Linux Transport Protocols Refactorings and State of DCCP on Linux

Conectiva Mandriva
acme@conectiva.com.br

Motivations for this work

- TCPfying the poor cousins reloaded

- Ease implementation of new AF_INET transport protocols

- Reduction of core network data structures

- Reduce the complexity of TCP code

- Use DCCP as a guinea pig for the refactorings

- Provide a test bed for Sally and Eddie

Transport Protocols Infrastructure - Past Work

- AF_INET
  sk->sk_prot

- IPv6
  struct or_calltable (minisocks)

- SCTP
  struct inet_protosw

Design for the current refactorings/DCCP stack

- Ease review of DCCP by TCP hackers

- Make it look like TCP as much as possible, same function names, same flow

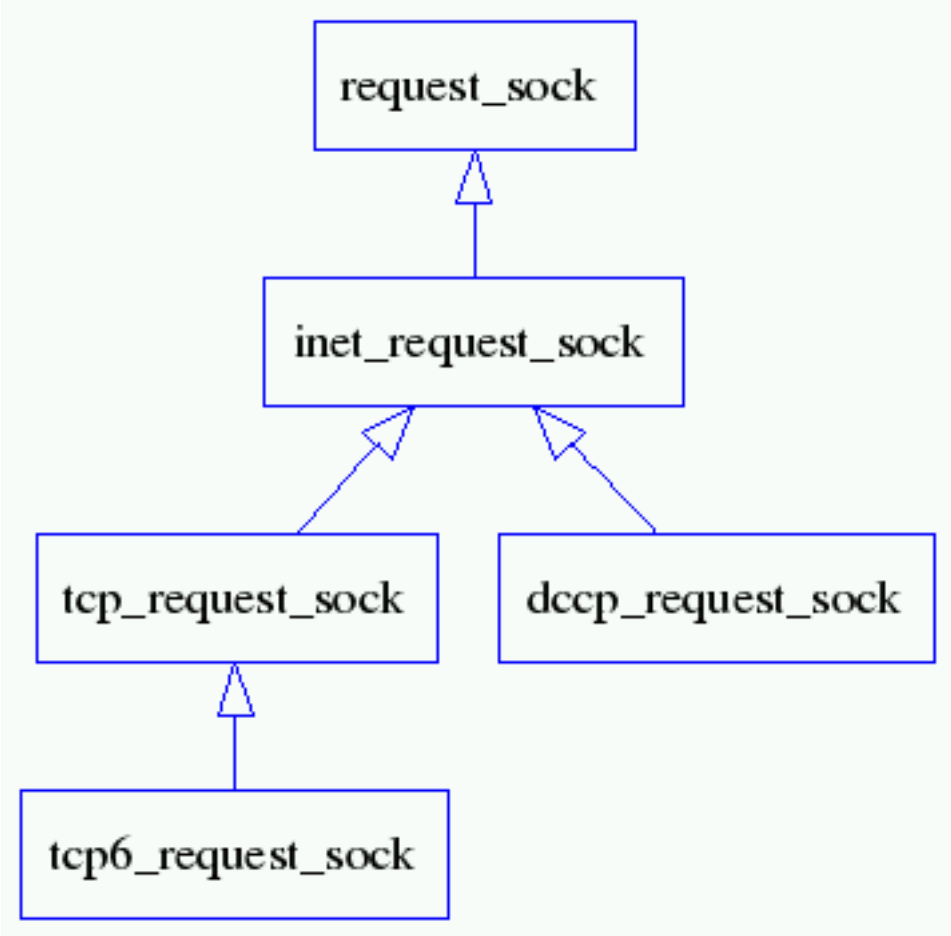
- Follow as close as possible the pseudocode in the DCCP draft

