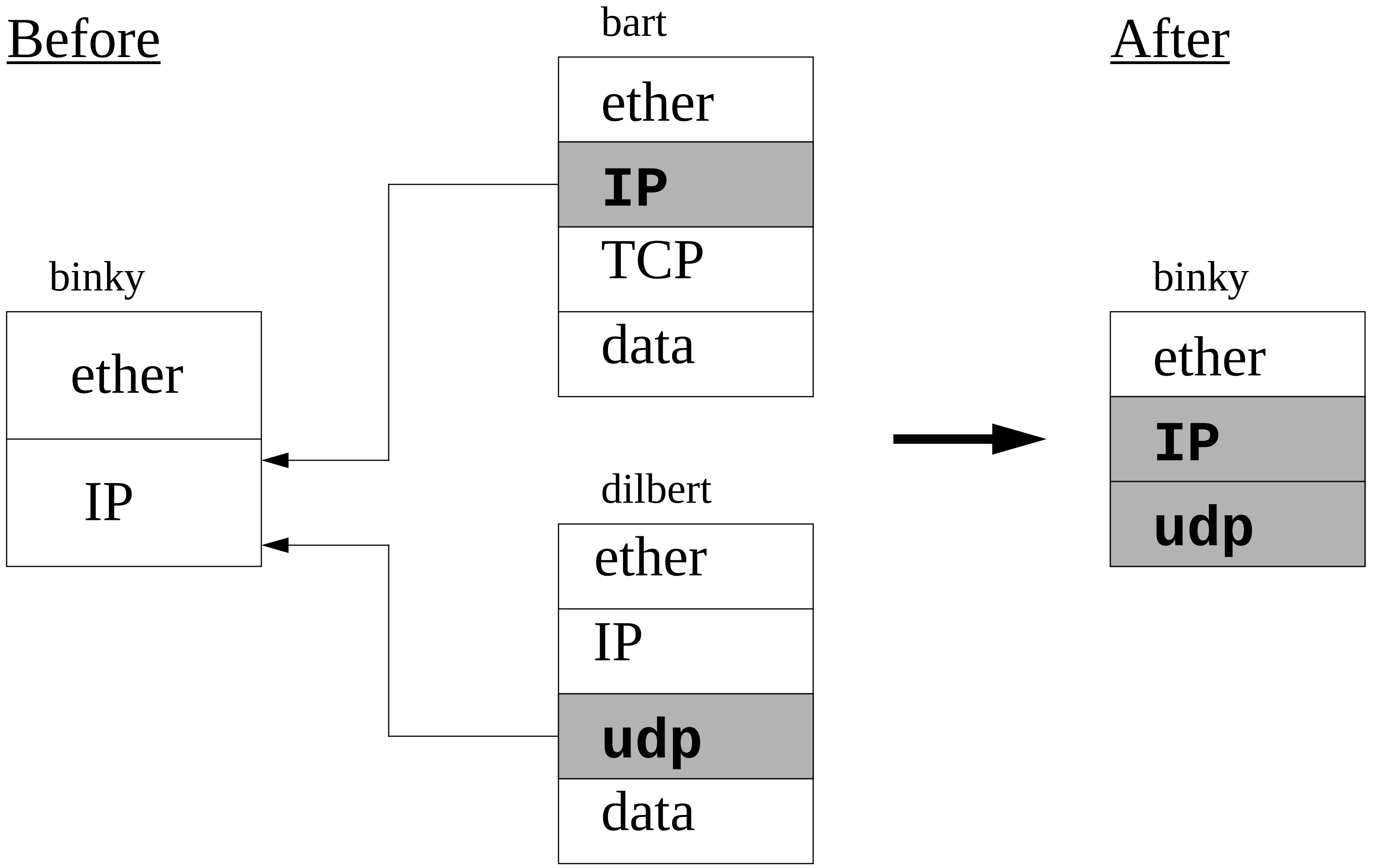
- Fetch or store byte/word/long

```
pread   pckt [b/w/l]  offset  [count]
pwrite  pckt [b/w/l]  offset  value...
```

- “Weave” packet from other packets

```
pcopy  dst-pkt/dst-layer src1/layer1 [src2/layer2...]
```

Example: `pcopy binky/ip  bart/ip  dilbert/udp`



Packet Table Operations

- Offset of specific protocol layer

```
poffset  name  prlayer
```

- Length of specific protocol layer

```
plen  name  prlayer
```

- Name of specific layer

```
pproto  name  prlayer
```

- Summarize packet table

```
ptbl  name
```

- Cast a layer to be a different protocol

```
pcast  name  prlayer  protocol
```

Per-Protocol Actions

- Print protocol summary

plist ***packet protocol***

- Print protocol element

pget ***packet protocol element***

- Set protocol element

pset ***packet*** **p*****rotocol*** **e*****lement value***

- Open a communication end-point

popen ***protocol descriptor protocol args...***

- Close a communication end-point

pclose ***protocol descriptor protocol args...***

Current State

- Packet re-sizing code not complete
- ‘pcast’ semantics still not solid
- Protocols implemented:
  - Ethernet link layer
  - IP
  - TCP
  - IP6
  - IP6 Fragment Headers
  - ICMP
  - ICMP6
  - Streams (not distributed)
  - “Data” pseudo-protocol
  - Snoop capture file “protocol”

Deep Questions

- Is “Sequential” the Right model?
  - Tk uses state/action paradigm...
- How well will fancier protocols work?
  - e.g., RPC
- How should timers be accomodated?

What Next?

Write more code...

- Use bufmod?
- “purify” the code
- “socket protocol” library
- Provide object-action style support
- Packet arrival time