- Refactored code has to be as fast as before, faster if possible

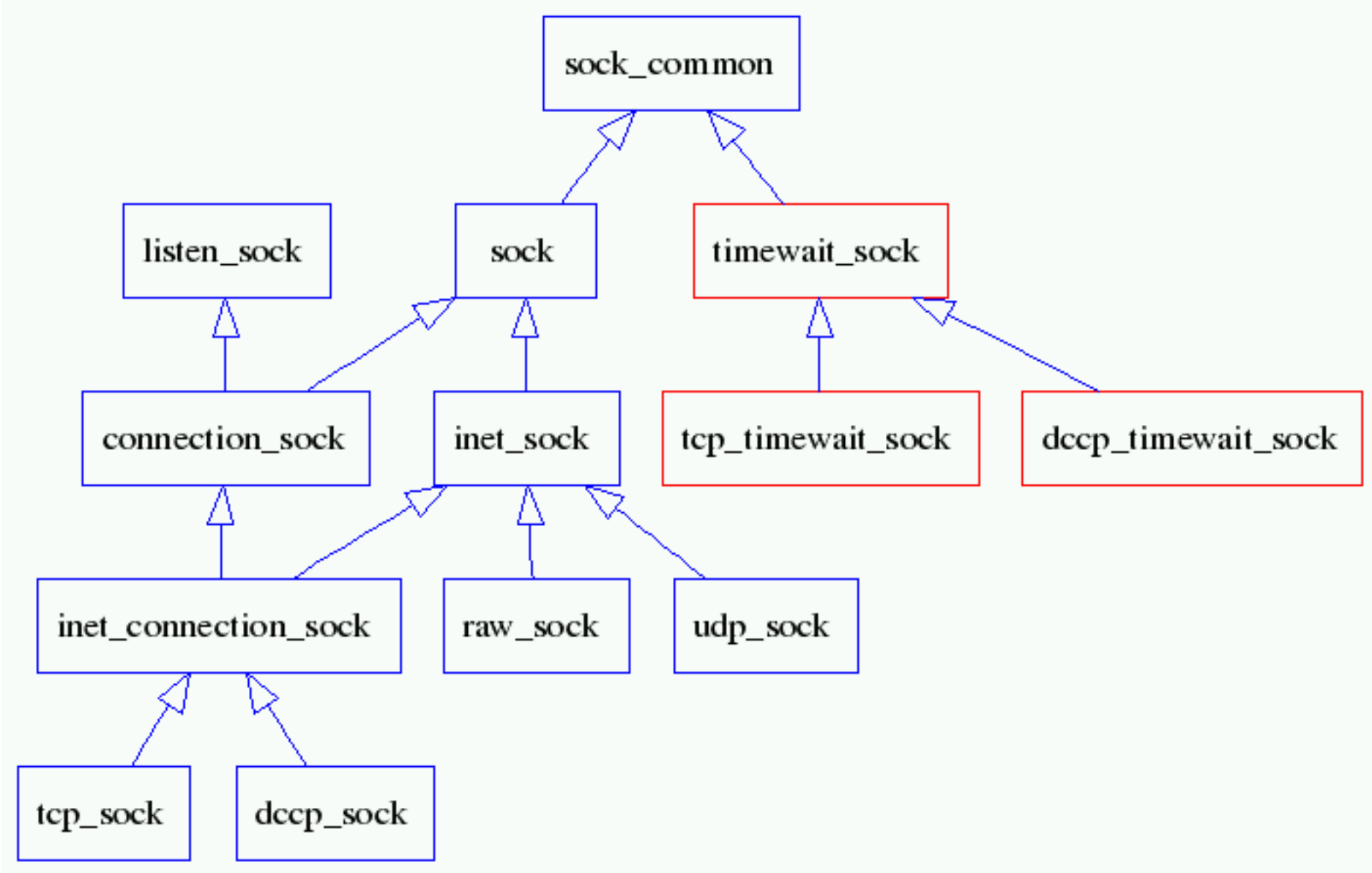
Connection mini socks

. struct open_request becomes request_sock
. struct or_calltable becomes request_sock_ops
. hierarchy similar to struct sock
. struct tcp_listen_opt becomes listen_sock
. reduces TCP v4 minisocks
. sk->sk_prot->rsk_prot: struct request_sock_ops
. proto_register allocates full sized sock and the minisock slabs
. already merged

Connection mini socks



struct sock hierarchy



Transition from request_sock to sock

. tcp_create_openreq_child
. turns a request_sock into a full sock
. sk_clone

Connection hash tables

. reuse tcp_hashinfo
. inet_sk_lookup
. inet_sk_lookup_listener
. __inet_sk_lookup_established

DCCP Basics

"It may be useful to think of DCCP as TCP minus bytestream semantics and reliability, or as UDP plus congestion control, handshakes, and acknowledgements." DCCP draft

. Unreliable, congestion controlled connections
. Packet types, not flags
. Packets, not bytes
. Selectable congestion control modules (CCIDs)
. Every packet has a 48bits seqno

DCCP packet types

. REQUEST
. RESPONSE
. ACK
. DATA/DATAACK
. CLOSE/CLOSEREQ
. RESET
. SYNC/SYNCACK

DCCP options implemented

. NDP - Non Data Packets
. ELAPSED_TIME
. TIMESTAMP/TIMESTAMP_ECHO
. ACK Vectors

ACK Vectors

ackno: 22 |0,0|3,0|0,0|3,0|0,0|3,0|0,0|3,0|0,1|

0, 0 means: DCCP_ACKPKTS_STATE_RECEIVED, RLE == just this packet
0, 1 means: DCCP_ACKPKTS_STATE_RECEIVED, RLE == two adjacent packets with the same state
3, 0 means: DCCP_ACKPKTS_STATE_NOT_RECEIVED, RLE == just this packet

22 was received
21 was not received
20 was received
19 was not received
18 was received
17 was not received
16 was received
15 was not received
14 was received
13 was received (this one was the 3way handshake RESPONSE)

CCIDs - Congestion Control Algorithms

. CCID2 - TCP-Like Congestion Control
. CCID3 - TCP Friendly Congestion Control (TFRC RFC 3448)

CCID Infrastructure

. Modelled after the CCID3 FreeBSD code needs
. Loadable modules

CCID3

. TFRC (RFC 3448)
. Equation Based - Avoid AIMD nature, less variations in available BW
. Implementation originally from FreeBSD (Lulea, Sweden)
. Collaboration with Waikato Universtity in New Zealand

Testing

ttcp
iproute2
iperf
libdnet
ostra

Future

. Merge TCP refactorings
. First batch already merged
. Implement CCID2
. Use Stephen’s TCP pluggable CA infrastructure
. Netfilter support
. Generalise tcp_tw_bucket